

# Boosting algorithms

Pavlo Mozharovskyi<sup>1</sup>

(with contributions of Laurent Rouviere<sup>2</sup> and Valentin Patilea<sup>3</sup>)

<sup>1</sup>LTCI, Télécom Paris, Institut Polytechnique de Paris

<sup>2</sup>Université Rennes 2

<sup>3</sup>Ensaï, CREST

Machine learning

Paris, March 12, 2022

# Today

## AdaBoost

- Idea and the algorithm

- Properties and examples

## Justification and generalization

- Minimization of the empirical risk

- Gradient boosting

- Tree-based gradient boosting

- LogitBoost

# Literature

**Learning materials** include but are not limited to:

- ▶ Hastie, T., Tibshirani, R., and Friedman, J. (2009).  
*The Elements of Statistics Learning: Data Mining, Inference, and Prediction (Second Edition)*.  
Springer.
  - ▶ Sections 10.{1, 4, 5, 9}.
- ▶ Slides of the lecture.
- ▶ Schapire, R. E. and Freund, Y. (2012).  
*Boosting: foundations and algorithms*.  
MIT Press.

# Binary supervised classification (reminder)

Notation:

- ▶ **Given:** for the random pair  $(X, Y)$  in  $\mathbb{R}^d \times \{-1, 1\}$  consisting of a random observation  $X$  and its random binary label  $Y$  (class), a sample of  $n$  i.i.d.:  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$ .
- ▶ **Goal:** predict the label of the new (unseen before) observation  $\mathbf{x}$ .
- ▶ **Method:** construct a classification rule:

$$g : \mathbb{R}^d \rightarrow \{-1, 1\}, \mathbf{x} \mapsto g(\mathbf{x}),$$

so  $g(\mathbf{x})$  is the prediction of the label for observation  $\mathbf{x}$ .

- ▶ **Criterion:** of the performance of  $g$  is the **error probability**:

$$R(g) = \mathbb{P}[g(X) \neq Y] = \mathbb{E}[\mathbb{1}(g(X) \neq Y)].$$

- ▶ **The best solution:** is to know the distribution of  $(X, Y)$ :

$$g(\mathbf{x}) = \text{sign}(2\mathbb{E}[Y|X = \mathbf{x}] - 1 > 0).$$

# Contents

## AdaBoost

- Idea and the algorithm

- Properties and examples

## Justification and generalization

- Minimization of the empirical risk

- Gradient boosting

- Tree-based gradient boosting

- LogitBoost

# Contents

## AdaBoost

- Idea and the algorithm

- Properties and examples

## Justification and generalization

- Minimization of the empirical risk

- Gradient boosting

- Tree-based gradient boosting

- LogitBoost

# The principle

The principle underlying the boosting:

- ▶ **First** boosting algorithm has been introduced by Yoav Freund and Robert E. Schapire in 1996.

# The principle

The principle underlying the boosting:

- ▶ **First** boosting algorithm has been introduced by Yoav Freund and Robert E. Schapire in 1996.
  
- ▶ Construct a **family of rules** (classifiers) and then aggregate them.



# The principle

The principle underlying the boosting:

- ▶ **First** boosting algorithm has been introduced by Yoav Freund and Robert E. Schapire in 1996.
- ▶ Construct a **family of rules** (classifiers) and then aggregate them.
- ▶ **Recursive** process:
  - the rule constructed on the  $k$ th step depends on the rule constructed on step  $k - 1$ .

# The principle

The principle underlying the boosting:

- ▶ **First** boosting algorithm has been introduced by Yoav Freund and Robert E. Schapire in 1996.
- ▶ Construct a **family of rules** (classifiers) and then aggregate them.
- ▶ **Recursive** process:
  - the rule constructed on the  $k$ th step depends on the rule constructed on step  $k - 1$ .
- ▶ Use **classification error** for both constructing the next rule and weighting the current one when aggregating.

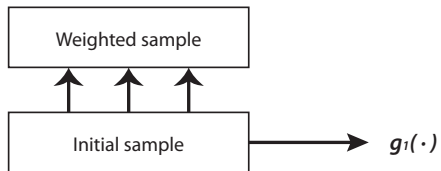
# The principle

Initial sample

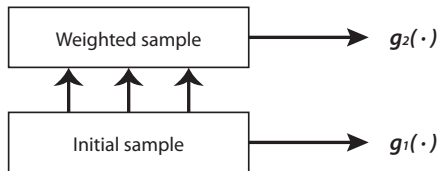
# The principle



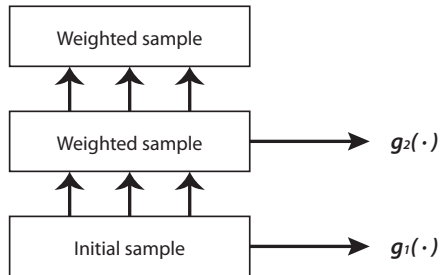
# The principle



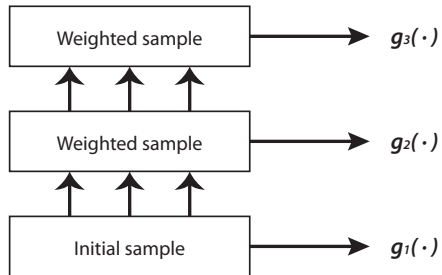
# The principle



# The principle

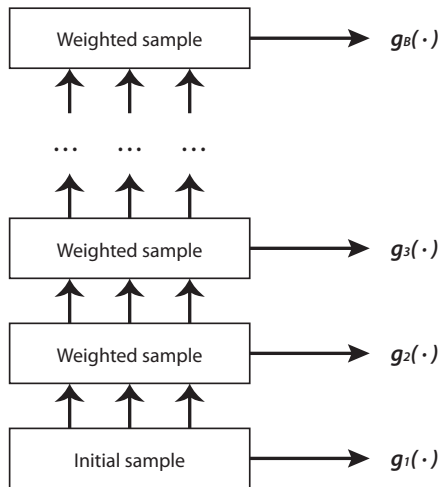


# The principle

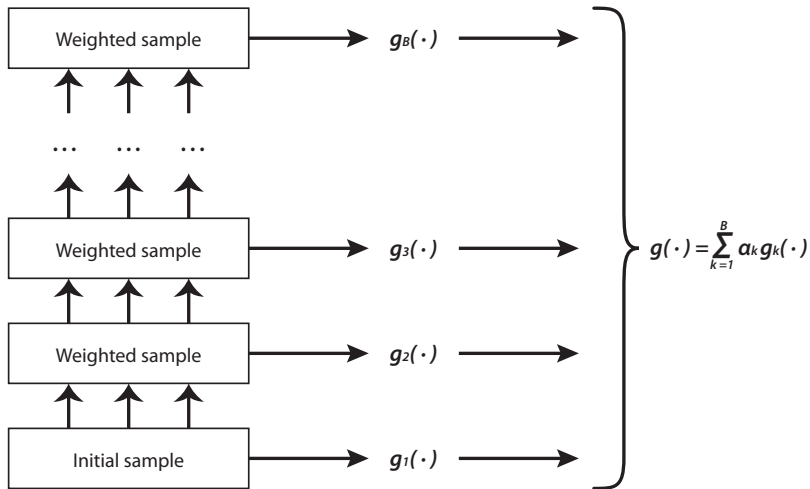




# The principle



# The principle



# The principle

- ▶ Term **boosting** is used to describe the family of methods allowing for construction of a precise rule based on weak learners.

# The principle

- ▶ Term **boosting** is used to describe the family of methods allowing for construction of a precise rule based on weak learners.
- ▶ A classification rule  $g(\cdot)$  is called a **weak learner** if it performs “slightly” better than a random guess:

$$\exists \epsilon > 0 \text{ such that } \mathbb{P}[g(X) \neq Y] = \frac{1}{2} - \epsilon.$$

# The principle

- ▶ Term **boosting** is used to describe the family of methods allowing for construction of a precise rule based on weak learners.
- ▶ A classification rule  $g(\cdot)$  is called a **weak learner** if it performs “slightly” better than a random guess:

$$\exists \epsilon > 0 \text{ such that } \mathbb{P}[g(X) \neq Y] = \frac{1}{2} - \epsilon.$$

- ▶ **Examples** of weak learners:

# The principle

- ▶ Term **boosting** is used to describe the family of methods allowing for construction of a precise rule based on weak learners.
- ▶ A classification rule  $g(\cdot)$  is called a **weak learner** if it performs “slightly” better than a random guess:

$$\exists \epsilon > 0 \text{ such that } \mathbb{P}[g(X) \neq Y] = \frac{1}{2} - \epsilon.$$

- ▶ **Examples** of weak learners:

- 1NN-classifier,

# The principle

- ▶ Term **boosting** is used to describe the family of methods allowing for construction of a precise rule based on weak learners.
- ▶ A classification rule  $g(\cdot)$  is called a **weak learner** if it performs “slightly” better than a random guess:

$$\exists \epsilon > 0 \text{ such that } \mathbb{P}[g(X) \neq Y] = \frac{1}{2} - \epsilon.$$

- ▶ **Examples** of weak learners:
  - 1NN-classifier,
  - classification tree of low depth (e.g. with 2 leaves only = a stump).

# AdaBoost (algorithm): Freund and Schapire (1997)

## Constructing a committee (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .



# AdaBoost (algorithm): Freund and Schapire (1997)

## Constructing a committee (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
- ▶ Weak learner  $g(\cdot)$ ,

# AdaBoost (algorithm): Freund and Schapire (1997)

## Constructing a committee (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
- ▶ Weak learner  $g(\cdot)$ , number of iterations  $B$ .

# AdaBoost (algorithm): Freund and Schapire (1997)

## Constructing a committee (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
  - ▶ Weak learner  $g(\cdot)$ , number of iterations  $B$ .
1. Initialize weights  $w_i = \frac{1}{n}$ ,  $i = 1, \dots, n$ .

# AdaBoost (algorithm): Freund and Schapire (1997)

## Constructing a committee (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
  - ▶ Weak learner  $g(\cdot)$ , number of iterations  $B$ .
1. Initialize weights  $w_i = \frac{1}{n}$ ,  $i = 1, \dots, n$ .
  2. For  $k = 1, \dots, B$

# AdaBoost (algorithm): Freund and Schapire (1997)

## Constructing a committee (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
  - ▶ Weak learner  $g(\cdot)$ , number of iterations  $B$ .
1. Initialize weights  $w_i = \frac{1}{n}$ ,  $i = 1, \dots, n$ .
  2. For  $k = 1, \dots, B$ 
    - 2.1 Learn a weak learner  $g_k(\cdot)$  on  $\mathcal{D}_n$  weighted by  $w_i$ ,  $i = 1, \dots, n$ .

# AdaBoost (algorithm): Freund and Schapire (1997)

## Constructing a committee (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
  - ▶ Weak learner  $g(\cdot)$ , number of iterations  $B$ .
1. Initialize weights  $w_i = \frac{1}{n}$ ,  $i = 1, \dots, n$ .
  2. For  $k = 1, \dots, B$ 
    - 2.1 Learn a weak learner  $g_k(\cdot)$  on  $\mathcal{D}_n$  weighted by  $w_i$ ,  $i = 1, \dots, n$ .
    - 2.2 Calculate error rate:

$$e_k = \sum_{i=1}^n w_i \mathbb{1}(g_k(\mathbf{x}_i) \neq y_i).$$

# AdaBoost (algorithm): Freund and Schapire (1997)

## Constructing a committee (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
  - ▶ Weak learner  $g(\cdot)$ , number of iterations  $B$ .
1. Initialize weights  $w_i = \frac{1}{n}$ ,  $i = 1, \dots, n$ .
  2. For  $k = 1, \dots, B$ 
    - 2.1 Learn a weak learner  $g_k(\cdot)$  on  $\mathcal{D}_n$  weighted by  $w_i$ ,  $i = 1, \dots, n$ .
    - 2.2 Calculate error rate:

$$e_k = \sum_{i=1}^n w_i \mathbb{1}(g_k(\mathbf{x}_i) \neq y_i).$$

- 2.3 Calculate learner's weight:  $\alpha_k = \log \frac{1-e_k}{e_k}$ .

# AdaBoost (algorithm): Freund and Schapire (1997)

## Constructing a committee (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
  - ▶ Weak learner  $g(\cdot)$ , number of iterations  $B$ .
1. Initialize weights  $w_i = \frac{1}{n}$ ,  $i = 1, \dots, n$ .
  2. For  $k = 1, \dots, B$ 
    - 2.1 Learn a weak learner  $g_k(\cdot)$  on  $\mathcal{D}_n$  weighted by  $w_i$ ,  $i = 1, \dots, n$ .
    - 2.2 Calculate error rate:

$$e_k = \sum_{i=1}^n w_i \mathbb{1}(g_k(\mathbf{x}_i) \neq y_i).$$

- 2.3 Calculate learner's weight:  $\alpha_k = \log \frac{1-e_k}{e_k}$ .
- 2.4 Readjust observations' weights:  $w_i = w_i e^{\alpha_k \mathbb{1}(g_k(\mathbf{x}_i) \neq y_i)}$ ,  $i = 1, \dots, n$ ;  
normalize to get  $\sum_{i=1}^n w_i = 1$ .



# AdaBoost (algorithm): Freund and Schapire (1997)

## Constructing a committee (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
  - ▶ Weak learner  $g(\cdot)$ , number of iterations  $B$ .
1. Initialize weights  $w_i = \frac{1}{n}$ ,  $i = 1, \dots, n$ .
  2. For  $k = 1, \dots, B$ 
    - 2.1 Learn a weak learner  $g_k(\cdot)$  on  $\mathcal{D}_n$  weighted by  $w_i$ ,  $i = 1, \dots, n$ .
    - 2.2 Calculate error rate:

$$e_k = \sum_{i=1}^n w_i \mathbb{1}(g_k(\mathbf{x}_i) \neq y_i).$$

- 2.3 Calculate learner's weight:  $\alpha_k = \log \frac{1-e_k}{e_k}$ .

- 2.4 Readjust observations' weights:  $w_i = w_i e^{\alpha_k \mathbb{1}(g_k(\mathbf{x}_i) \neq y_i)}$ ,  $i = 1, \dots, n$ ;  
normalize to get  $\sum_{i=1}^n w_i = 1$ .

**Output:** The committee  $g^{bst}(\cdot) = \text{sign}\left(\sum_{k=1}^B \alpha_k g_k(\cdot)\right)$ .

# AdaBoost (algorithm): Freund and Schapire (1997)

## Constructing a committee (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
  - ▶ Weak learner  $g(\cdot)$ , number of iterations  $B$ .
1. Initialize weights  $w_i = \frac{1}{n}$ ,  $i = 1, \dots, n$ .
  2. For  $k = 1, \dots, B$ 
    - 2.1 Learn a weak learner  $g_k(\cdot)$  on  $\mathcal{D}_n$  weighted by  $w_i$ ,  $i = 1, \dots, n$ .
    - 2.2 Calculate error rate:

$$e_k = \sum_{i=1}^n w_i \mathbb{1}(g_k(\mathbf{x}_i) \neq y_i).$$

- 2.3 Calculate learner's weight:  $\alpha_k = \log \frac{1-e_k}{e_k}$ .

- 2.4 Readjust observations' weights:  $w_i = w_i e^{\alpha_k \mathbb{1}(g_k(\mathbf{x}_i) \neq y_i)}$ ,  $i = 1, \dots, n$ ;  
normalize to get  $\sum_{i=1}^n w_i = 1$ .

**Output:** The committee  $g^{bst}(\cdot) = \text{sign}\left(\sum_{k=1}^B \alpha_k g_k(\cdot)\right)$ .

## Weighted majority voting (classification)

$$g^{bst}(\mathbf{x}) = \text{sign}\left(\sum_{k=1}^B \alpha_k g_k(\mathbf{x})\right).$$

# Contents

## AdaBoost

Idea and the algorithm

Properties and examples

## Justification and generalization

Minimization of the empirical risk

Gradient boosting

Tree-based gradient boosting

LogitBoost

## AdaBoost: some comments

- ▶ The step of learning requires the weak learner to take into account the **weights**  $w_i$ ,  $i = 1, \dots, n$  of observations.

## AdaBoost: some comments

- ▶ The step of learning requires the weak learner to take into account the **weights**  $w_i$ ,  $i = 1, \dots, n$  of observations.
- ▶ If weights cannot be taken into account, the weak learner can be trained on a **subsample** of  $\mathcal{D}_n$  drawn (with replacement) with probabilities  $w_i$ ,  $i = 1, \dots, n$ .

## AdaBoost: some comments

- ▶ The step of learning requires the weak learner to take into account the **weights**  $w_i$ ,  $i = 1, \dots, n$  of observations.
- ▶ If weights cannot be taken into account, the weak learner can be trained on a **subsample** of  $\mathcal{D}_n$  drawn (with replacement) with probabilities  $w_i$ ,  $i = 1, \dots, n$ .
- ▶ Observations' weights  $w_i$ ,  $i = 1, \dots, n$  are updated on each iteration:

## AdaBoost: some comments

- ▶ The step of learning requires the weak learner to take into account the **weights**  $w_i$ ,  $i = 1, \dots, n$  of observations.
- ▶ If weights cannot be taken into account, the weak learner can be trained on a **subsample** of  $\mathcal{D}_n$  drawn (with replacement) with probabilities  $w_i$ ,  $i = 1, \dots, n$ .
- ▶ Observations' weights  $w_i$ ,  $i = 1, \dots, n$  are updated on each iteration:
  - if the  $i$ th observation is **well classified**, then its weight is unchanged,

## AdaBoost: some comments

- ▶ The step of learning requires the weak learner to take into account the **weights**  $w_i$ ,  $i = 1, \dots, n$  of observations.
- ▶ If weights cannot be taken into account, the weak learner can be trained on a **subsample** of  $\mathcal{D}_n$  drawn (with replacement) with probabilities  $w_i$ ,  $i = 1, \dots, n$ .
- ▶ Observations' weights  $w_i$ ,  $i = 1, \dots, n$  are updated on each iteration:
  - if the  $i$ th observation is **well classified**, then its weight is unchanged,
  - if the  $i$ th observation is **wrongly classified**, then its weight is increased.



## AdaBoost: some comments

- ▶ The step of learning requires the weak learner to take into account the **weights**  $w_i$ ,  $i = 1, \dots, n$  of observations.
- ▶ If weights cannot be taken into account, the weak learner can be trained on a **subsample** of  $\mathcal{D}_n$  drawn (with replacement) with probabilities  $w_i$ ,  $i = 1, \dots, n$ .
- ▶ Observations' weights  $w_i$ ,  $i = 1, \dots, n$  are updated on each iteration:
  - if the  $i$ th observation is **well classified**, then its weight is unchanged,
  - if the  $i$ th observation is **wrongly classified**, then its weight is increased.
- ▶ Learner's weight  $\alpha_k$  **increases with performance of**  $g_k(\cdot)$  calculated on (weighted)  $\mathcal{D}_n$ :

## AdaBoost: some comments

- ▶ The step of learning requires the weak learner to take into account the **weights**  $w_i$ ,  $i = 1, \dots, n$  of observations.
- ▶ If weights cannot be taken into account, the weak learner can be trained on a **subsample** of  $\mathcal{D}_n$  drawn (with replacement) with probabilities  $w_i$ ,  $i = 1, \dots, n$ .
- ▶ Observations' weights  $w_i$ ,  $i = 1, \dots, n$  are updated on each iteration:
  - if the  $i$ th observation is **well classified**, then its weight is unchanged,
  - if the  $i$ th observation is **wrongly classified**, then its weight is increased.
- ▶ Learner's weight  $\alpha_k$  **increases with performance of**  $g_k(\cdot)$  calculated on (weighted)  $\mathcal{D}_n$ :
  - $\alpha_k$  increases with decreasing  $e_k$ ;

## AdaBoost: some comments

- ▶ The step of learning requires the weak learner to take into account the **weights**  $w_i$ ,  $i = 1, \dots, n$  of observations.
- ▶ If weights cannot be taken into account, the weak learner can be trained on a **subsample** of  $\mathcal{D}_n$  drawn (with replacement) with probabilities  $w_i$ ,  $i = 1, \dots, n$ .
- ▶ Observations' weights  $w_i$ ,  $i = 1, \dots, n$  are updated on each iteration:
  - if the  $i$ th observation is **well classified**, then its weight is unchanged,
  - if the  $i$ th observation is **wrongly classified**, then its weight is increased.
- ▶ Learner's weight  $\alpha_k$  **increases with performance of**  $g_k(\cdot)$  calculated on (weighted)  $\mathcal{D}_n$ :
  - $\alpha_k$  increases with decreasing  $e_k$ ;
  - anyway,  $g_k(\cdot)$  **should not be "too weak"**: if  $e_k > \frac{1}{2}$  then  $\alpha_k < 0$ .

# AdaBoost: some properties

- ▶ The **error rate calculated for the training sample**  $e_k$  of the learner  $g_k$  is given by:

$$e_k = \frac{\sum_{i=1}^n w_i \mathbb{1}(g_k(\mathbf{x}_i) \neq y_i)}{\sum_{i=1}^n w_i} .$$

# AdaBoost: some properties

- ▶ The **error rate calculated for the training sample**  $e_k$  of the learner  $g_k$  is given by:

$$e_k = \frac{\sum_{i=1}^n w_i \mathbb{1}(g_k(\mathbf{x}_i) \neq y_i)}{\sum_{i=1}^n w_i}.$$

- ▶ Denote  $\epsilon_k$  the **gain** of  $g_k$  **w.r.t. a purely random classifier**:

$$e_k = \frac{1}{2} - \epsilon_k.$$

# AdaBoost: some properties

- ▶ The **error rate calculated for the training sample**  $e_k$  of the learner  $g_k$  is given by:

$$e_k = \frac{\sum_{i=1}^n w_i \mathbb{1}(g_k(\mathbf{x}_i) \neq y_i)}{\sum_{i=1}^n w_i}.$$

- ▶ Denote  $\epsilon_k$  the **gain** of  $g_k$  **w.r.t. a purely random classifier**:

$$e_k = \frac{1}{2} - \epsilon_k.$$

- ▶ The empirical error (on training sample) is bounded by (Freund and Schapire, 1999):

$$L_n(g^{bst}) \leq e^{-2 \sum_{k=1}^B \epsilon_k^2}.$$

# AdaBoost: some properties

- ▶ The **error rate calculated for the training sample**  $e_k$  of the learner  $g_k$  is given by:

$$e_k = \frac{\sum_{i=1}^n w_i \mathbb{1}(g_k(\mathbf{x}_i) \neq y_i)}{\sum_{i=1}^n w_i}.$$

- ▶ Denote  $\epsilon_k$  the **gain** of  $g_k$  **w.r.t. a purely random classifier**:

$$e_k = \frac{1}{2} - \epsilon_k.$$

- ▶ The empirical error (on training sample) is bounded by (Freund and Schapire, 1999):

$$L_n(g^{bst}) \leq e^{-2 \sum_{k=1}^B \epsilon_k^2}.$$

- ▶ Consequently, the empirical error (on the training sample) **decreases to 0** with increasing  $B$ .

# AdaBoost: some properties

- ▶ The **error rate calculated for the training sample**  $e_k$  of the learner  $g_k$  is given by:

$$e_k = \frac{\sum_{i=1}^n w_i \mathbb{1}(g_k(\mathbf{x}_i) \neq y_i)}{\sum_{i=1}^n w_i}.$$

- ▶ Denote  $\epsilon_k$  the **gain** of  $g_k$  **w.r.t. a purely random classifier**:

$$e_k = \frac{1}{2} - \epsilon_k.$$

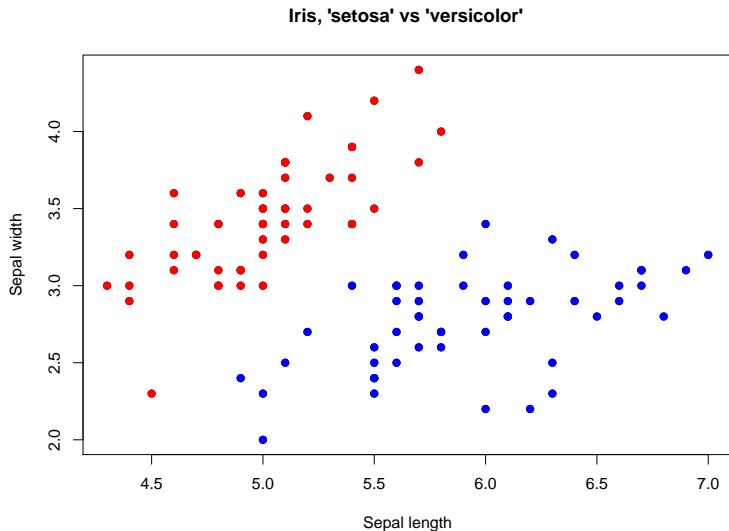
- ▶ The empirical error (on training sample) is bounded by (Freund and Schapire, 1999):

$$L_n(g^{bst}) \leq e^{-2 \sum_{k=1}^B \epsilon_k^2}.$$

- ▶ Consequently, the empirical error (on the training sample) **decreases to 0** with increasing  $B$ .
- ▶ Thus, if  $B$  is (too) large, AdaBoost tends to overfit the sample.

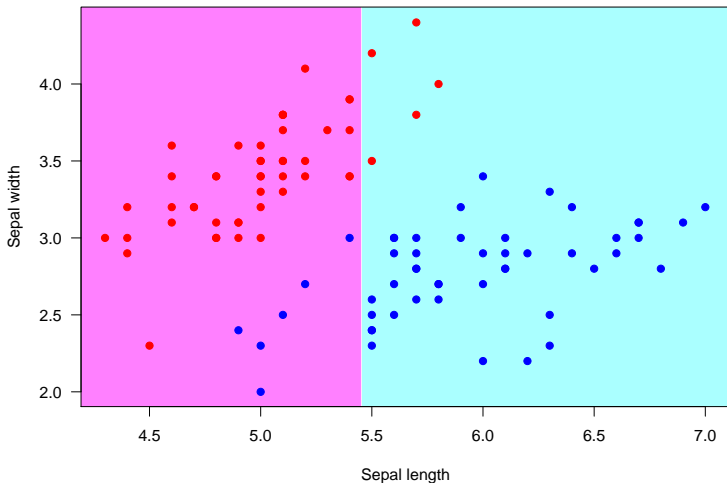


# AdaBoost (Iris data, "setosa" vs. "versicolor")



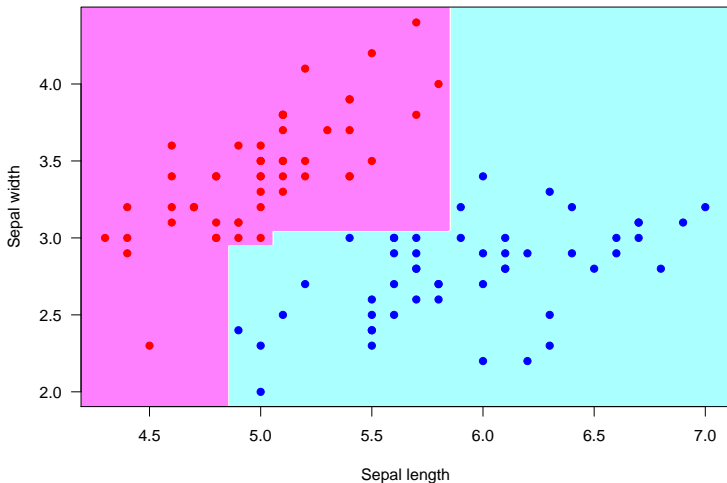
# AdaBoost (Iris data, “setosa” vs. “versicolor”)

AdaBoost for Iris data, maxdepth = 1, B = 1



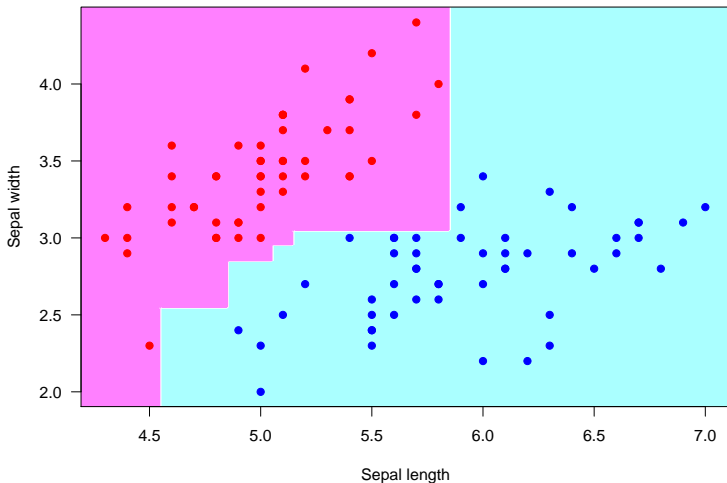
# AdaBoost (Iris data, “setosa” vs. “versicolor”)

AdaBoost for Iris data, maxdepth = 1, B = 10



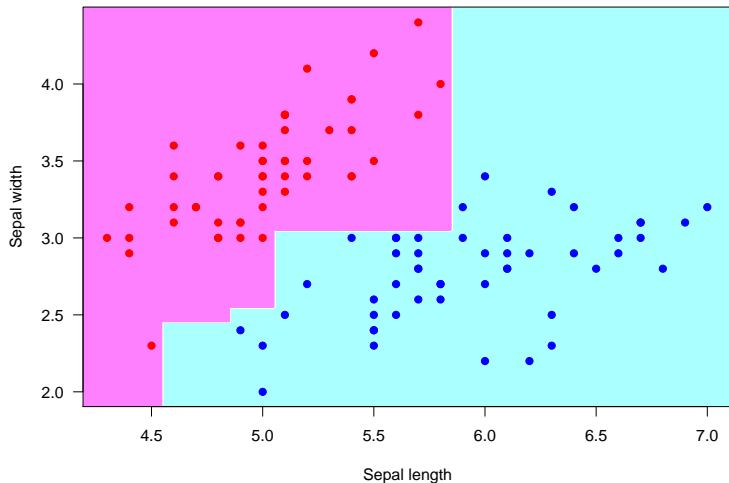
# AdaBoost (Iris data, “setosa” vs. “versicolor”)

AdaBoost for Iris data, maxdepth = 1, B = 50



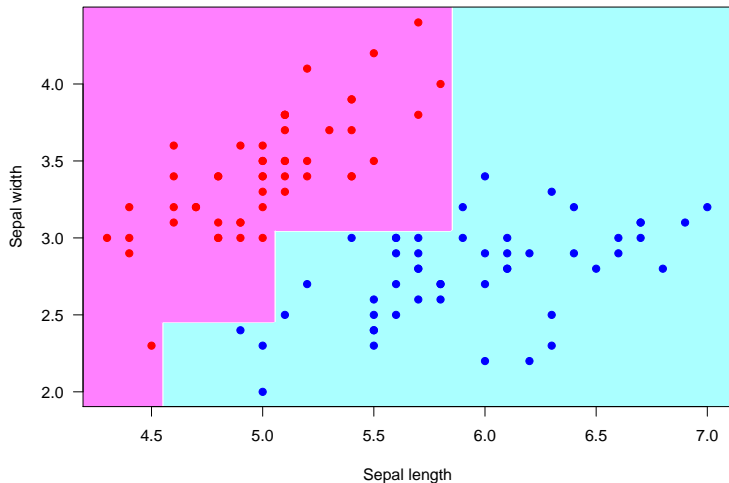
# AdaBoost (Iris data, “setosa” vs. “versicolor”)

AdaBoost for Iris data, maxdepth = 1, B = 100

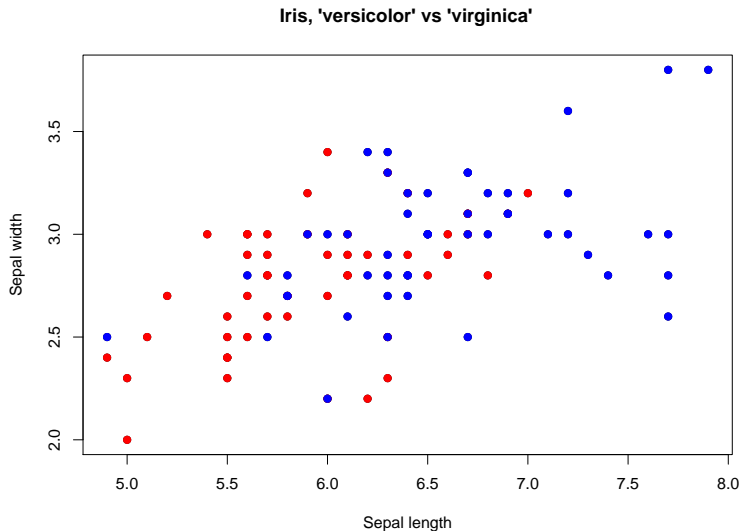


# AdaBoost (Iris data, “setosa” vs. “versicolor”)

AdaBoost for Iris data, maxdepth = 1, B = 500

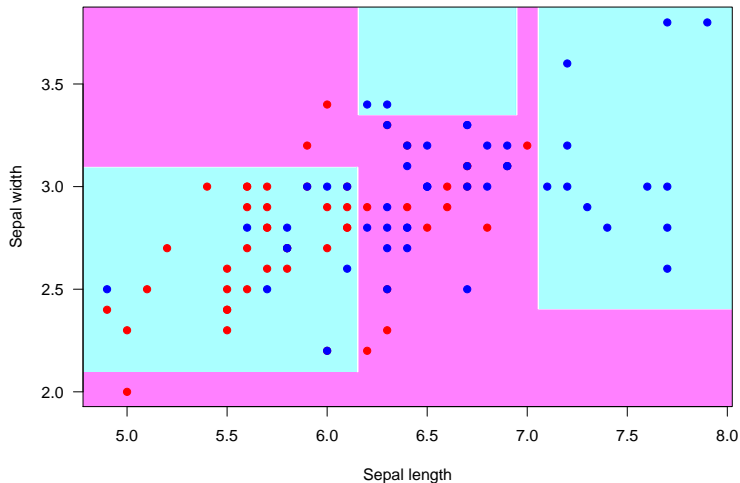


# AdaBoost (Iris data, "versicolor" vs. "virginica")



# AdaBoost (Iris data, “versicolor” vs. “virginica”)

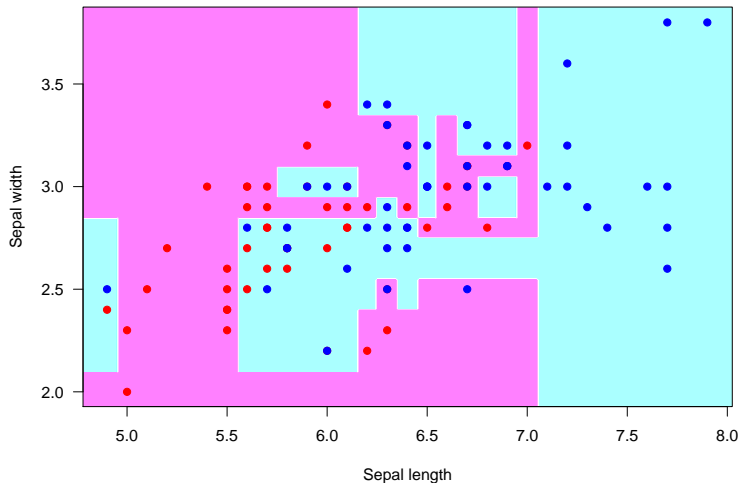
AdaBoost for Iris data, maxdepth = 2, B = 500





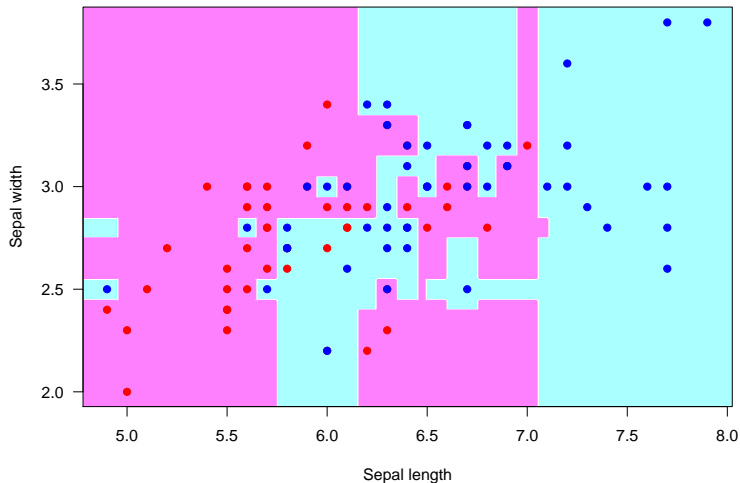
# AdaBoost (Iris data, “versicolor” vs. “virginica”)

AdaBoost for Iris data, maxdepth = 3, B = 500



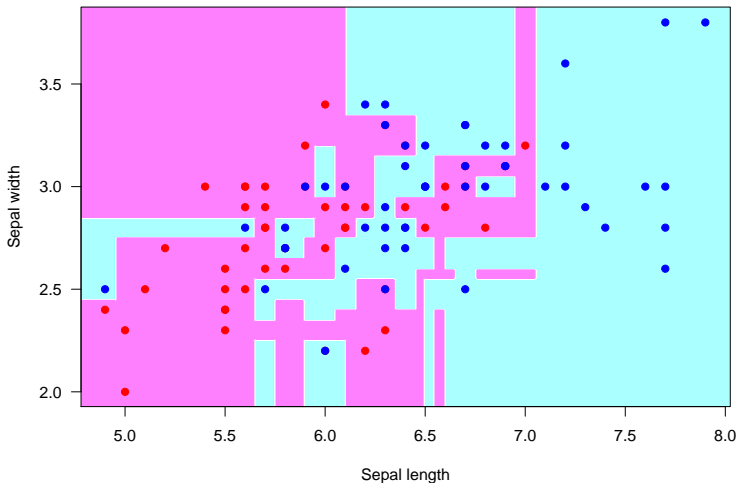
# AdaBoost (Iris data, “versicolor” vs. “virginica”)

AdaBoost for Iris data, maxdepth = 4, B = 500

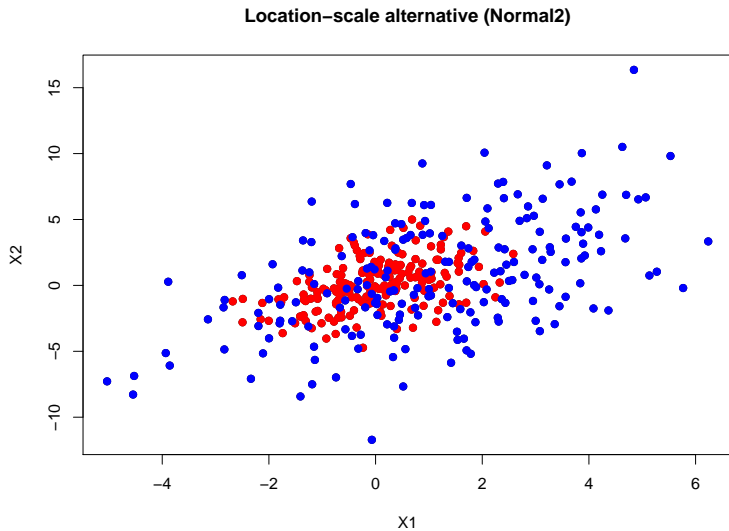


# AdaBoost (Iris data, “versicolor” vs. “virginica”)

AdaBoost for Iris data, maxdepth = 5, B = 500

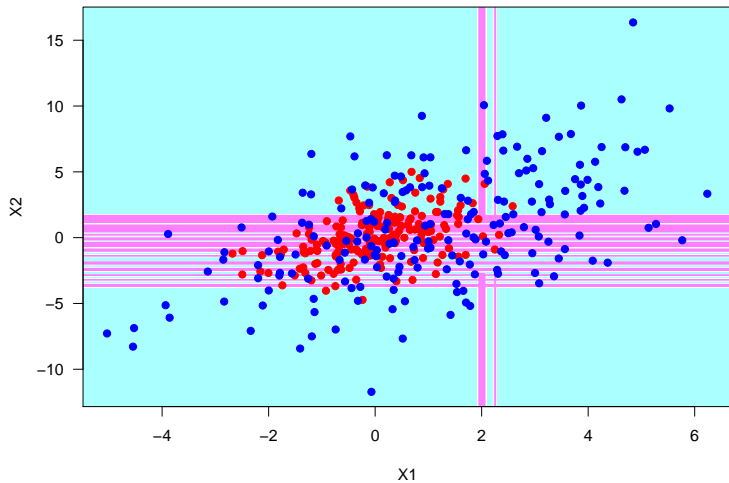


# AdaBoost (normal location-scale alternative)



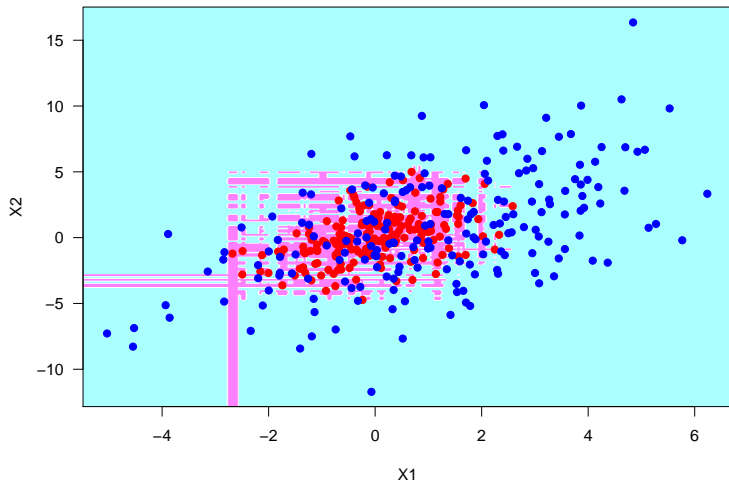
# AdaBoost (normal location-scale alternative)

AdaBoost for Normal2 data, maxdepth = 1, B = 500



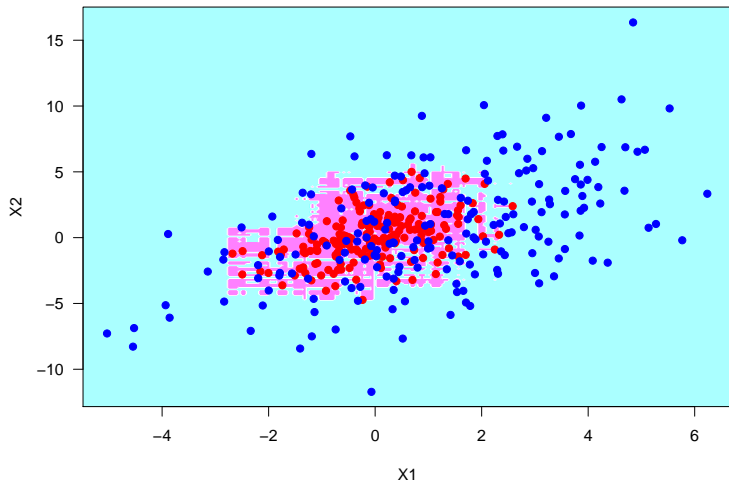
# AdaBoost (normal location-scale alternative)

AdaBoost for Normal2 data, maxdepth = 2, B = 500



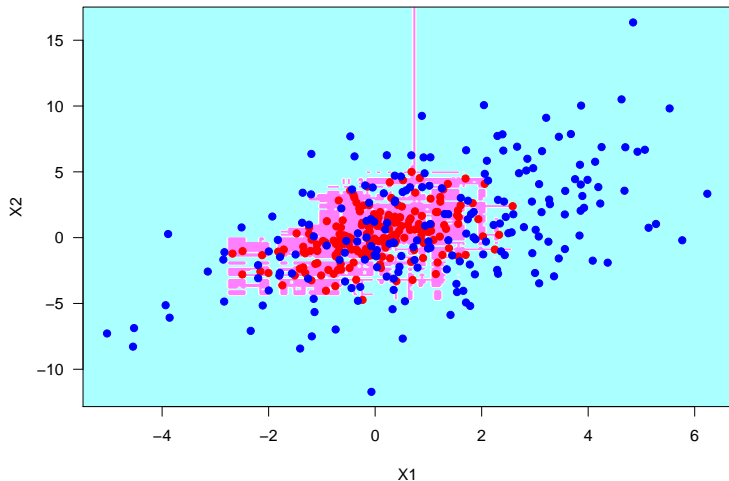
# AdaBoost (normal location-scale alternative)

AdaBoost for Normal2 data, maxdepth = 3, B = 500



# AdaBoost (normal location-scale alternative)

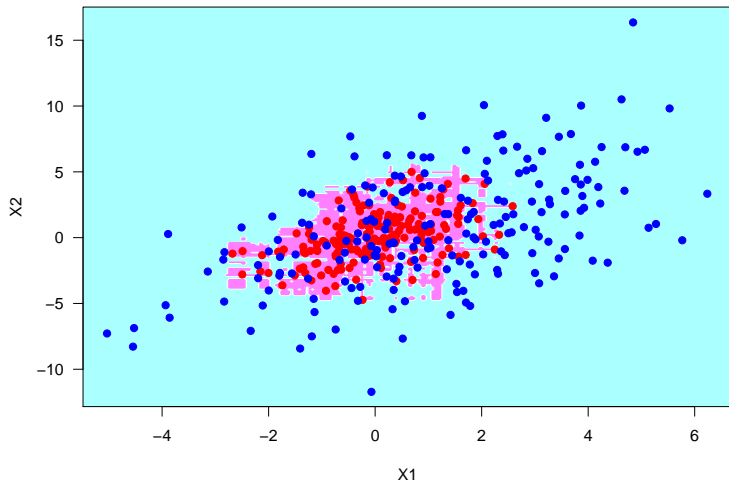
AdaBoost for Normal2 data, maxdepth = 4, B = 500





# AdaBoost (normal location-scale alternative)

AdaBoost for Normal2 data, maxdepth = 5, B = 500



# Contents

## AdaBoost

Idea and the algorithm

Properties and examples

## Justification and generalization

Minimization of the empirical risk

Gradient boosting

Tree-based gradient boosting

LogitBoost

# Contents

## AdaBoost

Idea and the algorithm

Properties and examples

## Justification and generalization

**Minimization of the empirical risk**

Gradient boosting

Tree-based gradient boosting

LogitBoost

# Minimization of the empirical risk

- ▶ Regard a **random pair**  $(X, Y)$  taking values in  $\mathbb{R}^d \times \{-1, 1\}$ .

# Minimization of the empirical risk

- ▶ Regard a **random pair**  $(X, Y)$  taking values in  $\mathbb{R}^d \times \{-1, 1\}$ .
- ▶ Consider a **class of classification rules**  $\mathcal{G}$ :

# Minimization of the empirical risk

- ▶ Regard a **random pair**  $(X, Y)$  taking values in  $\mathbb{R}^d \times \{-1, 1\}$ .
- ▶ Consider a **class of classification rules**  $\mathcal{G}$ :
  - one attempts to find **the best rule** in  $\mathcal{G}$ .

# Minimization of the empirical risk

- ▶ Regard a **random pair**  $(X, Y)$  taking values in  $\mathbb{R}^d \times \{-1, 1\}$ .
- ▶ Consider a **class of classification rules**  $\mathcal{G}$ :
  - one attempts to find **the best rule** in  $\mathcal{G}$ .
- ▶ **Try:** Choose the rule which minimizes a loss function, for example:

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

# Minimization of the empirical risk

- ▶ Regard a **random pair**  $(X, Y)$  taking values in  $\mathbb{R}^d \times \{-1, 1\}$ .
- ▶ Consider a **class of classification rules**  $\mathcal{G}$ :
  - one attempts to find **the best rule** in  $\mathcal{G}$ .
- ▶ **Try:** Choose the rule which minimizes a loss function, for example:

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

- ▶ **Problem:** As we cannot calculate  $\mathbb{P}$ , we cannot calculate  $L(g)$  as well.



# Minimization of the empirical risk

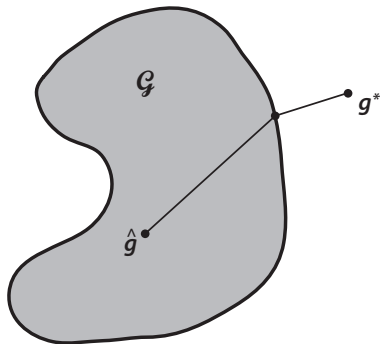
- ▶ Regard a **random pair**  $(X, Y)$  taking values in  $\mathbb{R}^d \times \{-1, 1\}$ .
- ▶ Consider a **class of classification rules**  $\mathcal{G}$ :
  - one attempts to find **the best rule** in  $\mathcal{G}$ .
- ▶ **Try:** Choose the rule which minimizes a loss function, for example:

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

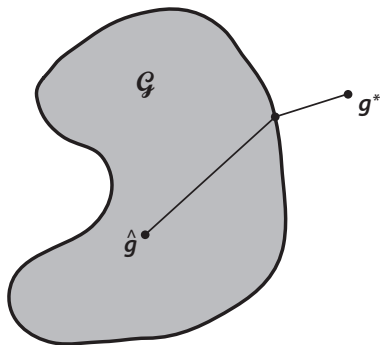
- ▶ **Problem:** As we cannot calculate  $\mathbb{P}$ , we cannot calculate  $L(g)$  as well.
- ▶ **Idea:** Choose a rule that minimizes the empirical version (*i.e.* on the training sample) of the loss function — the **empirical risk**:

$$L_n(g) = \frac{1}{n} \sum_{i=1}^n \mathbb{1}(g(\mathbf{x}_i) \neq y_i).$$

# Minimization of the empirical risk

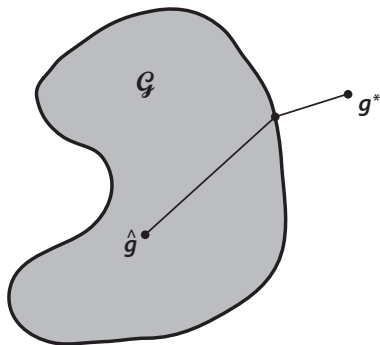


# Minimization of the empirical risk



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

# Minimization of the empirical risk



$$L(\hat{g}) - L(g^*) = \underbrace{L(\hat{g}) - \inf_{g \in \mathcal{G}} L(g)}_{\text{Bias}} + \underbrace{\inf_{g \in \mathcal{G}} L(g) - L(g^*)}_{\text{Variance}} .$$

- ▶ These two terms (usually) vary in inverse sense.

# Risk convexification

- ▶ **Problem:** The function

$$\mathcal{G} \rightarrow \mathbb{R}, \quad g \mapsto \frac{1}{n} \sum_{i=1}^n \mathbb{1}(g(\mathbf{x}_i) \neq y_i)$$

is (usually) **difficult to minimize**.

# Risk convexification

- ▶ **Problem:** The function

$$\mathcal{G} \rightarrow \mathbb{R}, \quad g \mapsto \frac{1}{n} \sum_{i=1}^n \mathbb{1}(g(\mathbf{x}_i) \neq y_i)$$

is (usually) **difficult to minimize**.

- ▶ **Idea:** Find another loss function  $\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$  such that

$$\mathcal{G} \rightarrow \mathbb{R}, \quad g \mapsto \frac{1}{n} \sum_{i=1}^n \ell(g(\mathbf{x}_i), y_i)$$

is “**easy**” to minimize, e.g. a **differentiable** one.

# Risk convexification

- ▶ **Problem:** The function

$$\mathcal{G} \rightarrow \mathbb{R}, \quad g \mapsto \frac{1}{n} \sum_{i=1}^n \mathbb{1}(g(\mathbf{x}_i) \neq y_i)$$

is (usually) **difficult to minimize**.

- ▶ **Idea:** Find another loss function  $\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$  such that

$$\mathcal{G} \rightarrow \mathbb{R}, \quad g \mapsto \frac{1}{n} \sum_{i=1}^n \ell(g(\mathbf{x}_i), y_i)$$

is “**easy**” to minimize, e.g. a **differentiable** one.

- ▶ This is even more the case if the function  $u \mapsto \ell(u, v)$  is **convex**.

# Risk convexification

- ▶ **Problem:** The function

$$\mathcal{G} \rightarrow \mathbb{R}, \quad g \mapsto \frac{1}{n} \sum_{i=1}^n \mathbb{1}(g(\mathbf{x}_i) \neq y_i)$$

is (usually) **difficult to minimize**.

- ▶ **Idea:** Find another loss function  $\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$  such that

$$\mathcal{G} \rightarrow \mathbb{R}, \quad g \mapsto \frac{1}{n} \sum_{i=1}^n \ell(g(\mathbf{x}_i), y_i)$$

is **“easy” to minimize**, e.g. a **differentiable** one.

- ▶ This is even more the case if the function  $u \mapsto \ell(u, v)$  is **convex**.
- ▶ The loss function  $\ell(g(\mathbf{x}), y)$  should **measure the difference** between the value to be predicted  $y \in \{-1, 1\}$  and  $g(\mathbf{x})$ .



# Risk convexification

- ▶ **Problem:** The function

$$\mathcal{G} \rightarrow \mathbb{R}, \quad g \mapsto \frac{1}{n} \sum_{i=1}^n \mathbb{1}(g(\mathbf{x}_i) \neq y_i)$$

is (usually) **difficult to minimize**.

- ▶ **Idea:** Find another loss function  $\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$  such that

$$\mathcal{G} \rightarrow \mathbb{R}, \quad g \mapsto \frac{1}{n} \sum_{i=1}^n \ell(g(\mathbf{x}_i), y_i)$$

is **“easy” to minimize**, e.g. a **differentiable** one.

- ▶ This is even more the case if the function  $u \mapsto \ell(u, v)$  is **convex**.
- ▶ The loss function  $\ell(g(\mathbf{x}), y)$  should **measure the difference** between the value to be predicted  $y \in \{-1, 1\}$  and  $g(\mathbf{x})$ .
- ▶ Thus  $\ell(g(\mathbf{x}), y)$  should take:

# Risk convexification

- ▶ **Problem:** The function

$$\mathcal{G} \rightarrow \mathbb{R}, \quad g \mapsto \frac{1}{n} \sum_{i=1}^n \mathbb{1}(g(\mathbf{x}_i) \neq y_i)$$

is (usually) **difficult to minimize**.

- ▶ **Idea:** Find another loss function  $\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$  such that

$$\mathcal{G} \rightarrow \mathbb{R}, \quad g \mapsto \frac{1}{n} \sum_{i=1}^n \ell(g(\mathbf{x}_i), y_i)$$

is “**easy**” to minimize, e.g. a **differentiable** one.

- ▶ This is even more the case if the function  $u \mapsto \ell(u, v)$  is **convex**.
- ▶ The loss function  $\ell(g(\mathbf{x}), y)$  should **measure the difference** between the value to be predicted  $y \in \{-1, 1\}$  and  $g(\mathbf{x})$ .
- ▶ Thus  $\ell(g(\mathbf{x}), y)$  should take:
  - **large** values if  $yg(\mathbf{x}) < 0$ ,

# Risk convexification

- ▶ **Problem:** The function

$$\mathcal{G} \rightarrow \mathbb{R}, \quad g \mapsto \frac{1}{n} \sum_{i=1}^n \mathbb{1}(g(\mathbf{x}_i) \neq y_i)$$

is (usually) **difficult to minimize**.

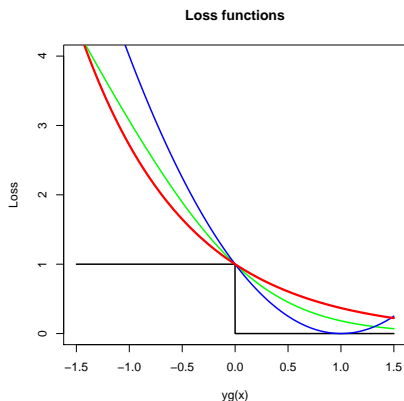
- ▶ **Idea:** Find another loss function  $\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$  such that

$$\mathcal{G} \rightarrow \mathbb{R}, \quad g \mapsto \frac{1}{n} \sum_{i=1}^n \ell(g(\mathbf{x}_i), y_i)$$

is “**easy**” to minimize, e.g. a **differentiable** one.

- ▶ This is even more the case if the function  $u \mapsto \ell(u, v)$  is **convex**.
- ▶ The loss function  $\ell(g(\mathbf{x}), y)$  should **measure the difference** between the value to be predicted  $y \in \{-1, 1\}$  and  $g(\mathbf{x})$ .
- ▶ Thus  $\ell(g(\mathbf{x}), y)$  should take:
  - **large** values if  $yg(\mathbf{x}) < 0$ ,
  - **small** values if  $yg(\mathbf{x}) > 0$ .

# Loss functions



— Misclassification:

$$\ell(g(\mathbf{x}), y) = \mathbb{1}(yg(\mathbf{x}) < 0).$$

— Exponential:

$$\ell(g(\mathbf{x}), y) = e^{-yg(\mathbf{x})}.$$

— Binomial log-likelihood:

$$\ell(g(\mathbf{x}), y) = -\log(1 + e^{-2yg(\mathbf{x})}).$$

— Squared error:

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

# Summary

- ▶ For:
  - a random pair  $(X, Y)$  taking values in  $\mathbb{R}^d \times \{-1, 1\}$ ,
  - a loss function  $\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$

# Summary

- ▶ For:
  - a random pair  $(X, Y)$  taking values in  $\mathbb{R}^d \times \{-1, 1\}$ ,
  - a loss function  $\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$one seeks a classifier close to:

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

# Summary

- ▶ For:
  - a random pair  $(X, Y)$  taking values in  $\mathbb{R}^d \times \{-1, 1\}$ ,
  - a loss function  $\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$one seeks a classifier close to:

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

- ▶ **Strategy:** Given a training sample  $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$  of  $(X, Y)$ , one minimizes the empirical version of  $\mathbb{E}[\ell(g(X), Y)]$ :

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

# Summary

- ▶ For:
  - a random pair  $(X, Y)$  taking values in  $\mathbb{R}^d \times \{-1, 1\}$ ,
  - a loss function  $\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$one seeks a classifier close to:

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

- ▶ **Strategy:** Given a training sample  $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$  of  $(X, Y)$ , one minimizes the empirical version of  $\mathbb{E}[\ell(g(X), Y)]$ :

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

- ▶ **Recursive approach:** Approximate  $g^*$  by  $\hat{g}(\cdot) = \sum_{k=1}^B g_k(\cdot)$ , where  $g_k(\cdot)$ s are constructed recursively.



# Summary

- ▶ For:
  - a random pair  $(X, Y)$  taking values in  $\mathbb{R}^d \times \{-1, 1\}$ ,
  - a loss function  $\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$one seeks a classifier close to:

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

- ▶ **Strategy:** Given a training sample  $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$  of  $(X, Y)$ , one minimizes the empirical version of  $\mathbb{E}[\ell(g(X), Y)]$ :

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

- ▶ **Recursive approach:** Approximate  $g^*$  by  $\hat{g}(\cdot) = \sum_{k=1}^B g_k(\cdot)$ , where  $g_k(\cdot)$ s are constructed recursively.
- ▶ **Method:** Numerical optimization, e.g., **gradient descent**.

# Contents

## AdaBoost

Idea and the algorithm

Properties and examples

## Justification and generalization

Minimization of the empirical risk

**Gradient boosting**

Tree-based gradient boosting

LogitBoost

## Functional gradient descent (FGD)

- ▶ Let  $\mathbf{g}_k = (g_k(\mathbf{x}_1), g_k(\mathbf{x}_2), \dots, g_k(\mathbf{x}_n))$ , and

# Functional gradient descent (FGD)

- ▶ Let  $\mathbf{g}_k = (g_k(\mathbf{x}_1), g_k(\mathbf{x}_2), \dots, g_k(\mathbf{x}_n))$ , and

$$J(\mathbf{g}_k) = \frac{1}{n} \sum_{i=1}^n \ell(g_k(\mathbf{x}_i), y_i).$$

# Functional gradient descent (FGD)

- ▶ Let  $\mathbf{g}_k = (g_k(\mathbf{x}_1), g_k(\mathbf{x}_2), \dots, g_k(\mathbf{x}_n))$ , and

$$J(\mathbf{g}_k) = \frac{1}{n} \sum_{i=1}^n \ell(g_k(\mathbf{x}_i), y_i).$$

- ▶ The gradient descent algorithm can be expressed by the recursive formula:

$$\mathbf{g}_k = \mathbf{g}_{k-1} - \lambda \nabla J(\mathbf{g}_{k-1})$$

with  $\lambda > 0$  being the step size of the gradient descent.

# Functional gradient descent (FGD)

- ▶ Let  $\mathbf{g}_k = (g_k(\mathbf{x}_1), g_k(\mathbf{x}_2), \dots, g_k(\mathbf{x}_n))$ , and

$$J(\mathbf{g}_k) = \frac{1}{n} \sum_{i=1}^n \ell(g_k(\mathbf{x}_i), y_i).$$

- ▶ The gradient descent algorithm can be expressed by the recursive formula:

$$\mathbf{g}_k = \mathbf{g}_{k-1} - \lambda \nabla J(\mathbf{g}_{k-1})$$

with  $\lambda > 0$  being the step size of the gradient descent.

Such an approach possesses certain disadvantages:

# Functional gradient descent (FGD)

- ▶ Let  $\mathbf{g}_k = (g_k(\mathbf{x}_1), g_k(\mathbf{x}_2), \dots, g_k(\mathbf{x}_n))$ , and

$$J(\mathbf{g}_k) = \frac{1}{n} \sum_{i=1}^n \ell(g_k(\mathbf{x}_i), y_i).$$

- ▶ The gradient descent algorithm can be expressed by the recursive formula:

$$\mathbf{g}_k = \mathbf{g}_{k-1} - \lambda \nabla J(\mathbf{g}_{k-1})$$

with  $\lambda > 0$  being the step size of the gradient descent.

Such an approach possesses certain disadvantages:

- ▶ The regularity of the estimated function is not taken into account:

# Functional gradient descent (FGD)

- ▶ Let  $\mathbf{g}_k = (g_k(\mathbf{x}_1), g_k(\mathbf{x}_2), \dots, g_k(\mathbf{x}_n))$ , and

$$J(\mathbf{g}_k) = \frac{1}{n} \sum_{i=1}^n \ell(g_k(\mathbf{x}_i), y_i).$$

- ▶ The gradient descent algorithm can be expressed by the recursive formula:

$$\mathbf{g}_k = \mathbf{g}_{k-1} - \lambda \nabla J(\mathbf{g}_{k-1})$$

with  $\lambda > 0$  being the step size of the gradient descent.

Such an approach possesses certain disadvantages:

- ▶ The regularity of the estimated function is not taken into account:
  - if  $\mathbf{x}_i$  is close to  $\mathbf{x}_j$  then  $\hat{g}(\mathbf{x}_i)$  is close to  $\hat{g}(\mathbf{x}_j)$ .



# Functional gradient descent (FGD)

- ▶ Let  $\mathbf{g}_k = (g_k(\mathbf{x}_1), g_k(\mathbf{x}_2), \dots, g_k(\mathbf{x}_n))$ , and

$$J(\mathbf{g}_k) = \frac{1}{n} \sum_{i=1}^n \ell(g_k(\mathbf{x}_i), y_i).$$

- ▶ The gradient descent algorithm can be expressed by the recursive formula:

$$\mathbf{g}_k = \mathbf{g}_{k-1} - \lambda \nabla J(\mathbf{g}_{k-1})$$

with  $\lambda > 0$  being the step size of the gradient descent.

Such an approach possesses certain disadvantages:

- ▶ The regularity of the estimated function is not taken into account:
  - if  $\mathbf{x}_i$  is close to  $\mathbf{x}_j$  then  $\hat{g}(\mathbf{x}_i)$  is close to  $\hat{g}(\mathbf{x}_j)$ .
- ▶ The estimator is calculated at the points of the training sample only  $\mathbf{x}_1, \dots, \mathbf{x}_n$ .

# Gradient boost (algorithm): Friedman (2001)

## Constructing a classifier (training)

**Input:**

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .

# Gradient boost (algorithm): Friedman (2001)

## Constructing a classifier (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
- ▶ Weak learner  $g(\cdot)$ .

# Gradient boost (algorithm): Friedman (2001)

## Constructing a classifier (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
- ▶ Weak learner  $g(\cdot)$ .
- ▶ Number of iterations  $B$ .

# Gradient boost (algorithm): Friedman (2001)

## Constructing a classifier (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
- ▶ Weak learner  $g(\cdot)$ .
- ▶ Number of iterations  $B$ .
- ▶ Regularization parameter  $\lambda \in (0, 1]$ .

# Gradient boost (algorithm): Friedman (2001)

## Constructing a classifier (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
- ▶ Weak learner  $g(\cdot)$ .
- ▶ Number of iterations  $B$ .
- ▶ Regularization parameter  $\lambda \in (0, 1]$ .

1. Initialization:  $g_0(\cdot) = \arg \min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n \ell(c, y_i)$ .

# Gradient boost (algorithm): Friedman (2001)

## Constructing a classifier (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
- ▶ Weak learner  $g(\cdot)$ .
- ▶ Number of iterations  $B$ .
- ▶ Regularization parameter  $\lambda \in (0, 1]$ .

1. Initialization:  $g_0(\cdot) = \arg \min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n \ell(c, y_i)$ .
2. For  $k = 1, \dots, B$

# Gradient boost (algorithm): Friedman (2001)

## Constructing a classifier (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
- ▶ Weak learner  $g(\cdot)$ .
- ▶ Number of iterations  $B$ .
- ▶ Regularization parameter  $\lambda \in (0, 1]$ .

1. Initialization:  $g_0(\cdot) = \arg \min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n \ell(c, y_i)$ .

2. For  $k = 1, \dots, B$

2.1 Calculate the negative gradient  $-\frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i)$  and evaluate it at points  $g_{k-1}(\mathbf{x}_i)$ :

$$v_i = -\frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i) \Bigg|_{g(\mathbf{x}_i)=g_{k-1}(\mathbf{x}_i)}, \quad i = 1, \dots, n.$$



# Gradient boost (algorithm): Friedman (2001)

## Constructing a classifier (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
- ▶ Weak learner  $g(\cdot)$ .
- ▶ Number of iterations  $B$ .
- ▶ Regularization parameter  $\lambda \in (0, 1]$ .

1. Initialization:  $g_0(\cdot) = \arg \min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n \ell(c, y_i)$ .

2. For  $k = 1, \dots, B$

2.1 Calculate the negative gradient  $-\frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i)$  and evaluate it at points  $g_{k-1}(\mathbf{x}_i)$ :

$$v_i = -\frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i) \Bigg|_{g(\mathbf{x}_i)=g_{k-1}(\mathbf{x}_i)}, \quad i = 1, \dots, n.$$

2.2 Learn a weak learner  $h_k(\cdot)$  on  $(\mathbf{x}_1, v_1), (\mathbf{x}_2, v_2), \dots, (\mathbf{x}_n, v_n)$ .

# Gradient boost (algorithm): Friedman (2001)

## Constructing a classifier (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
- ▶ Weak learner  $g(\cdot)$ .
- ▶ Number of iterations  $B$ .
- ▶ Regularization parameter  $\lambda \in (0, 1]$ .

1. Initialization:  $g_0(\cdot) = \arg \min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n \ell(c, y_i)$ .

2. For  $k = 1, \dots, B$

2.1 Calculate the negative gradient  $-\frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i)$  and evaluate it at points  $g_{k-1}(\mathbf{x}_i)$ :

$$v_i = -\frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i) \Bigg|_{g(\mathbf{x}_i)=g_{k-1}(\mathbf{x}_i)}, \quad i = 1, \dots, n.$$

2.2 Learn a weak learner  $h_k(\cdot)$  on  $(\mathbf{x}_1, v_1), (\mathbf{x}_2, v_2), \dots, (\mathbf{x}_n, v_n)$ .

2.3 Update the classifier:  $g_k(\cdot) = g_{k-1}(\cdot) + \lambda h_k(\cdot)$ .

# Gradient boost (algorithm): Friedman (2001)

## Constructing a classifier (training)

### Input:

- ▶ Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .
- ▶ Weak learner  $g(\cdot)$ .
- ▶ Number of iterations  $B$ .
- ▶ Regularization parameter  $\lambda \in (0, 1]$ .

1. Initialization:  $g_0(\cdot) = \arg \min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n \ell(c, y_i)$ .

2. For  $k = 1, \dots, B$

2.1 Calculate the negative gradient  $-\frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i)$  and evaluate it at points  $g_{k-1}(\mathbf{x}_i)$ :

$$v_i = -\frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i) \Bigg|_{g(\mathbf{x}_i)=g_{k-1}(\mathbf{x}_i)}, \quad i = 1, \dots, n.$$

2.2 Learn a weak learner  $h_k(\cdot)$  on  $(\mathbf{x}_1, v_1), (\mathbf{x}_2, v_2), \dots, (\mathbf{x}_n, v_n)$ .

2.3 Update the classifier:  $g_k(\cdot) = g_{k-1}(\cdot) + \lambda h_k(\cdot)$ .

**Output:** The classification rule  $\hat{g}(\cdot) = g_k(\cdot)$ .

## Gradient boost: some comments

- ▶ The output of  $\hat{g}(\cdot)$  is a **real number**. To predict the class label one should use the “sign” operator:

$$\hat{y} = \text{sign}(\hat{g}(\mathbf{x})) .$$

## Gradient boost: some comments

- ▶ The output of  $\hat{g}(\cdot)$  is a **real number**. To predict the class label one should use the “sign” operator:

$$\hat{y} = \text{sign}(\hat{g}(\mathbf{x})) .$$

- ▶ The algorithm (almost) **coincides with AdaBoost** for:

## Gradient boost: some comments

- ▶ The output of  $\hat{g}(\cdot)$  is a **real number**. To predict the class label one should use the “sign” operator:

$$\hat{y} = \text{sign}(\hat{g}(\mathbf{x})) .$$

- ▶ The algorithm (almost) **coincides with AdaBoost** for:
  - $\ell(g(\mathbf{x}), y) = e^{-yg(\mathbf{x})}$ ,

## Gradient boost: some comments

- ▶ The output of  $\hat{g}(\cdot)$  is a **real number**. To predict the class label one should use the “sign” operator:

$$\hat{y} = \text{sign}(\hat{g}(\mathbf{x})) .$$

- ▶ The algorithm (almost) **coincides with AdaBoost** for:
  - $\ell(g(\mathbf{x}), y) = e^{-yg(\mathbf{x})}$ ,
  - $\lambda = 1$ .

## Gradient boost: some comments

- ▶ The output of  $\hat{g}(\cdot)$  is a **real number**. To predict the class label one should use the “sign” operator:

$$\hat{y} = \text{sign}(\hat{g}(\mathbf{x})) .$$

- ▶ The algorithm (almost) **coincides with AdaBoost** for:
  - $\ell(g(\mathbf{x}), y) = e^{-yg(\mathbf{x})}$ ,
  - $\lambda = 1$ .
- ▶ The **choice of the regularization parameter**  $\lambda$  is connected to the number of iterations  $B$ .  
It “controls” the speed of minimization of function

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



## Gradient boost: some comments

- ▶ The output of  $\hat{g}(\cdot)$  is a **real number**. To predict the class label one should use the “sign” operator:

$$\hat{y} = \text{sign}(\hat{g}(\mathbf{x})) .$$

- ▶ The algorithm (almost) **coincides with AdaBoost** for:
  - $\ell(g(\mathbf{x}), y) = e^{-yg(\mathbf{x})}$ ,
  - $\lambda = 1$ .
- ▶ The **choice of the regularization parameter**  $\lambda$  is connected to the number of iterations  $B$ .  
It “controls” the speed of minimization of function

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

- ▶ Larger values of  $\lambda$  correspond to smaller values of  $B$ , and *vice versa*.

# Contents

## AdaBoost

Idea and the algorithm

Properties and examples

## Justification and generalization

Minimization of the empirical risk

Gradient boosting

Tree-based gradient boosting

LogitBoost

# Weak learners

- ▶ For Gradient boost, as well as for AdaBoost, the classification rule used in the algorithm should be **weak** (slightly better than the purely random choice).

## Weak learners

- ▶ For Gradient boost, as well as for AdaBoost, the classification rule used in the algorithm should be **weak** (slightly better than the purely random choice).
- ▶ Boosting a **non-weak** classification **rule** usually delivers **poor performance** increase.

## Weak learners

- ▶ For Gradient boost, as well as for AdaBoost, the classification rule used in the algorithm should be **weak** (slightly better than the purely random choice).
- ▶ Boosting a **non-weak** classification **rule** usually delivers **poor performance** increase.
- ▶ It is recommended to use a classification rule with **high bias** but **small variance** (boosting allows to reduce bias and not variance).

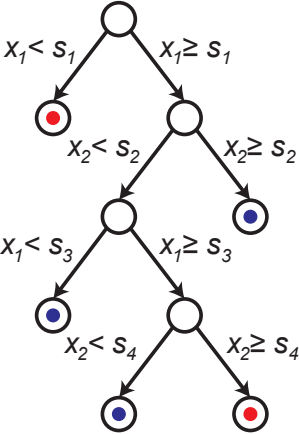
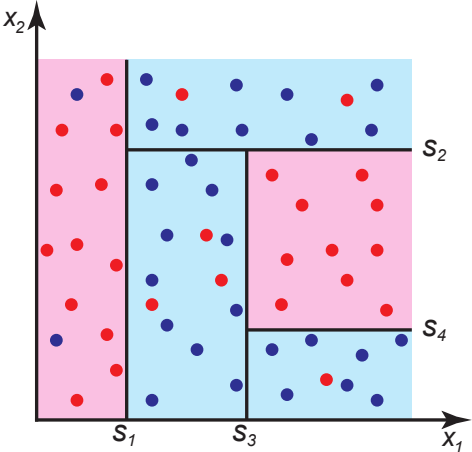
# Weak learners

- ▶ For Gradient boost, as well as for AdaBoost, the classification rule used in the algorithm should be **weak** (slightly better than the purely random choice).
- ▶ Boosting a **non-weak** classification **rule** usually delivers **poor performance** increase.
- ▶ It is recommended to use a classification rule with **high bias** but **small variance** (boosting allows to reduce bias and not variance).
- ▶ Often **tree** is used as a weak learner. For high bias one usually chooses **trees with a few leaves only** (*i.e.*, of low depth).

# Weak learners

- ▶ For Gradient boost, as well as for AdaBoost, the classification rule used in the algorithm should be **weak** (slightly better than the purely random choice).
- ▶ Boosting a **non-weak** classification **rule** usually delivers **poor performance** increase.
- ▶ It is recommended to use a classification rule with **high bias** but **small variance** (boosting allows to reduce bias and not variance).
- ▶ Often **tree** is used as a weak learner. For high bias one usually chooses **trees with a few leaves only** (*i.e.*, of low depth).
- ▶ In numerous implementations of boosting **regression trees** are used.

# Reminder: classification tree





# Regression tree

- ▶ Regression tree is grown in a **(very) similar to the classification tree manner** with slight changes:

# Regression tree

- ▶ Regression tree is grown in a **(very) similar to the classification tree manner** with slight changes:
- ▶ Suppose that a **partition** into  $M$  regions  $R_1, R_2, \dots, R_M$  is given.

# Regression tree

- ▶ Regression tree is grown in a **(very) similar to the classification tree manner** with slight changes:
- ▶ Suppose that a **partition** into  $M$  regions  $R_1, R_2, \dots, R_M$  is given.
- ▶ The **response is** then modeled as a **constant  $c_m$  in each region**:

$$f(\mathbf{x}) = \sum_{m=1}^M c_m \mathbb{1}(\mathbf{x} \in R_m).$$

# Regression tree

- ▶ Regression tree is grown in a **(very) similar to the classification tree manner** with slight changes:
- ▶ Suppose that a **partition** into  $M$  regions  $R_1, R_2, \dots, R_M$  is given.
- ▶ The **response** is then modeled as a **constant**  $c_m$  in each region:

$$f(\mathbf{x}) = \sum_{m=1}^M c_m \mathbb{1}(\mathbf{x} \in R_m).$$

- ▶ Adopt **sum of squares as impurity measure**:

$$Q^{(m)}(T) = \frac{n^{(m_L)}}{n^{(m)}} \sum_{\mathbf{x}_i \in R_{m_L}} (y_i - \hat{c}_{m_L})^2 + \frac{n^{(m_R)}}{n^{(m)}} \sum_{\mathbf{x}_i \in R_{m_R}} (y_i - \hat{c}_{m_R})^2.$$

# Regression tree

- ▶ Regression tree is grown in a **(very) similar to the classification tree manner** with slight changes:
- ▶ Suppose that a **partition** into  $M$  regions  $R_1, R_2, \dots, R_M$  is given.
- ▶ The **response** is then modeled as a **constant**  $c_m$  in each region:

$$f(\mathbf{x}) = \sum_{m=1}^M c_m \mathbb{1}(\mathbf{x} \in R_m).$$

- ▶ Adopt **sum of squares as impurity measure**:

$$Q^{(m)}(T) = \frac{n^{(m_L)}}{n^{(m)}} \sum_{\mathbf{x}_i \in R_{m_L}} (y_i - \hat{c}_{m_L})^2 + \frac{n^{(m_R)}}{n^{(m)}} \sum_{\mathbf{x}_i \in R_{m_R}} (y_i - \hat{c}_{m_R})^2.$$

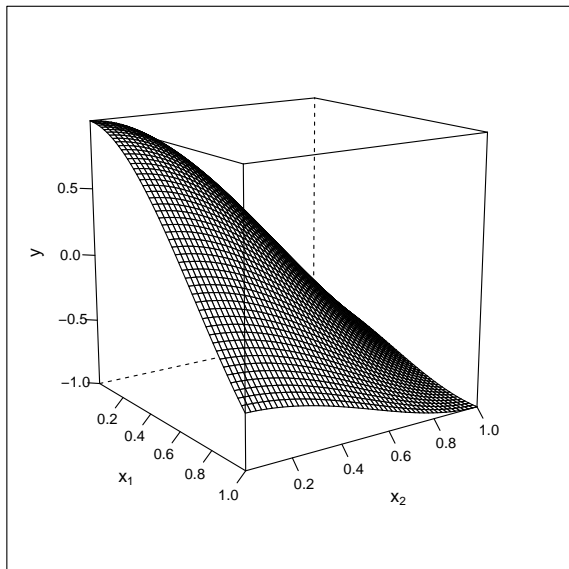
- ▶ One can check that the **optimal choice of  $\hat{c}_m$**  for region  $R_m$  is the **average over  $R_m$** :

$$\hat{c}_m = \frac{1}{n^{(m)}} \sum_{\mathbf{x}_i \in R_m} y_i,$$

with  $n^{(m)}$  being the number of observations  $\mathbf{x}_i \in \mathcal{D}_n$  in region  $R_m$ .

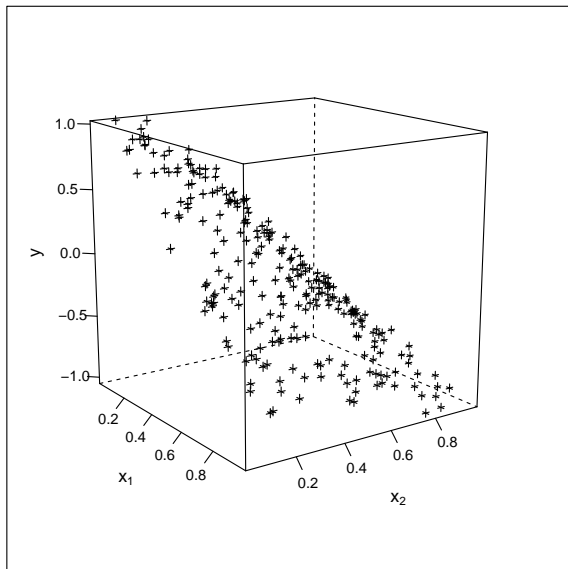
# Regression tree (illustration)

Function surface



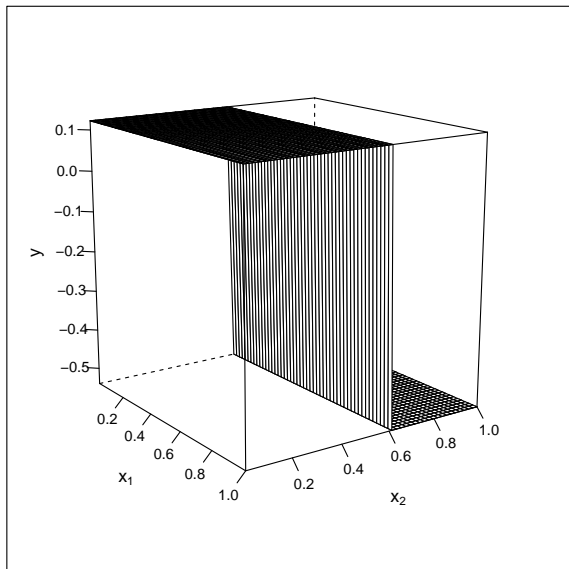
# Regression tree (illustration)

Sample points



# Regression tree (illustration)

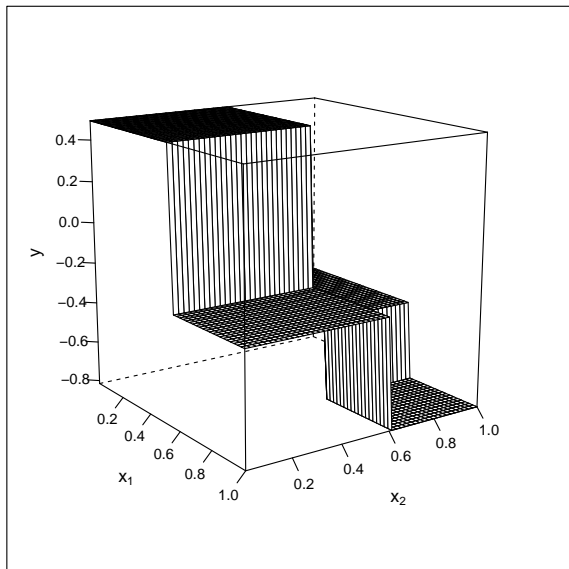
Regression tree surface, maxdepth = 1





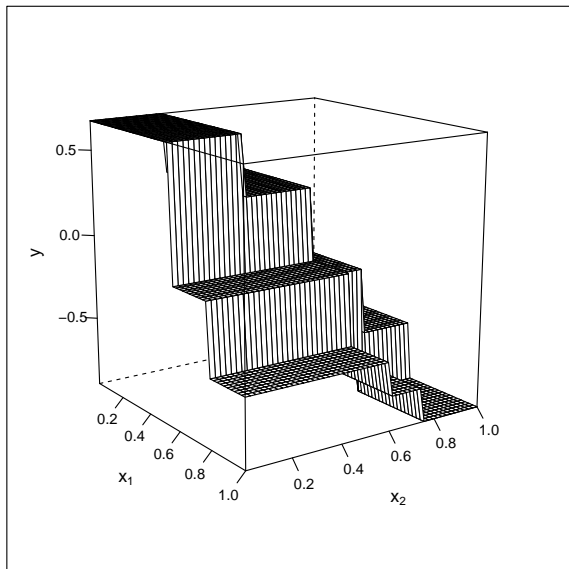
# Regression tree (illustration)

Regression tree surface, maxdepth = 2



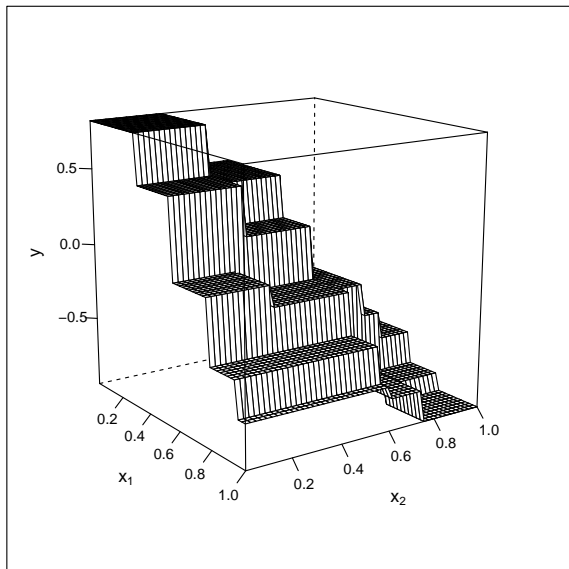
# Regression tree (illustration)

Regression tree surface, maxdepth = 3



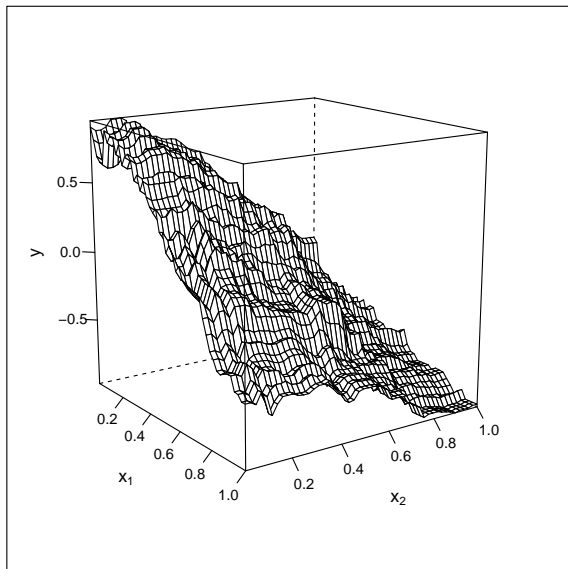
# Regression tree (illustration)

Regression tree surface, maxdepth = 4



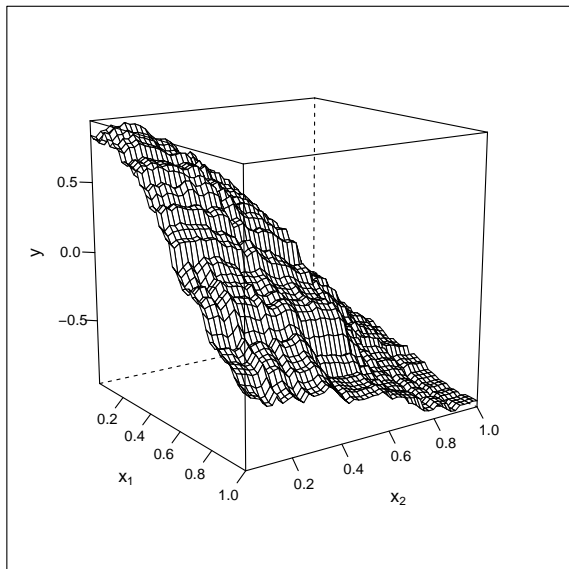
# Regression tree (illustration)

Random forest surface,  $B = 10$



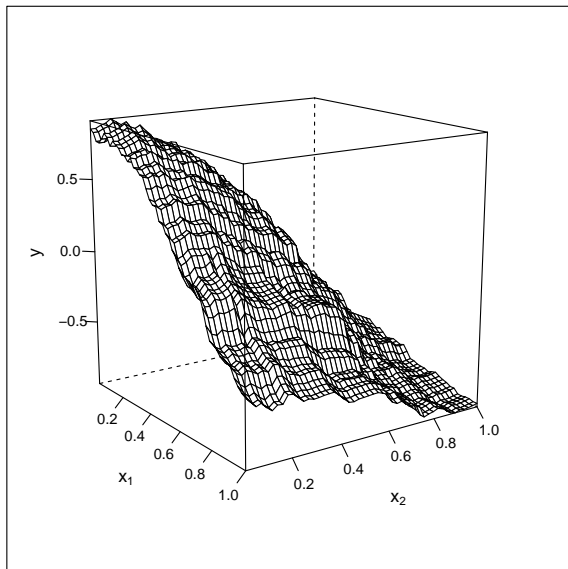
# Regression tree (illustration)

Random forest surface,  $B = 50$



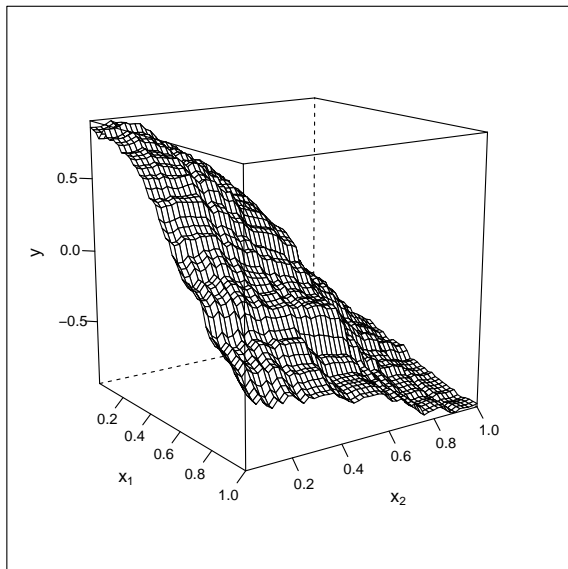
# Regression tree (illustration)

Random forest surface,  $B = 100$



# Regression tree (illustration)

Random forest surface,  $B = 500$



# Boosting regression trees in R

- ▶ Function `gbm` of R-package `gbm` by Ridgeway (2006).



# Boosting regression trees in R

- ▶ Function `gbm` of R-package `gbm` by Ridgeway (2006).

Input parameters (arguments):

- ▶ loss function  $\ell$  (`distribution`),

# Boosting regression trees in R

- ▶ Function `gbm` of R-package `gbm` by Ridgeway (2006).

Input parameters (arguments):

- ▶ loss function  $\ell$  (`distribution`),
- ▶ number of iterations  $B$  (`n.trees`),

# Boosting regression trees in R

- ▶ Function `gbm` of R-package `gbm` by Ridgeway (2006).

Input parameters (arguments):

- ▶ loss function  $\ell$  (`distribution`),
- ▶ number of iterations  $B$  (`n.trees`),
- ▶ depth of the grown tree  $K$  (`interaction.depth`),

# Boosting regression trees in R

- ▶ Function `gbm` of R-package `gbm` by Ridgeway (2006).

Input parameters (arguments):

- ▶ loss function  $\ell$  (`distribution`),
- ▶ number of iterations  $B$  (`n.trees`),
- ▶ depth of the grown tree  $K$  (`interaction.depth`),
- ▶ regularization parameter  $\lambda$  (`shrinkage`).

FGD-trees (algorithm): Friedman (2001), Ridgeway (2006)

**Constructing a classifier (training)**

**Input:** Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .

# FGD-trees (algorithm): Friedman (2001), Ridgeway (2006)

## Constructing a classifier (training)

**Input:** Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .

1. Initialization:  $g_0(\cdot) = \arg \min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n \ell(c, y_i)$ .

# FGD-trees (algorithm): Friedman (2001), Ridgeway (2006)

## Constructing a classifier (training)

**Input:** Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .

1. Initialization:  $g_0(\cdot) = \arg \min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n \ell(c, y_i)$ .
2. For  $k = 1, \dots, B$

# FGD-trees (algorithm): Friedman (2001), Ridgeway (2006)

## Constructing a classifier (training)

**Input:** Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .

1. Initialization:  $g_0(\cdot) = \arg \min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n \ell(c, y_i)$ .
2. For  $k = 1, \dots, B$ 
  - 2.1 Calculate the negative gradient  $-\frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i)$  and evaluate it at points  $g_{k-1}(\mathbf{x}_i)$ :

$$v_i = -\frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i) \Big|_{g(\mathbf{x}_i)=g_{k-1}(\mathbf{x}_i)}, \quad i = 1, \dots, n.$$



# FGD-trees (algorithm): Friedman (2001), Ridgeway (2006)

## Constructing a classifier (training)

**Input:** Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .

1. Initialization:  $g_0(\cdot) = \arg \min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n \ell(c, y_i)$ .
2. For  $k = 1, \dots, B$ 
  - 2.1 Calculate the negative gradient  $-\frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i)$  and evaluate it at points  $g_{k-1}(\mathbf{x}_i)$ :

$$v_i = -\left. \frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i) \right|_{g(\mathbf{x}_i)=g_{k-1}(\mathbf{x}_i)}, \quad i = 1, \dots, n.$$

- 2.2 Grow the regression tree of maximal depth  $K$  based on the sample  $(\mathbf{x}_1, v_1), (\mathbf{x}_2, v_2), \dots, (\mathbf{x}_n, v_n)$ .

# FGD-trees (algorithm): Friedman (2001), Ridgeway (2006)

## Constructing a classifier (training)

**Input:** Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .

1. Initialization:  $g_0(\cdot) = \arg \min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n \ell(c, y_i)$ .
2. For  $k = 1, \dots, B$ 
  - 2.1 Calculate the negative gradient  $-\frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i)$  and evaluate it at points  $g_{k-1}(\mathbf{x}_i)$ :

$$v_i = - \left. \frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i) \right|_{g(\mathbf{x}_i) = g_{k-1}(\mathbf{x}_i)}, \quad i = 1, \dots, n.$$

- 2.2 Grow the regression tree of maximal depth  $K$  based on the sample  $(\mathbf{x}_1, v_1), (\mathbf{x}_2, v_2), \dots, (\mathbf{x}_n, v_n)$ .

- 2.3 For each leaf  $m = 1, \dots, M$  calculate the optimal prediction:

$$\rho_m = \arg \min_{\rho \in \mathbb{R}} \sum_{\mathbf{x}_i \in R_m} \ell(g_{k-1}(\mathbf{x}_i) + \rho, y_i).$$

# FGD-trees (algorithm): Friedman (2001), Ridgeway (2006)

## Constructing a classifier (training)

**Input:** Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .

1. Initialization:  $g_0(\cdot) = \arg \min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n \ell(c, y_i)$ .
2. For  $k = 1, \dots, B$ 
  - 2.1 Calculate the negative gradient  $-\frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i)$  and evaluate it at points  $g_{k-1}(\mathbf{x}_i)$ :

$$v_i = - \left. \frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i) \right|_{g(\mathbf{x}_i) = g_{k-1}(\mathbf{x}_i)}, \quad i = 1, \dots, n.$$

- 2.2 Grow the regression tree of maximal depth  $K$  based on the sample  $(\mathbf{x}_1, v_1), (\mathbf{x}_2, v_2), \dots, (\mathbf{x}_n, v_n)$ .

- 2.3 For each leaf  $m = 1, \dots, M$  calculate the optimal prediction:

$$\rho_m = \arg \min_{\rho \in \mathbb{R}} \sum_{\mathbf{x}_i \in R_m} \ell(g_{k-1}(\mathbf{x}_i) + \rho, y_i).$$

- 2.4 Update the classifier:  $g_k(\mathbf{x}) = g_{k-1}(\mathbf{x}) + \lambda \rho_{m(\mathbf{x})}$  with  $m(\mathbf{x})$  being a leaf containing  $\mathbf{x}$ .

# FGD-trees (algorithm): Friedman (2001), Ridgeway (2006)

## Constructing a classifier (training)

**Input:** Training sample  $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) \subset \mathbb{R}^d \times \{-1, 1\}$ .

1. Initialization:  $g_0(\cdot) = \arg \min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n \ell(c, y_i)$ .
2. For  $k = 1, \dots, B$ 
  - 2.1 Calculate the negative gradient  $-\frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i)$  and evaluate it at points  $g_{k-1}(\mathbf{x}_i)$ :

$$v_i = -\left. \frac{\partial}{\partial g(\mathbf{x}_i)} \ell(g(\mathbf{x}_i), y_i) \right|_{g(\mathbf{x}_i) = g_{k-1}(\mathbf{x}_i)}, \quad i = 1, \dots, n.$$

- 2.2 Grow the regression tree of maximal depth  $K$  based on the sample  $(\mathbf{x}_1, v_1), (\mathbf{x}_2, v_2), \dots, (\mathbf{x}_n, v_n)$ .

- 2.3 For each leaf  $m = 1, \dots, M$  calculate the optimal prediction:

$$\rho_m = \arg \min_{\rho \in \mathbb{R}} \sum_{\mathbf{x}_i \in R_m} \ell(g_{k-1}(\mathbf{x}_i) + \rho, y_i).$$

- 2.4 Update the classifier:  $g_k(\mathbf{x}) = g_{k-1}(\mathbf{x}) + \lambda \rho_{m(\mathbf{x})}$  with  $m(\mathbf{x})$  being a leaf containing  $\mathbf{x}$ .

**Output:** The classification rule  $\hat{g}(\mathbf{x}) = g_k(\mathbf{x})$ . □

# Contents

## AdaBoost

Idea and the algorithm

Properties and examples

## Justification and generalization

Minimization of the empirical risk

Gradient boosting

Tree-based gradient boosting

LogitBoost

## Connection to AdaBoost

- ▶ One can show (Hastie, Tibshirani, Friedman, 2000) that the AdaBoost algorithm is a gradient descent method for minimizing

$$\mathbb{E}[\ell(g(X), Y)].$$

with  $\ell(g(\mathbf{x}), y) = e^{-yg(\mathbf{x})}$ .

## Connection to AdaBoost

- ▶ One can show (Hastie, Tibshirani, Friedman, 2000) that the AdaBoost algorithm is a gradient descent method for minimizing

$$\mathbb{E}[\ell(g(X), Y)].$$

with  $\ell(g(\mathbf{x}), y) = e^{-yg(\mathbf{x})}$ .

- ▶ Thus the output  $\hat{g}(\mathbf{x})$  is an estimator of

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

# Connection to AdaBoost

- ▶ One can show (Hastie, Tibshirani, Friedman, 2000) that the AdaBoost algorithm is a gradient descent method for minimizing

$$\mathbb{E}[\ell(g(X), Y)].$$

with  $\ell(g(\mathbf{x}), y) = e^{-yg(\mathbf{x})}$ .

- ▶ Thus the output  $\hat{g}(\mathbf{x})$  is an estimator of

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

- ▶ Hence:

$$\mathbb{P}[Y = 1|X = \mathbf{x}] = \frac{e^{g^*(\mathbf{x})}}{e^{-g^*(\mathbf{x})} + e^{g^*(\mathbf{x})}},$$

$$\mathbb{P}[Y = -1|X = \mathbf{x}] = \frac{e^{-g^*(\mathbf{x})}}{e^{-g^*(\mathbf{x})} + e^{g^*(\mathbf{x})}}.$$



## Connection to AdaBoost

- ▶ One can show (Hastie, Tibshirani, Friedman, 2000) that the AdaBoost algorithm is a gradient descent method for minimizing

$$\mathbb{E}[\ell(g(X), Y)].$$

with  $\ell(g(\mathbf{x}), y) = e^{-yg(\mathbf{x})}$ .

- ▶ Thus the output  $\hat{g}(\mathbf{x})$  is an estimator of

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

- ▶ Hence:

$$\mathbb{P}[Y = 1|X = \mathbf{x}] = \frac{e^{g^*(\mathbf{x})}}{e^{-g^*(\mathbf{x})} + e^{g^*(\mathbf{x})}},$$

$$\mathbb{P}[Y = -1|X = \mathbf{x}] = \frac{e^{-g^*(\mathbf{x})}}{e^{-g^*(\mathbf{x})} + e^{g^*(\mathbf{x})}}.$$

- ▶ The **gradient descent** algorithm can be applied to **further loss functions**.

# LogitBoost

- ▶ Suppose (here) that  $Y$  takes values from  $\{0, 1\}$ .

# LogitBoost

- ▶ Suppose (here) that  $Y$  takes values from  $\{0, 1\}$ .
- ▶ The conditional random variable  $Y|X = \mathbf{x}$  follows the Bernoulli distribution with parameter  $p(\mathbf{x}) = \mathbb{P}[Y = 1|X = \mathbf{x}]$ , and its **likelihood as a function of  $p(\mathbf{x})$**  for an observation  $(\mathbf{x}, y)$  can be written as:

$$p(\mathbf{x})^y (1 - p(\mathbf{x}))^{1-y}.$$

# LogitBoost

- ▶ Suppose (here) that  $Y$  takes values from  $\{0, 1\}$ .
- ▶ The conditional random variable  $Y|X = \mathbf{x}$  follows the Bernoulli distribution with parameter  $p(\mathbf{x}) = \mathbb{P}[Y = 1|X = \mathbf{x}]$ , and its **likelihood as a function of  $p(\mathbf{x})$**  for an observation  $(\mathbf{x}, y)$  can be written as:

$$p(\mathbf{x})^y (1 - p(\mathbf{x}))^{1-y}.$$

- ▶ The **logistic regression model** assumes:

$$p(\mathbf{x}) = \frac{1}{1 + e^{-(\beta_0 + \beta^T \mathbf{x})}} = \frac{e^{\beta_0 + \beta^T \mathbf{x}}}{1 + e^{\beta_0 + \beta^T \mathbf{x}}},$$

where  $(\beta_0, \beta^T)^T$  is estimated by maximizing the likelihood.

# LogitBoost

- ▶ Suppose (here) that  $Y$  takes values from  $\{0, 1\}$ .
- ▶ The conditional random variable  $Y|X = \mathbf{x}$  follows the Bernoulli distribution with parameter  $p(\mathbf{x}) = \mathbb{P}[Y = 1|X = \mathbf{x}]$ , and its **likelihood as a function of  $p(\mathbf{x})$**  for an observation  $(\mathbf{x}, y)$  can be written as:

$$p(\mathbf{x})^y (1 - p(\mathbf{x}))^{1-y}.$$

- ▶ The **logistic regression model** assumes:

$$p(\mathbf{x}) = \frac{1}{1 + e^{-(\beta_0 + \beta^T \mathbf{x})}} = \frac{e^{\beta_0 + \beta^T \mathbf{x}}}{1 + e^{\beta_0 + \beta^T \mathbf{x}}},$$

where  $(\beta_0, \beta^T)^T$  is estimated by maximizing the likelihood.

- ▶ One can use the functional gradient descent method to **overcome the linearity assumption** on parameters of  $(1, \mathbf{x}^T)^T$  in  $p(\mathbf{x})$ .

# LogitBoost

- ▶ One can rewrite:

$$p(\mathbf{x}) = \frac{1}{1 + e^{-2f(\mathbf{x})}} = \frac{e^{f(\mathbf{x})}}{1 + e^{f(\mathbf{x})}}$$

with  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  being an unknown function.

# LogitBoost

- ▶ One can rewrite:

$$p(\mathbf{x}) = \frac{1}{1 + e^{-2f(\mathbf{x})}} = \frac{e^{f(\mathbf{x})}}{1 + e^{f(\mathbf{x})}}$$

with  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  being an unknown function.

- ▶ As it is done in logistic regression, one can **estimate  $f(\mathbf{x})$  based on the likelihood**.

# LogitBoost

- ▶ One can rewrite:

$$p(\mathbf{x}) = \frac{1}{1 + e^{-2f(\mathbf{x})}} = \frac{e^{f(\mathbf{x})}}{1 + e^{f(\mathbf{x})}}$$

with  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  being an unknown function.

- ▶ As it is done in logistic regression, one can **estimate  $f(\mathbf{x})$  based on the likelihood**.
- ▶ **Choice of the loss function:** the negative log-likelihood (which will be minimized):

$$-\left(y \log p(\mathbf{x}) + (1 - y) \log(1 - p(\mathbf{x}))\right) = \log(1 + e^{-2\tilde{y}f(\mathbf{x})})$$

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



# LogitBoost

- ▶ One can rewrite:

$$p(\mathbf{x}) = \frac{1}{1 + e^{-2f(\mathbf{x})}} = \frac{e^{f(\mathbf{x})}}{1 + e^{f(\mathbf{x})}}$$

with  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  being an unknown function.

- ▶ As it is done in logistic regression, one can **estimate  $f(\mathbf{x})$  based on the likelihood**.
- ▶ **Choice of the loss function:** the negative log-likelihood (which will be minimized):

$$-\left(y \log p(\mathbf{x}) + (1 - y) \log(1 - p(\mathbf{x}))\right) = \log(1 + e^{-2\tilde{y}f(\mathbf{x})})$$

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

- ▶ One can simply verify convexity of:

$$v \mapsto \log(1 + e^{-2\tilde{y}v}).$$

# LogitBoost

- ▶ The functional gradient descent algorithm applied to the loss function:

$$\begin{aligned} \ell : \mathbb{R} \times \{-1, 1\} &\rightarrow \mathbb{R} \\ (f(\mathbf{x}), \tilde{y}) &\mapsto \log(1 + e^{-2\tilde{y}f(\mathbf{x})}) \end{aligned}$$

is called **LogitBoost**.

# LogitBoost

- ▶ The functional gradient descent algorithm applied to the loss function:

$$\begin{aligned} \ell : \mathbb{R} \times \{-1, 1\} &\rightarrow \mathbb{R} \\ (f(\mathbf{x}), \tilde{y}) &\mapsto \log(1 + e^{-2\tilde{y}f(\mathbf{x})}) \end{aligned}$$

is called **LogitBoost**.

- ▶ For this loss function, the function  $\mathbb{E}[\ell(f(X), Y)]$  is minimized at:

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

# LogitBoost

- ▶ The functional gradient descent algorithm applied to the loss function:

$$\begin{aligned} \ell : \mathbb{R} \times \{-1, 1\} &\rightarrow \mathbb{R} \\ (f(\mathbf{x}), \tilde{y}) &\mapsto \log(1 + e^{-2\tilde{y}f(\mathbf{x})}) \end{aligned}$$

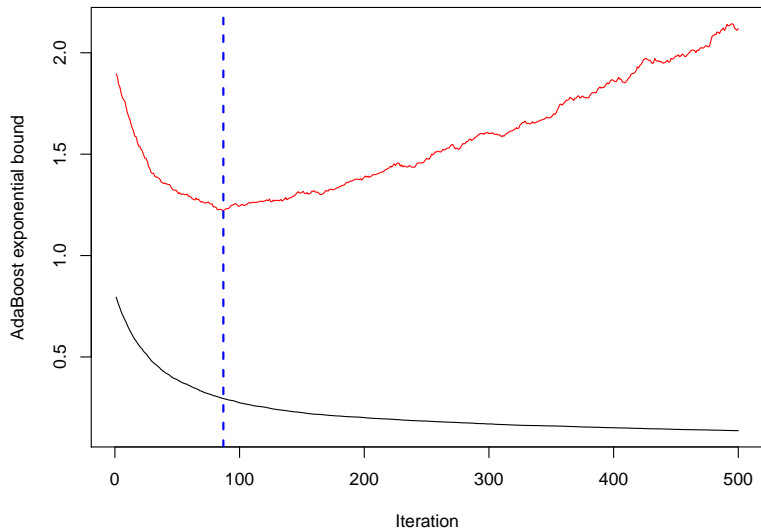
is called **LogitBoost**.

- ▶ For this loss function, the function  $\mathbb{E}[\ell(f(X), Y)]$  is minimized at:

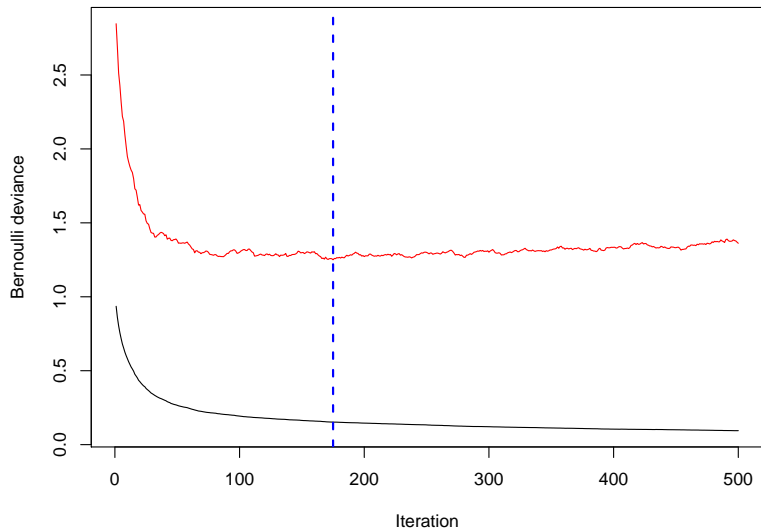
$$f^*(\mathbf{x}) = \frac{1}{2} \log \frac{\mathbb{P}[Y = 1|X = \mathbf{x}]}{\mathbb{P}[Y = -1|X = \mathbf{x}]}.$$

- ▶ Thus, AdaBoost and LogitBoost provide an estimator of the same quantity.

# Performance validation, spam data (AdaBoost, FGD)



# Performance validation, spam data (LogitBoost, FGD)



Thank you for your attention!

Thank you for your attention!

## And some references

- ▶ Bühlmann, P. and Hothorn, T. (2007).  
Boosting algorithms: Regularization, prediction and model fitting.  
*Statistical Science*, 22, 477–505.
- ▶ Friedman, J. H. (2001).  
Greedy function approximation: A gradient boosting machine.  
*The Annals of Statistics*, 29, 1189–1232.
- ▶ Friedman, J., Hastie, T., and Tibshirani, R. (2000).  
Additive logistic regression: A statistical view of boosting.  
*The Annals of Statistics*, 28, 337–407.
- ▶ Hastie, T., Tibshirani, R., and Friedman, J. (2009).  
*The Elements of Statistics Learning: Data Mining, Inference, and Prediction (Second Edition)*.  
Springer.
- ▶ Schapire, R. E. and Freund, Y. (2012).  
*Boosting: foundations and algorithms*.  
MIT Press.