Boosting algorithms

Pavlo Mozharovskyi[1]

(with contributions of Laurent Rouviere[2] and Valentin Patilea[3])

[1]LTCI, Télécom Paris, Institut Polytechnique de Paris
[2]Université Rennes 2
[3]Ensai, 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.: $(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)$.
- **Goal:** predict the label of the new (unseen before) observation $\boldsymbol{x}$.
- **Method:** construct a classification rule:

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

  so $g(\boldsymbol{x})$ is the prediction of the label for observation $\boldsymbol{x}$.
- **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):

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

Contents

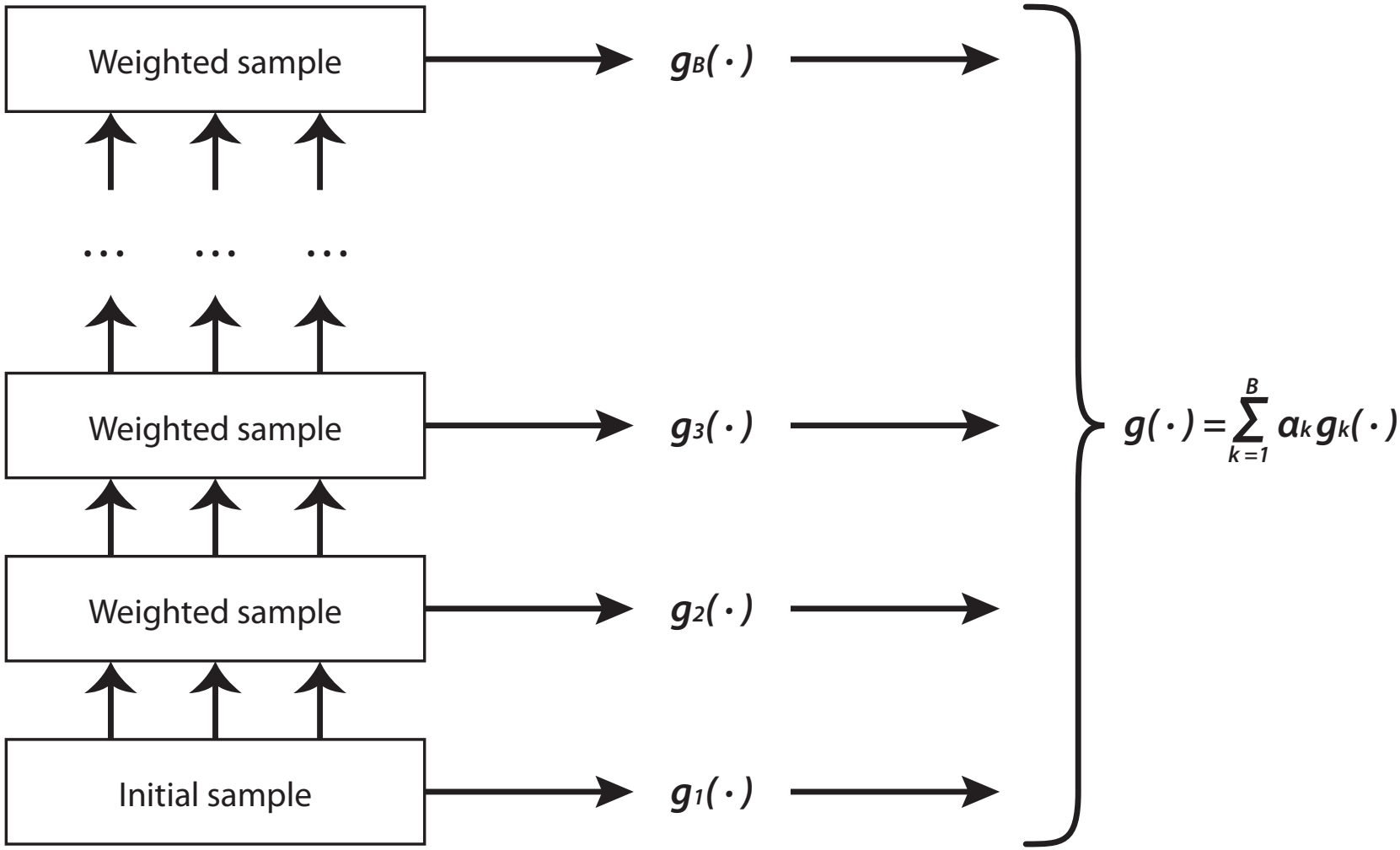
Contents

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 kth 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



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 $g_\cdot(\cdot)$ is called a **weak learner** if it performs "slightly" better than a random guess:

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

- **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 $\big((\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)\big) \subset \mathbb{R}^d \times \{-1, 1\}$.
- Weak learner $g_\cdot(\cdot)$, number of iterations B.

1. Initialize weights $w_i = \frac{1}{n}$, $i = 1, ..., n$.
2. For $k = 1, ..., B$
   - 2.1 Learn a weak learner $g_k(\cdot)$ on $\mathcal{D}_n$ weighted by w_i, $i = 1, ..., n$.
   - 2.2 Calculate error rate:
   $$e_k = \sum_{i=1}^{n} w_i \mathbb{1}\big(g_k(\boldsymbol{x}_i) \neq y_i\big) .$$
   - 2.3 Calculate learner's weight: $\alpha_k = \log \frac{1-e_k}{e_k}$.
   - 2.4 Readjust observations' weights: $w_i = w_i e^{\alpha_k \mathbb{1}\left(g_k(\boldsymbol{x}_i) \neq y_i\right)}$, $i = 1, ..., n$; normalize to get $\sum_{i=1}^{n} w_i = 1$.

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

Weighted majority voting (classification)

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

Contents

AdaBoost: some comments

- The step of learning requires the weak learner to take into account the **weights** w_i, $i = 1, ..., n$ 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, $i = 1, ..., n$.
- Observations' weights w_i, $i = 1, ..., n$ are updated on each iteration:
  - if the ith observation is **well classified**, then its weight is unchanged,
  - if the ith observation is **wrongly classified**, then its weight is increased.
- Learner's weight α_k **increases with performance of** $g_k(\cdot)$ calculated on (weighted) $\mathcal{D}_n$:
  - α_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:

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

- Denote ϵ_k the **gain** of g_k **w.r.t. a purely random classifier**:

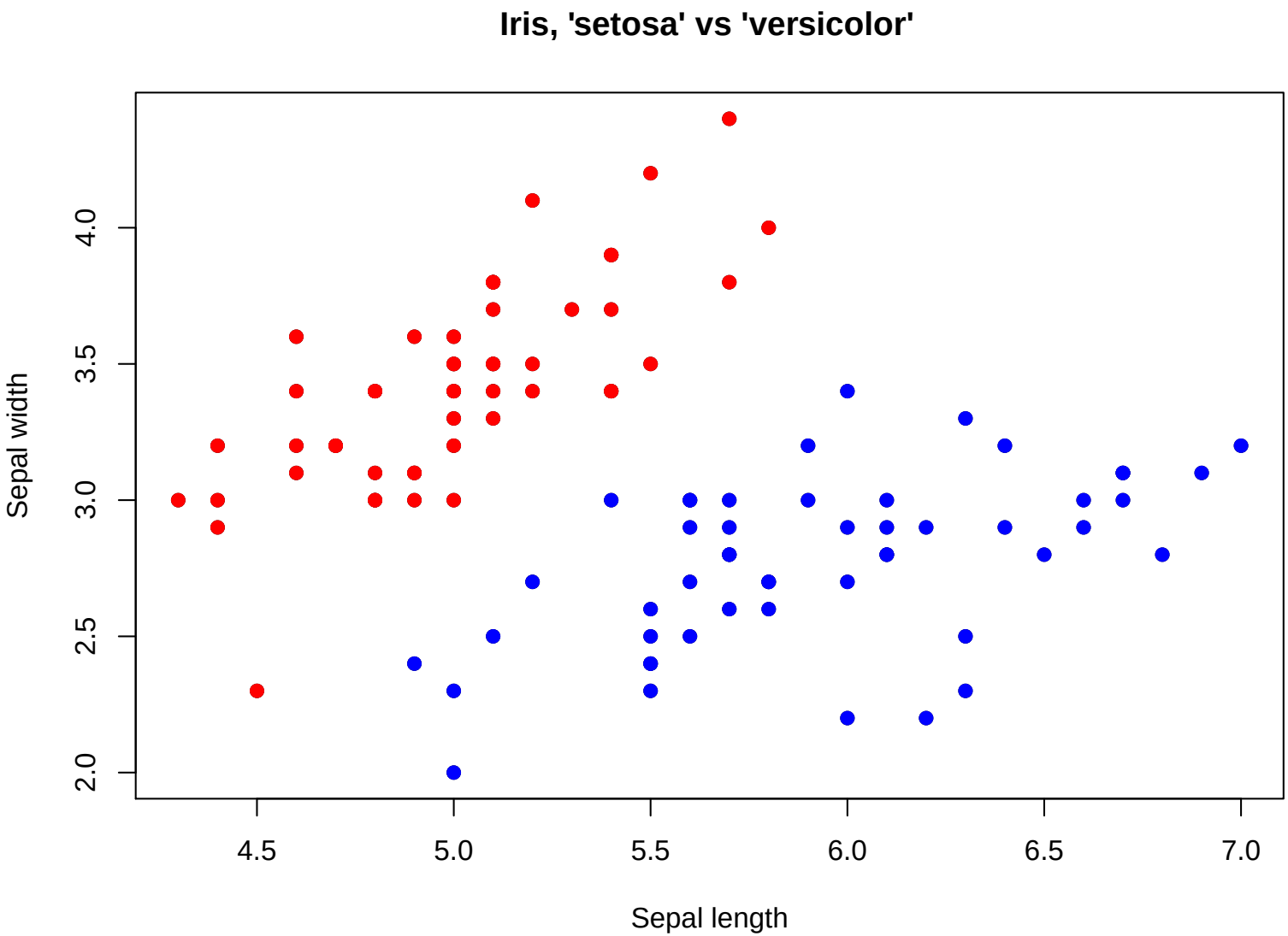
$$e_k = \frac{1}{2} - \epsilon_k \,.$$

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

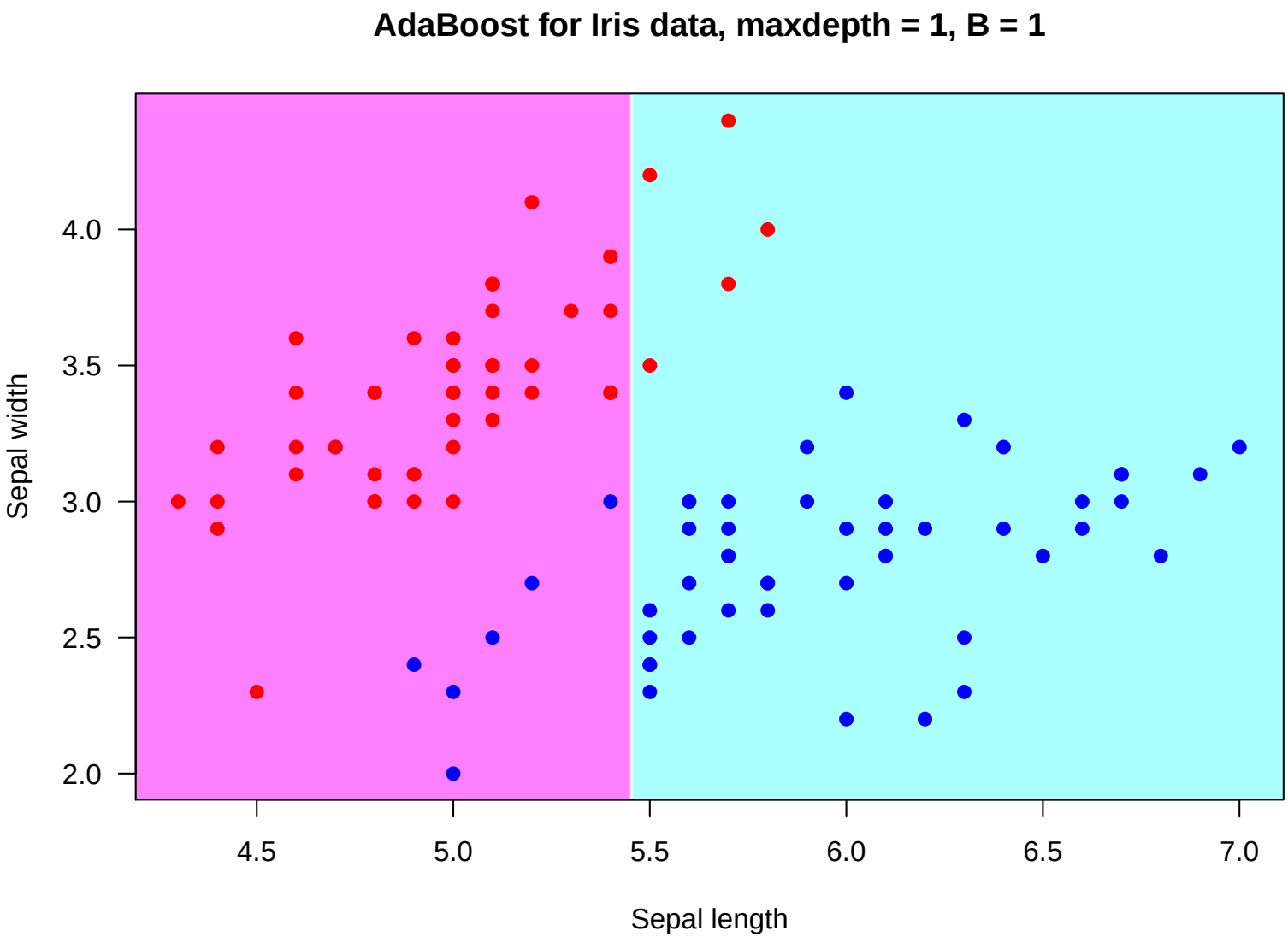
$$L_n(g^{bst}) \leq e^{-2\sum_{k=1}^{B} \epsilon_k^2} \,.$$

- 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.

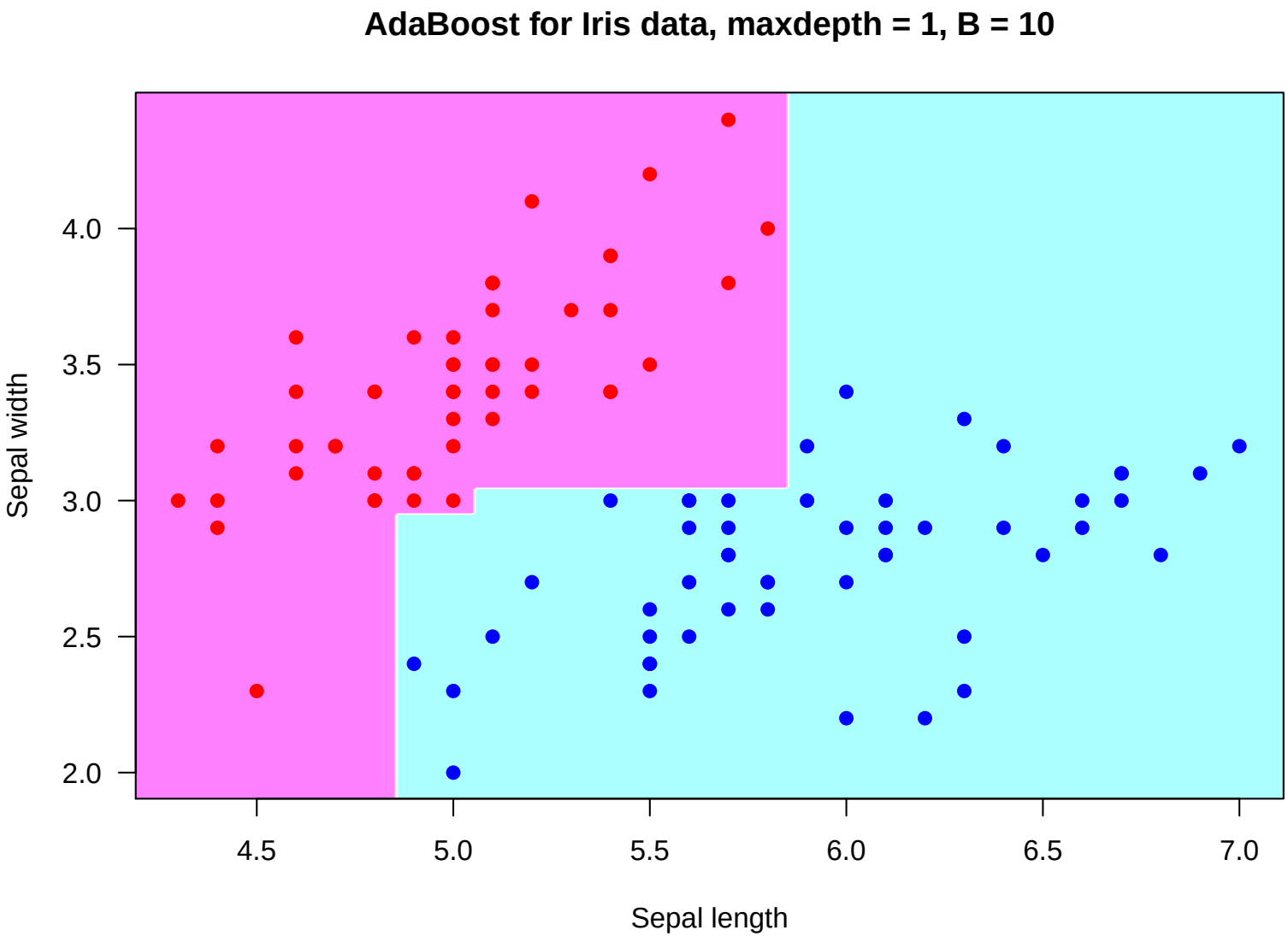
AdaBoost (Iris data, "setosa" *vs.* "versicolor")



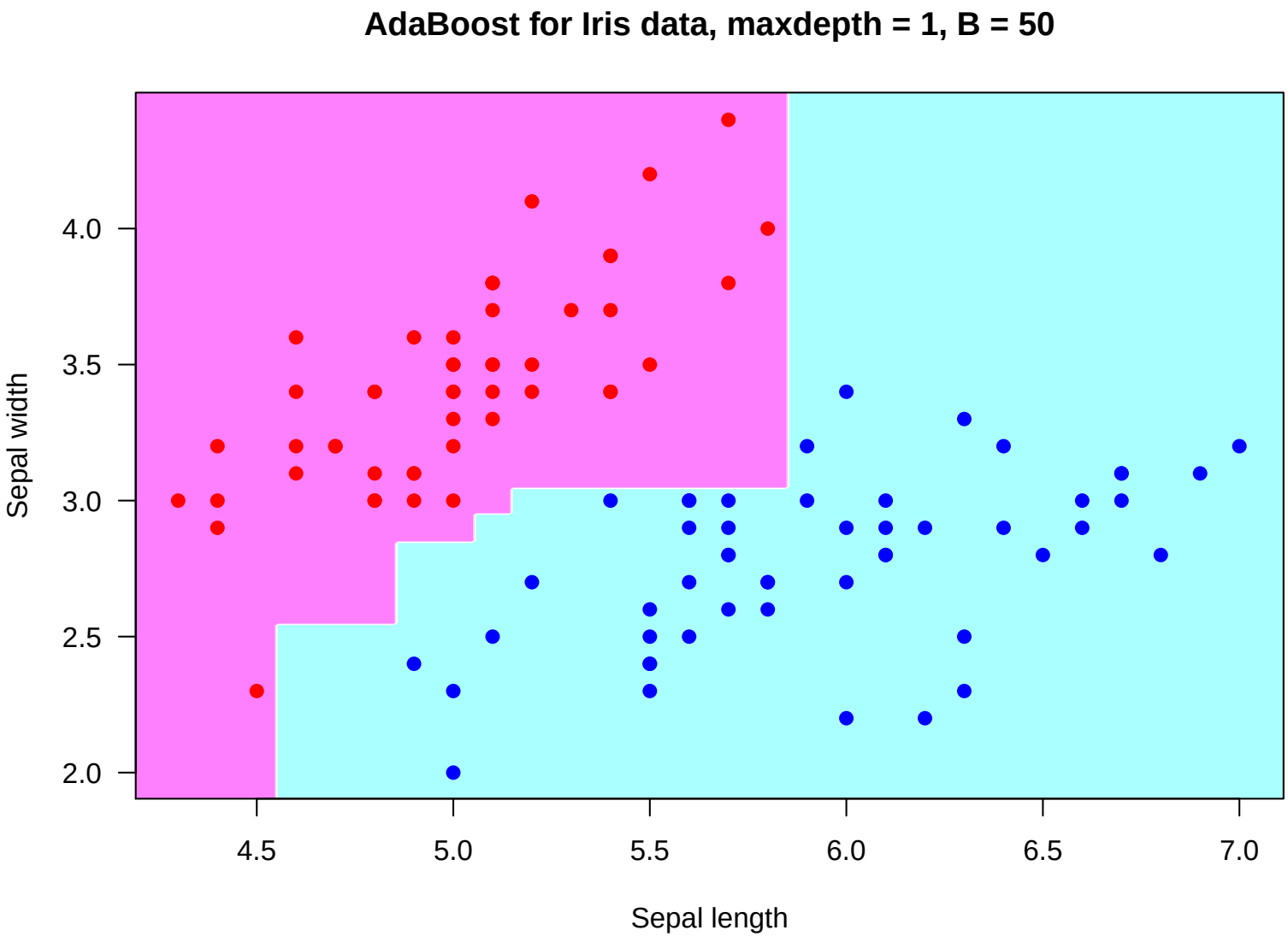
AdaBoost (Iris data, "setosa" *vs.* "versicolor")



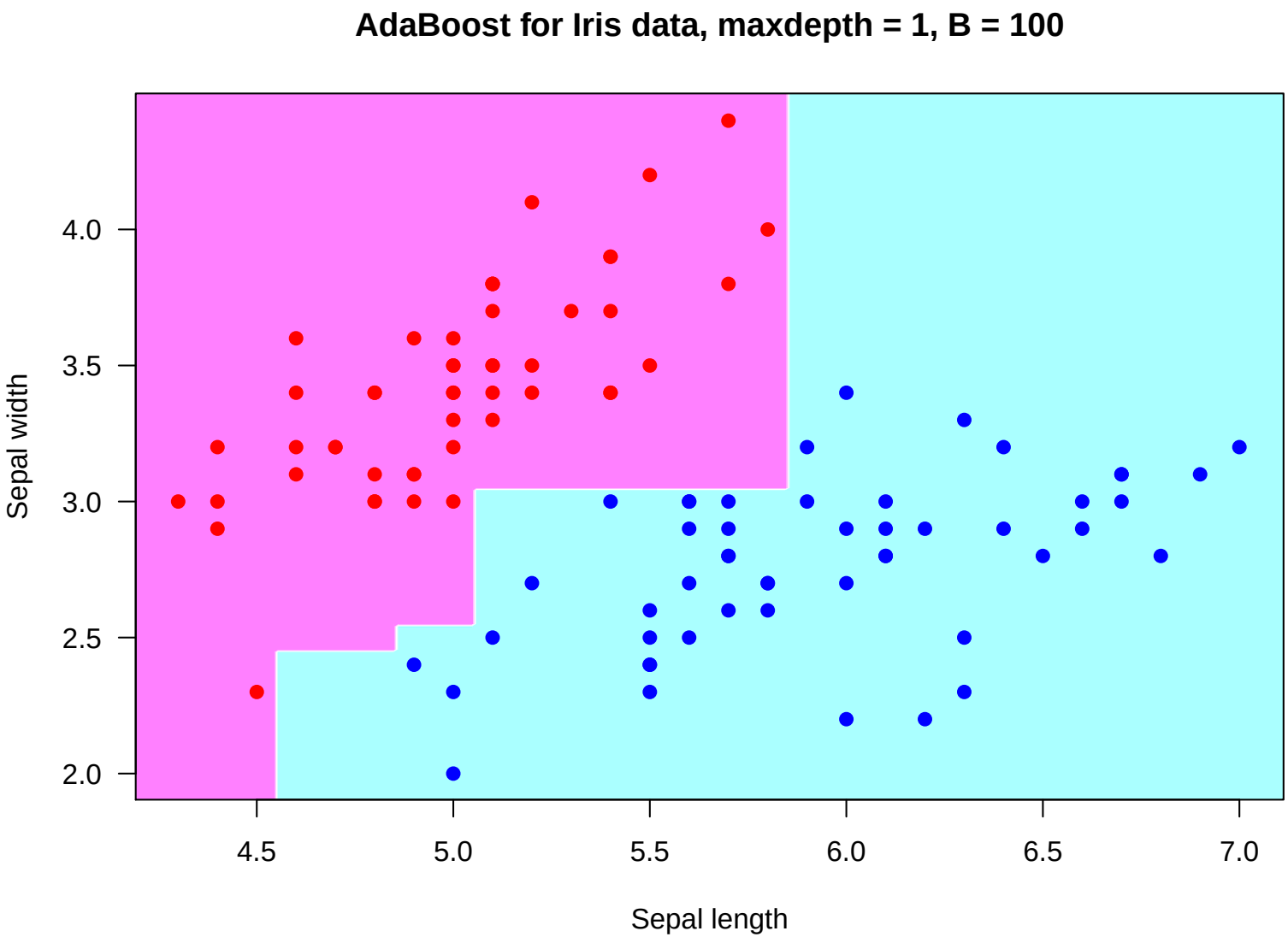
AdaBoost (Iris data, "setosa" *vs.* "versicolor")



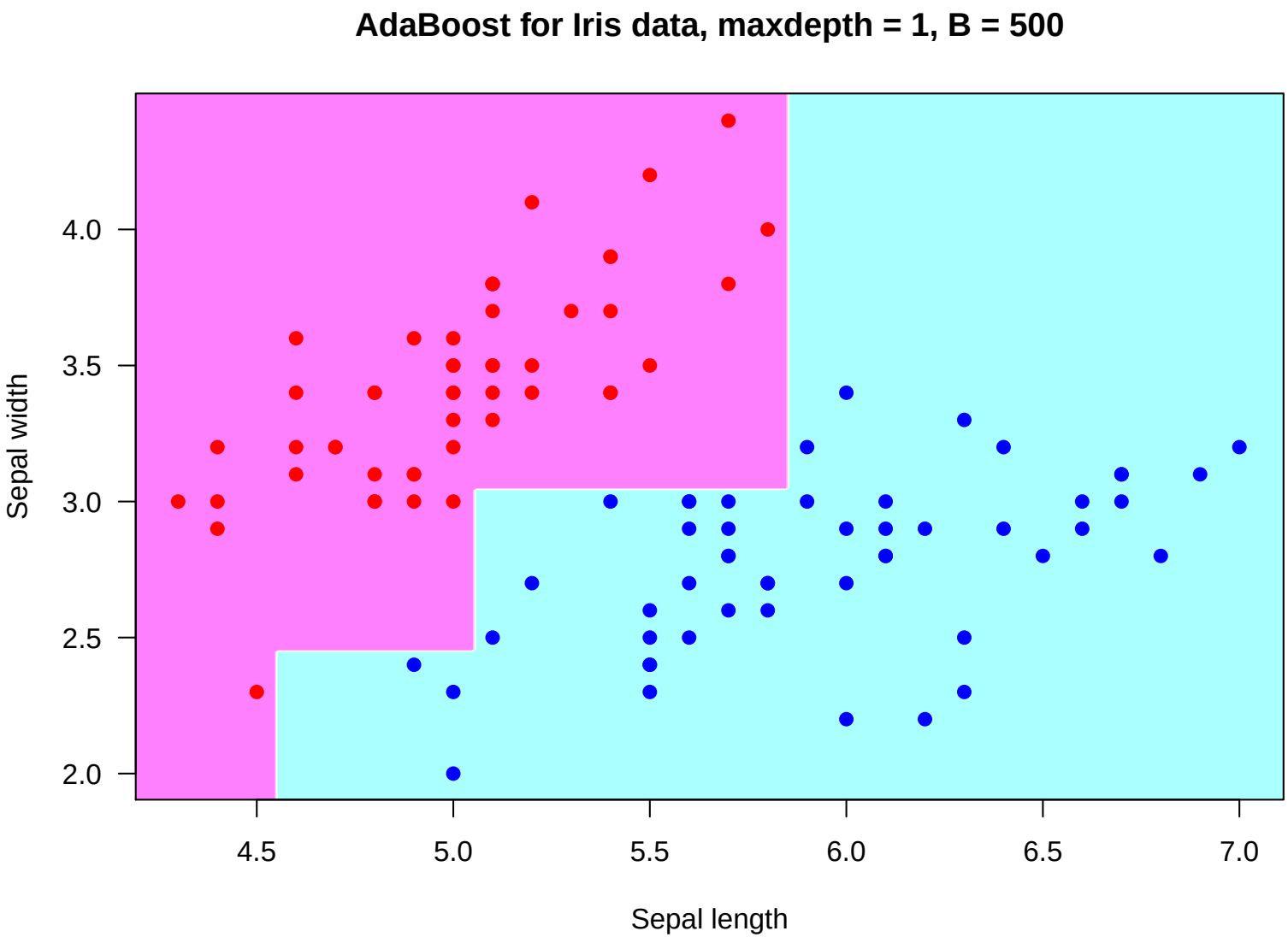
AdaBoost (Iris data, "setosa" *vs.* "versicolor")



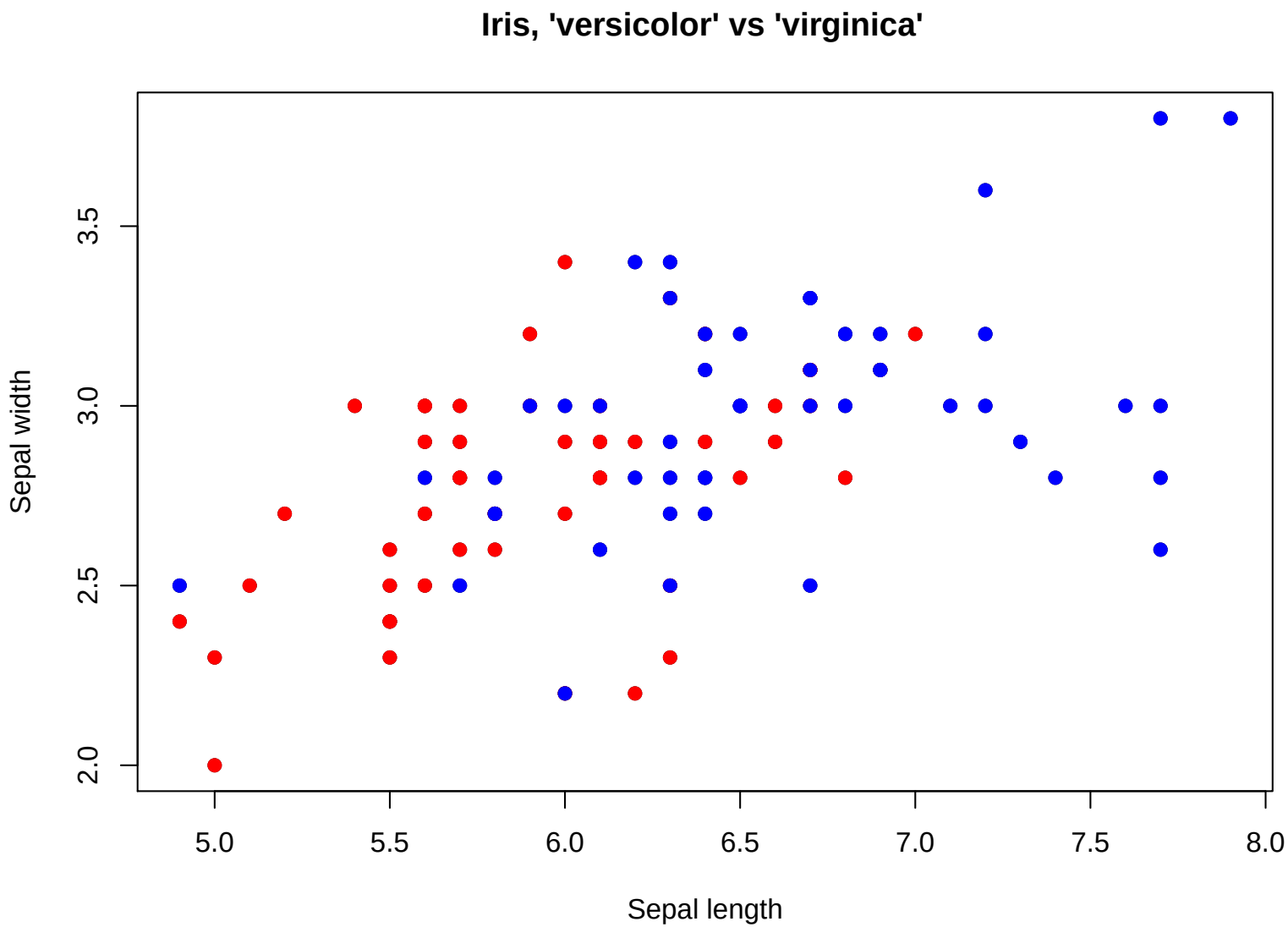
AdaBoost (Iris data, "setosa" *vs.* "versicolor")



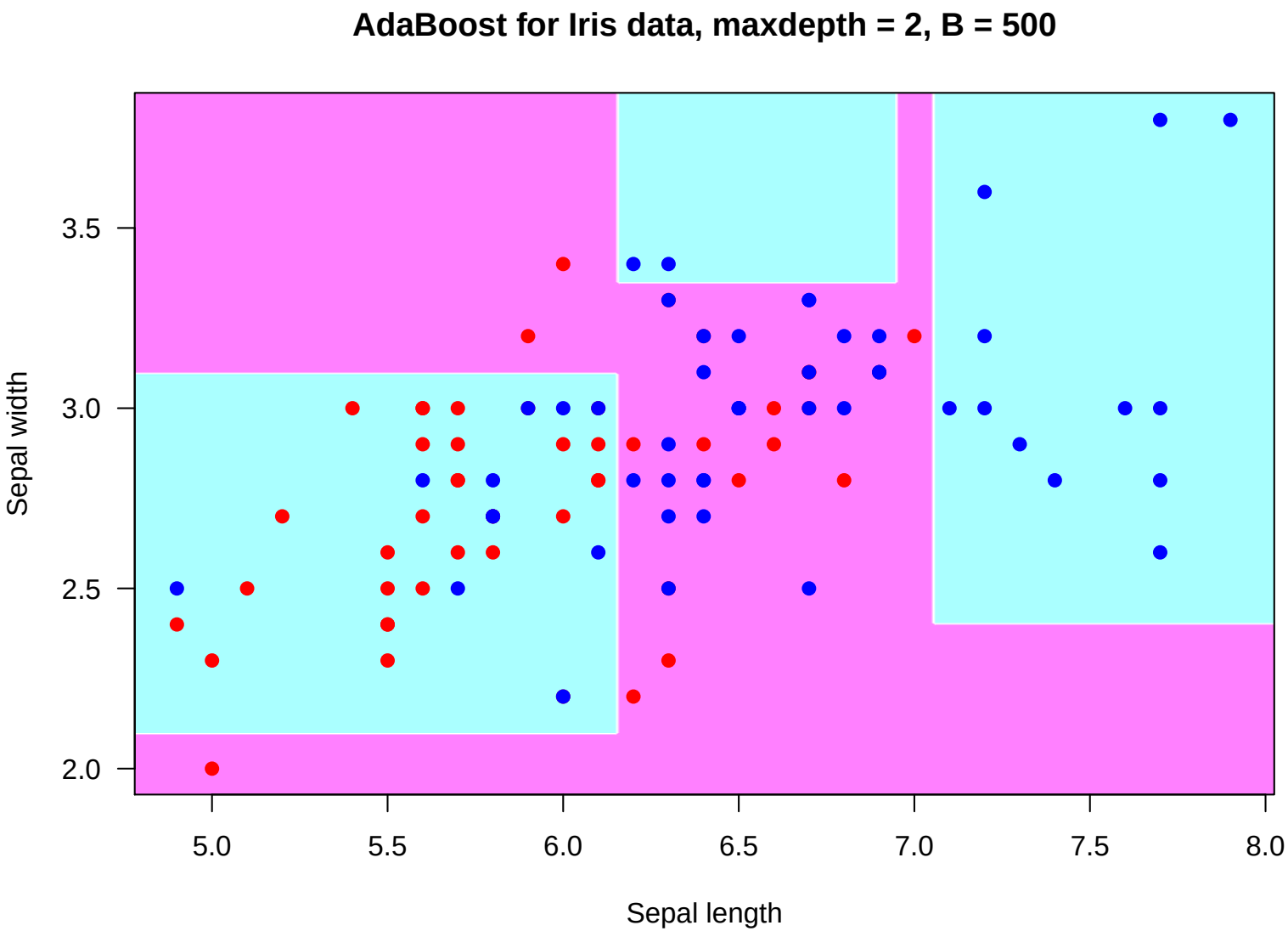
AdaBoost (Iris data, "setosa" *vs.* "versicolor")



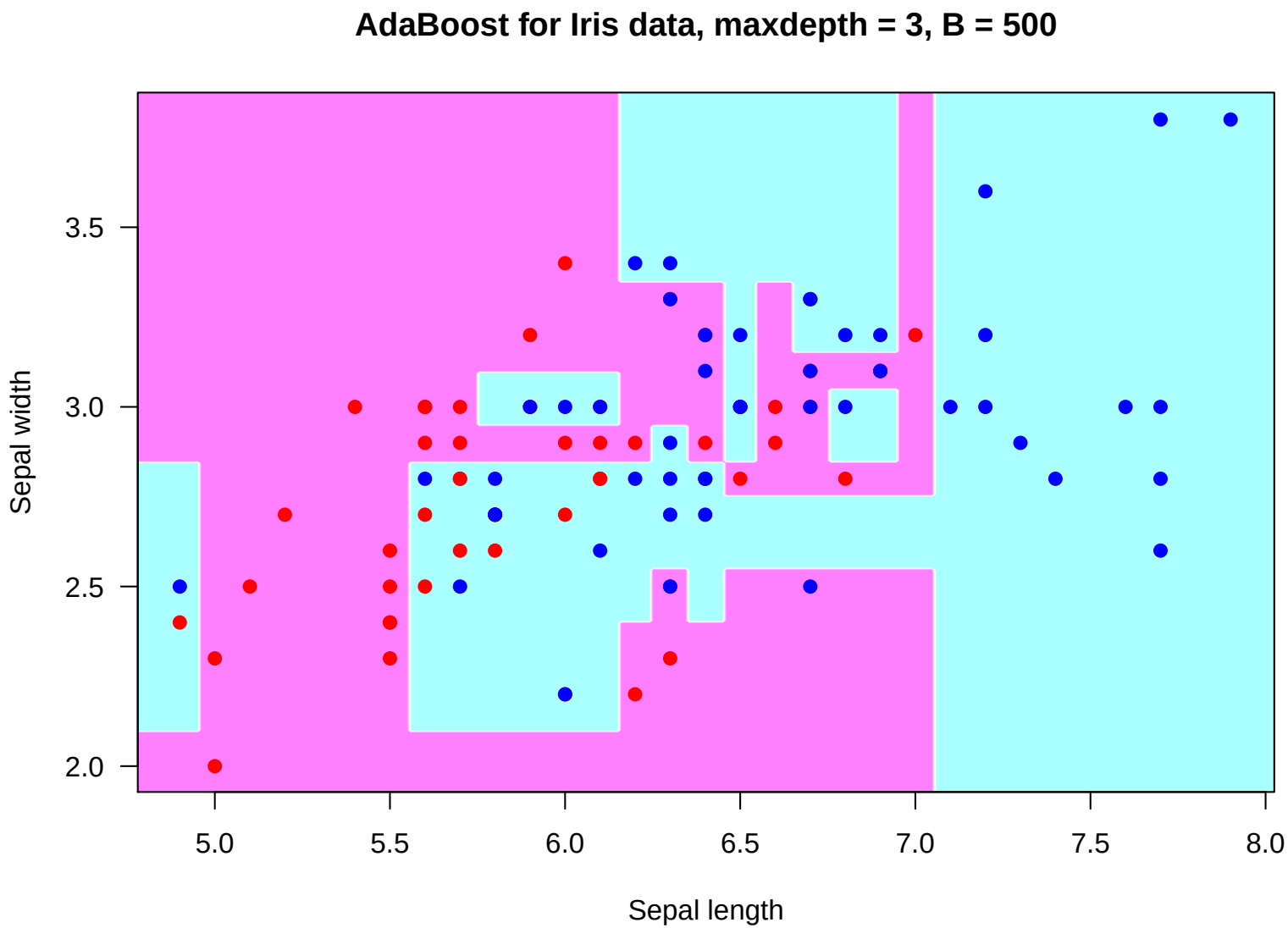
AdaBoost (Iris data, "versicolor" *vs.* "virginica")



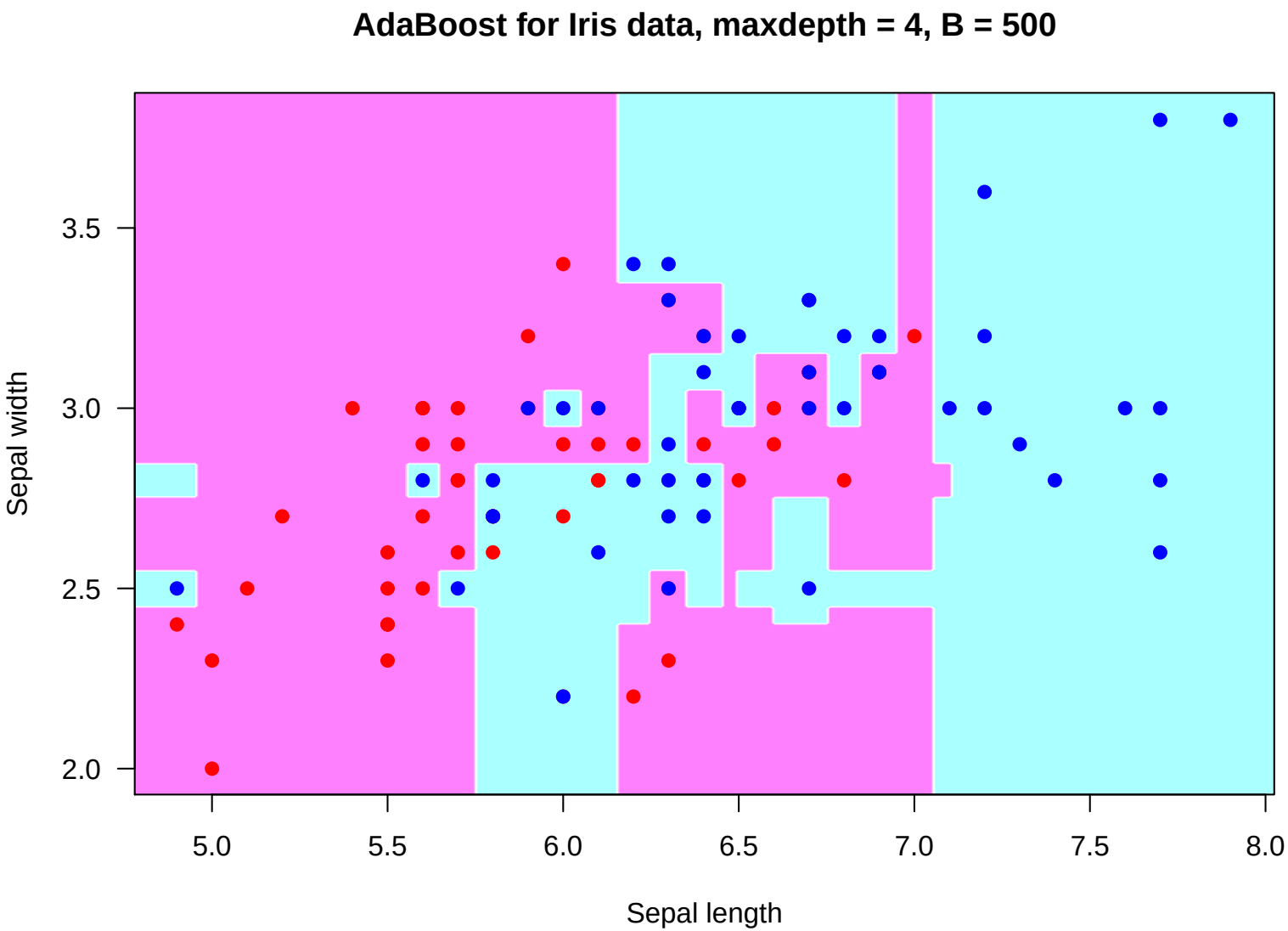
AdaBoost (Iris data, "versicolor" *vs.* "virginica")



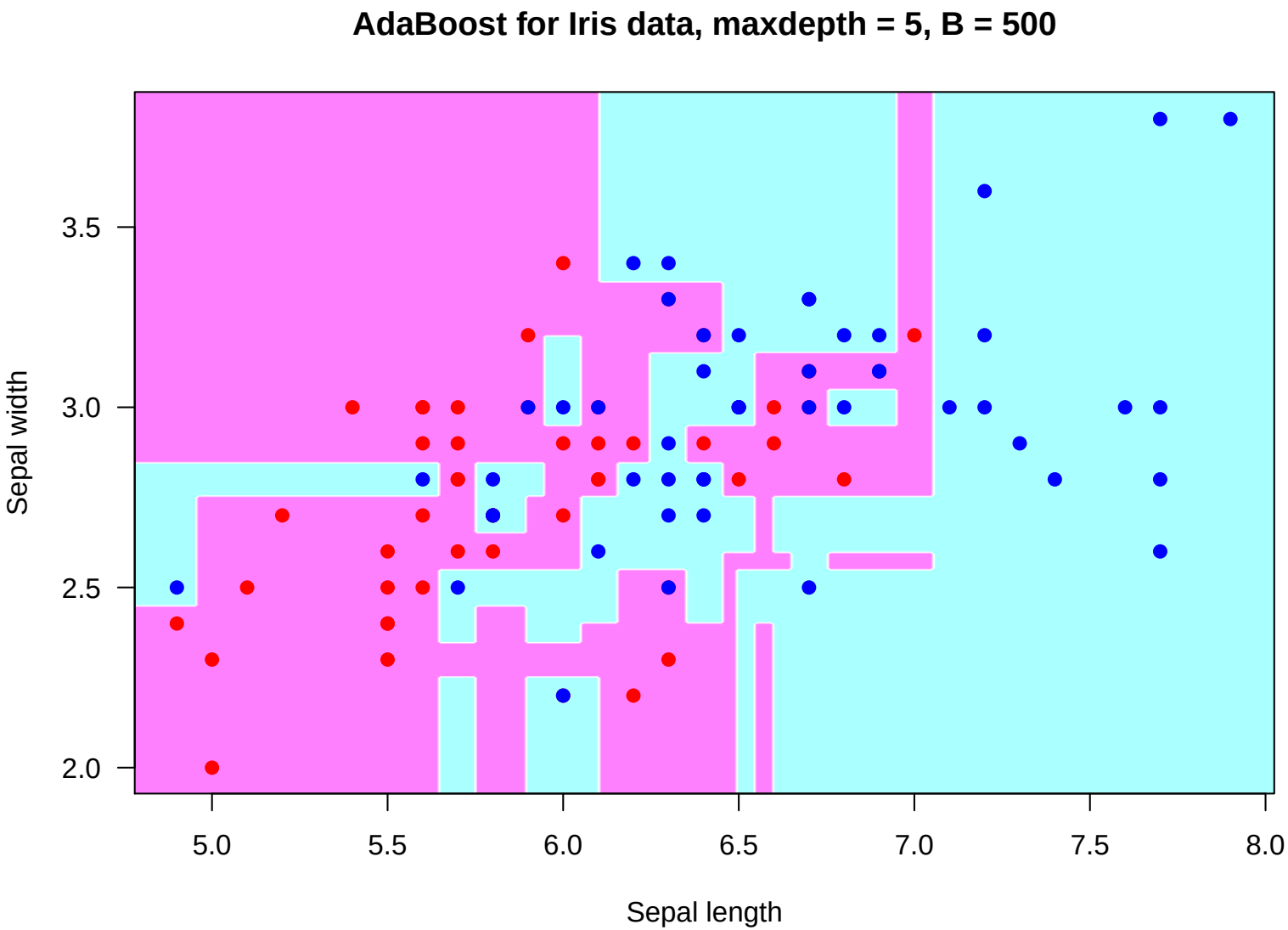
AdaBoost (Iris data, "versicolor" *vs.* "virginica")



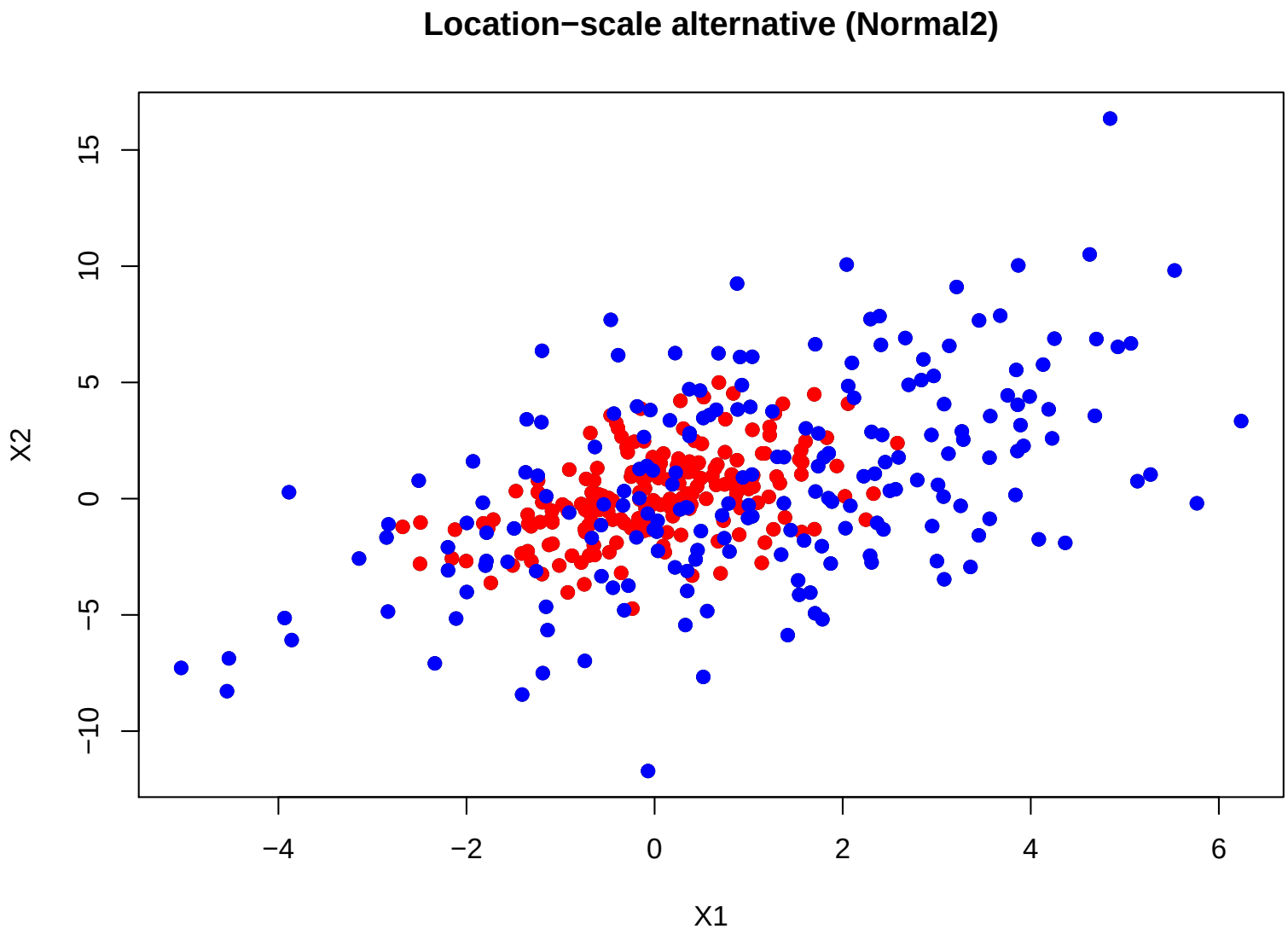
AdaBoost (Iris data, "versicolor" *vs.* "virginica")



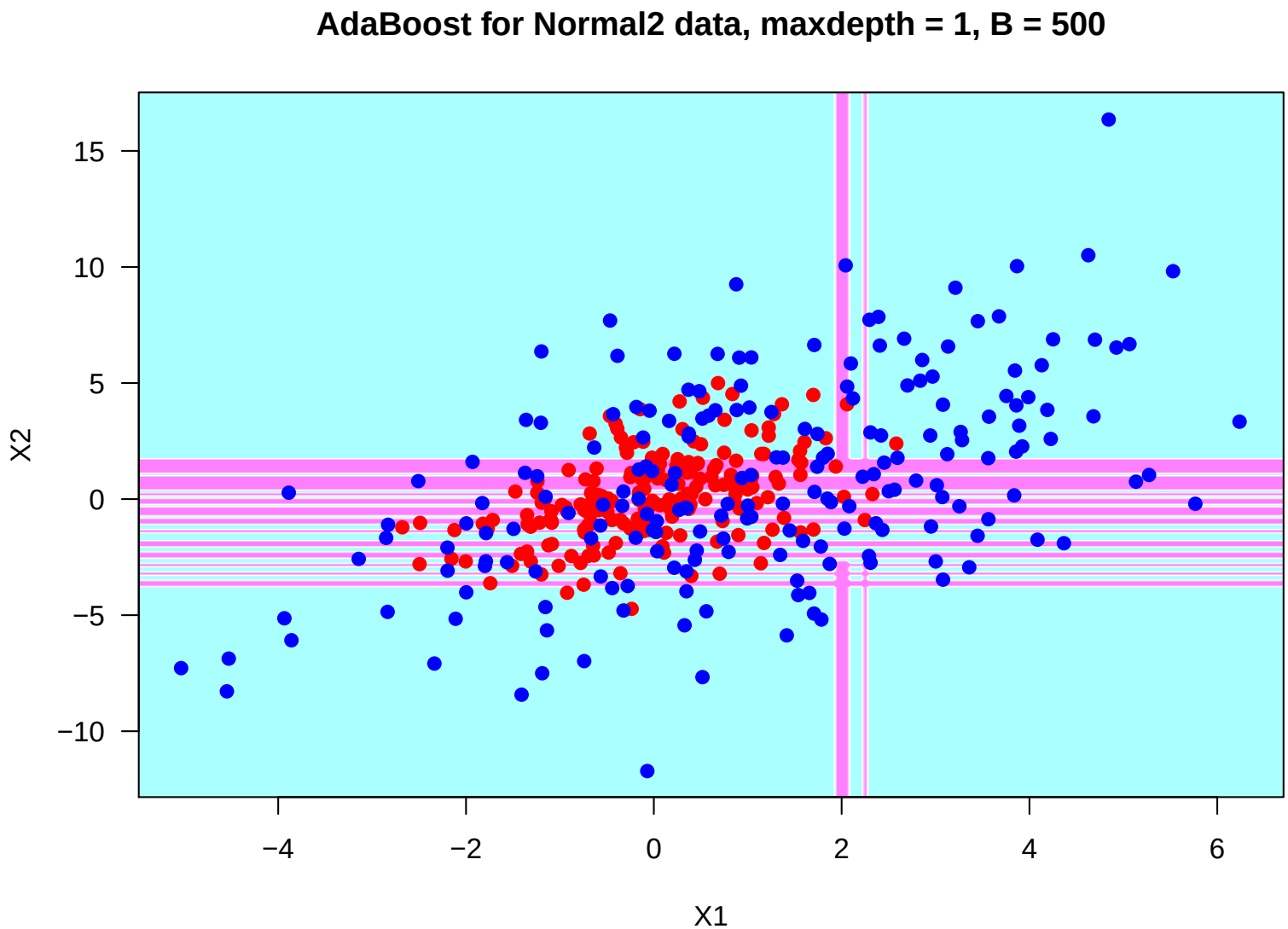
AdaBoost (Iris data, "versicolor" *vs.* "virginica")



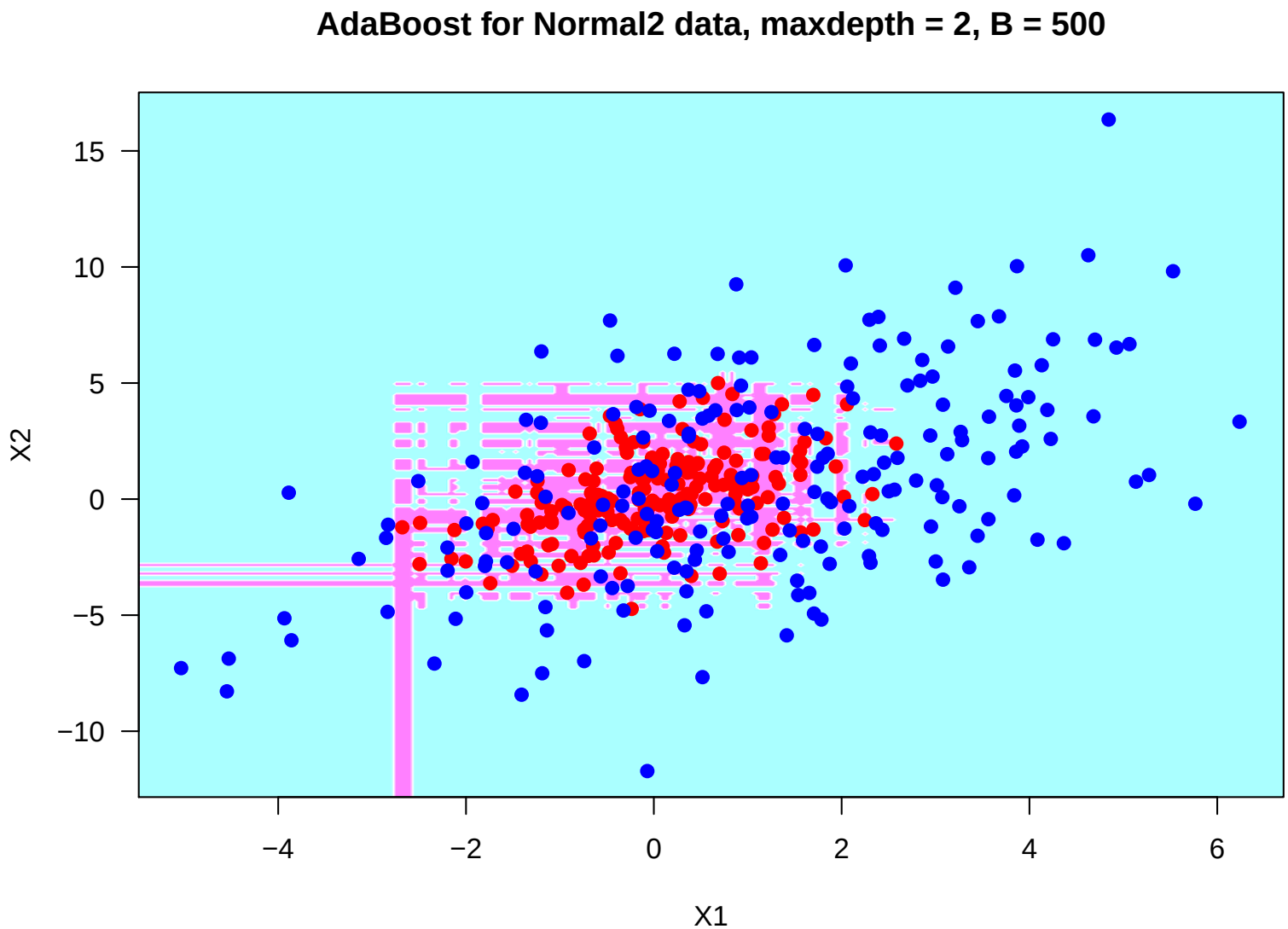
AdaBoost (normal location-scale alternative)



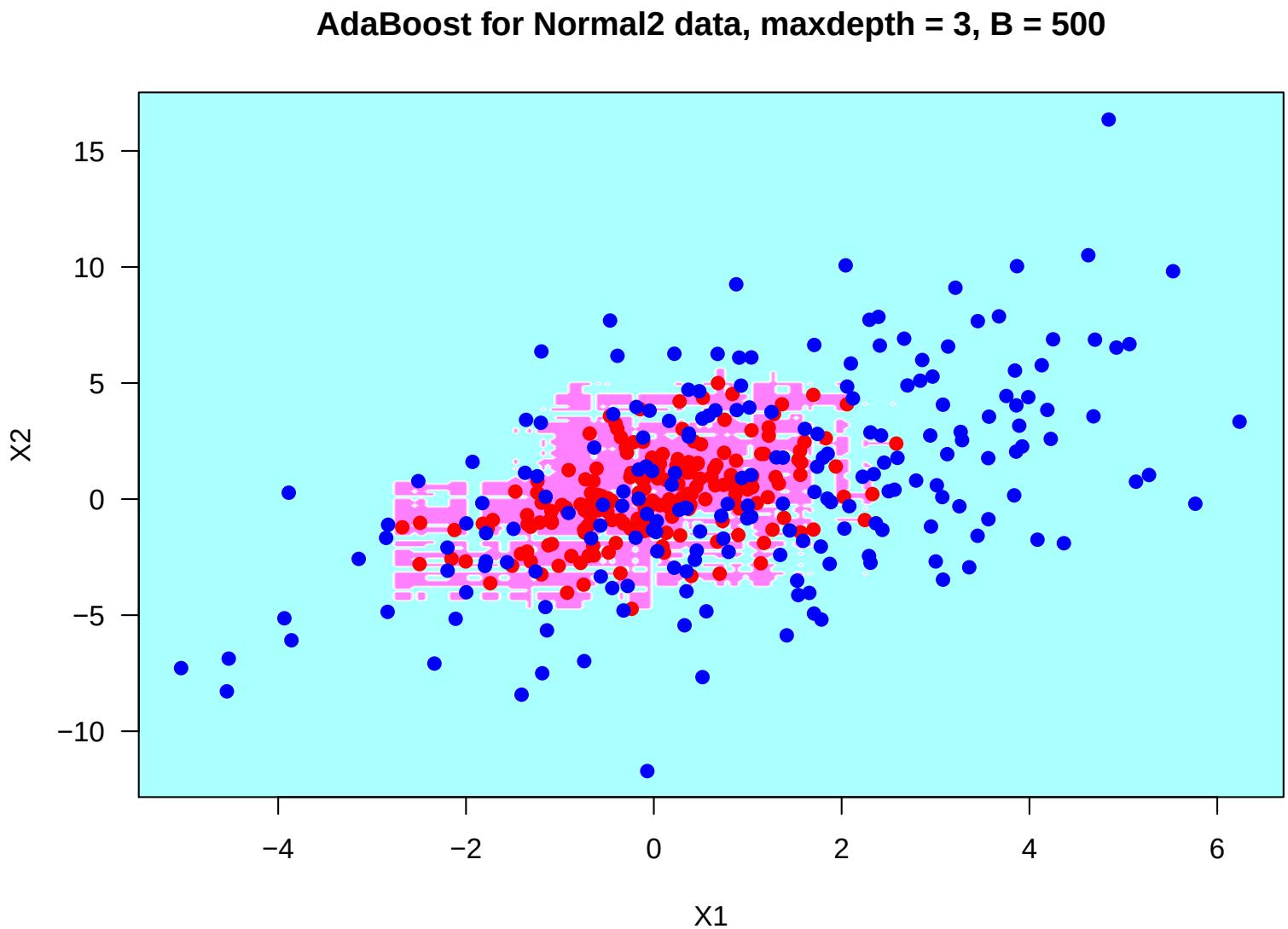
AdaBoost (normal location-scale alternative)



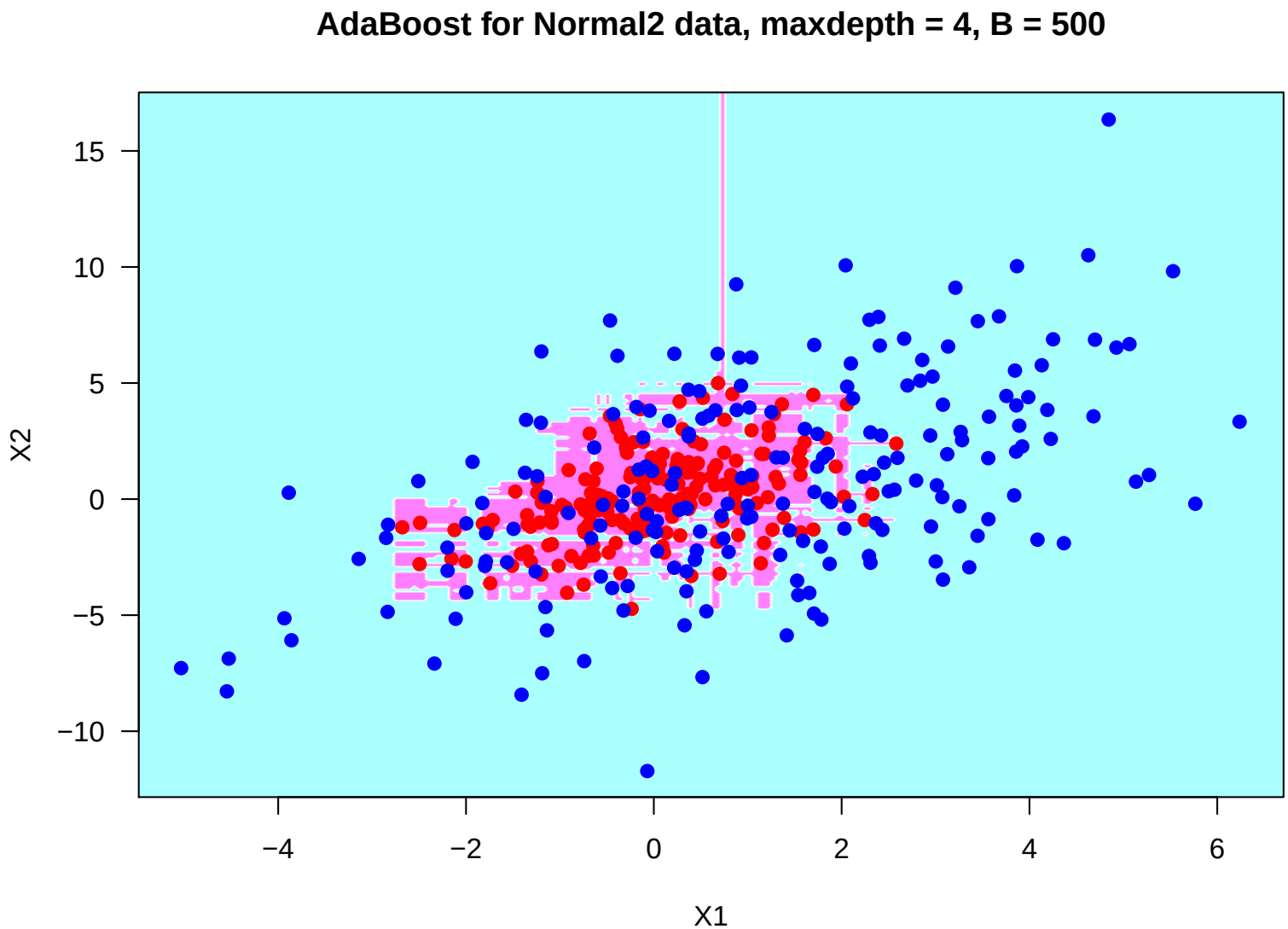
AdaBoost (normal location-scale alternative)



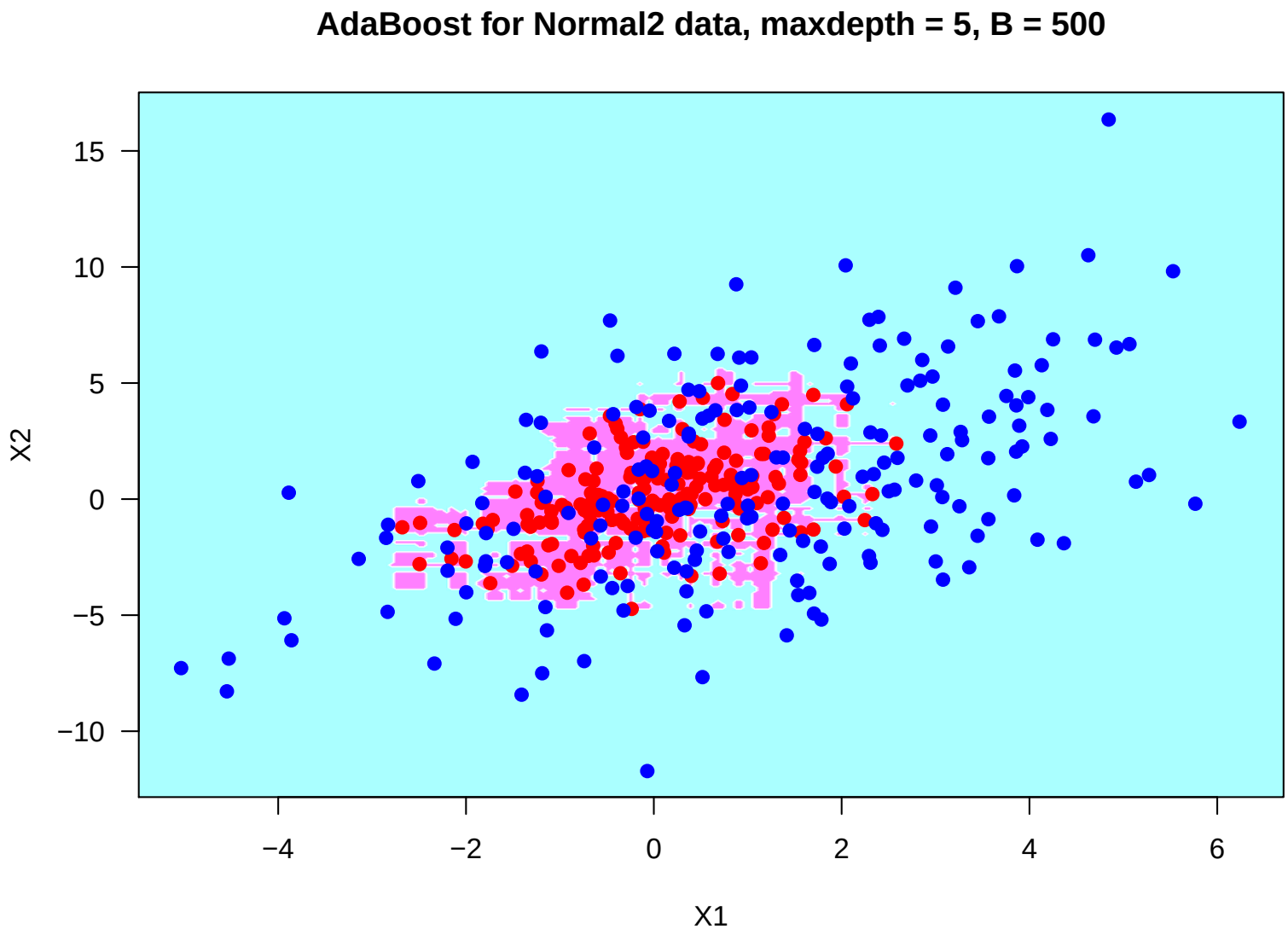
AdaBoost (normal location-scale alternative)



AdaBoost (normal location-scale alternative)



AdaBoost (normal location-scale alternative)



Contents

Contents

Minimization of the empirical risk

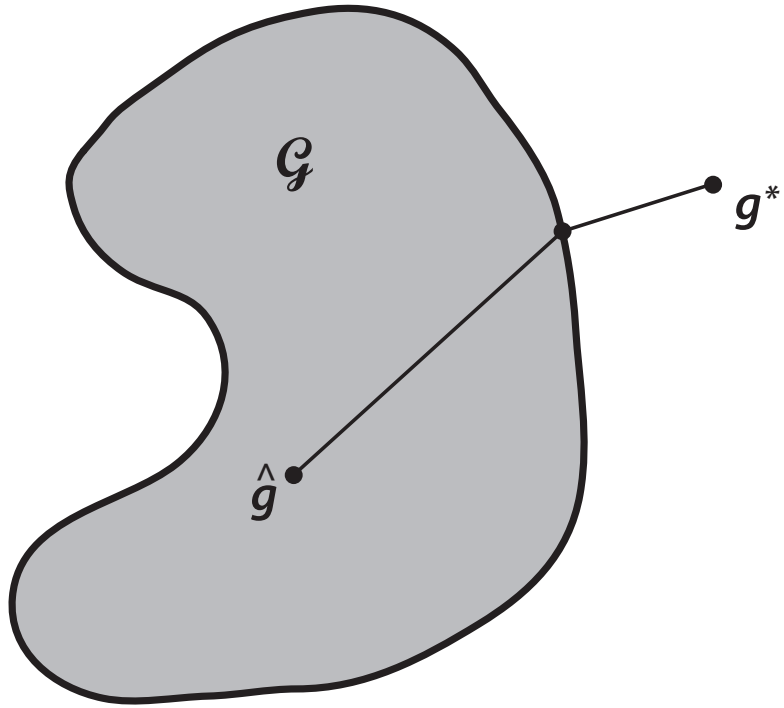
- Regard a **random pair** (X, Y) taking values in $\mathbb{R}^d \times \{-1, 1\}$.
- Consider a **class of** classification **rules** $\mathcal{G}$:
  - one attempts to find **the best rule** in $\mathcal{G}$.
- **Try:** Choose the rule which minimizes a loss function, for example:

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

- **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**:

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

Minimization of the empirical risk



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

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

Risk convexification

- **Problem:** The function

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

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

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

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

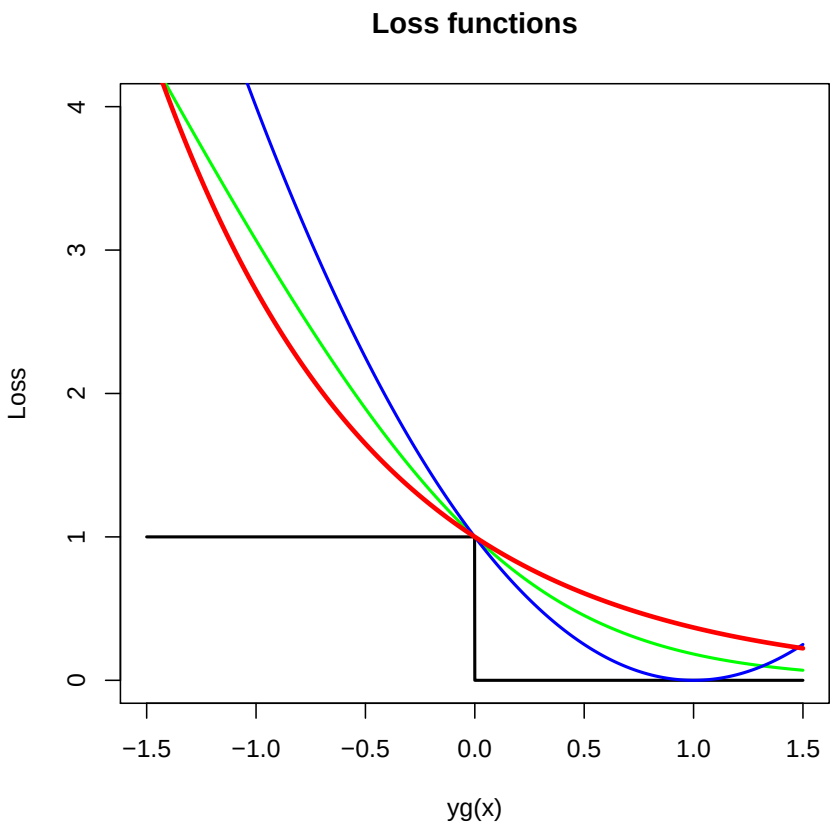
  is **"easy" to minimize**, *e.g.* a **differentiable** one.

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

- The loss function $\ell\big(g(\boldsymbol{x}), y\big)$ should **measure the difference** between the value to be predicted $y \in \{-1, 1\}$ and $g(\boldsymbol{x})$.

- Thus $\ell\big(g(\boldsymbol{x}), y\big)$ should take:
  - **large** values if $yg(\boldsymbol{x}) < 0$,
  - **small** values if $yg(\boldsymbol{x}) > 0$.

Loss functions



— Misclassification:

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

— Exponential:

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

— Binomial log-likelihood:

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

— Squared error:

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

Summary

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

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

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

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

- **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

Functional gradient descent (FGD)

- Let $\boldsymbol{g}_k = \big(g_k(\boldsymbol{x}_1), g_k(\boldsymbol{x}_2), ..., g_k(\boldsymbol{x}_n)\big)$, and

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

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

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

  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 $\boldsymbol{x}_i$ is close to $\boldsymbol{x}_j$ then $\hat{g}(\boldsymbol{x}_i)$ is close to $\hat{g}(\boldsymbol{x}_j)$.
- The estimator is calculated at the points of the training sample only $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$.

Gradient boost (algorithm): Friedman (2001)

Constructing a classifier (training)

Input:

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

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

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

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

$$\hat{y} = \text{sign}\big(\hat{g}(\boldsymbol{x})\big) \,.$$

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

- The **choice of the regularization parameter** λ is connected to the number of iterations B.
  It "controls" the speed of minimization of function

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

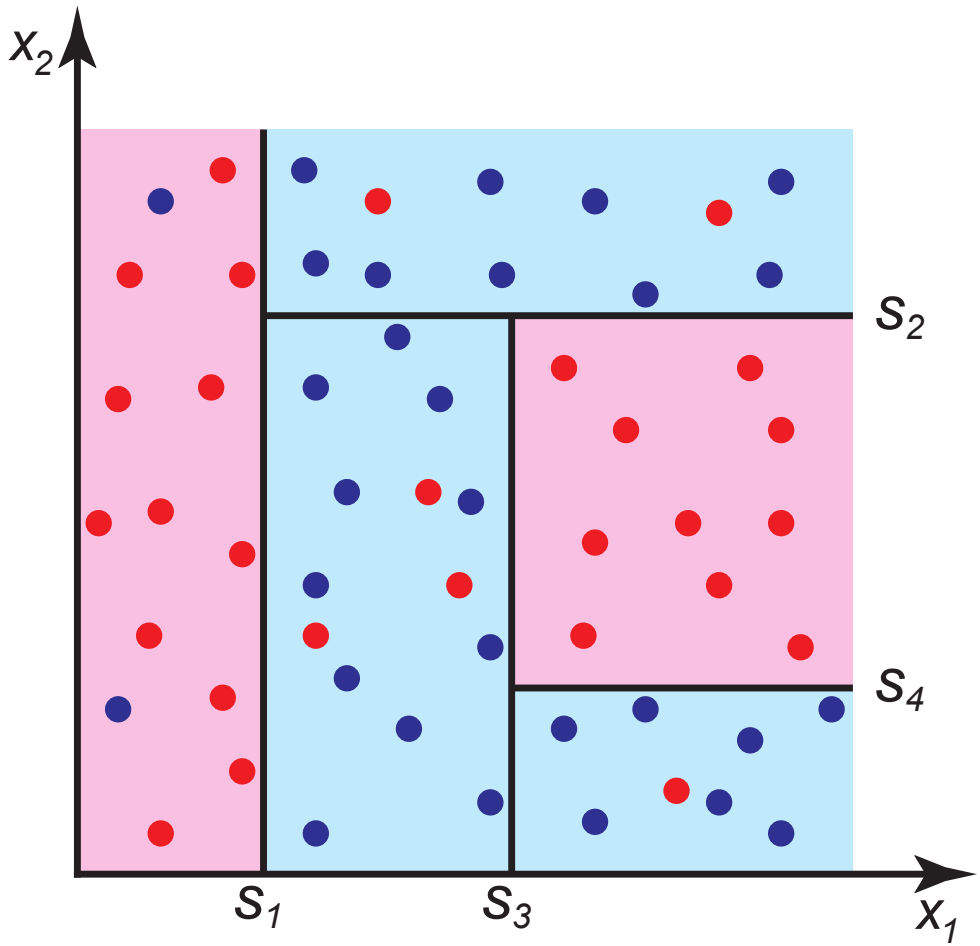
- Larger values of λ correspond to smaller values of B, and *vice versa*.

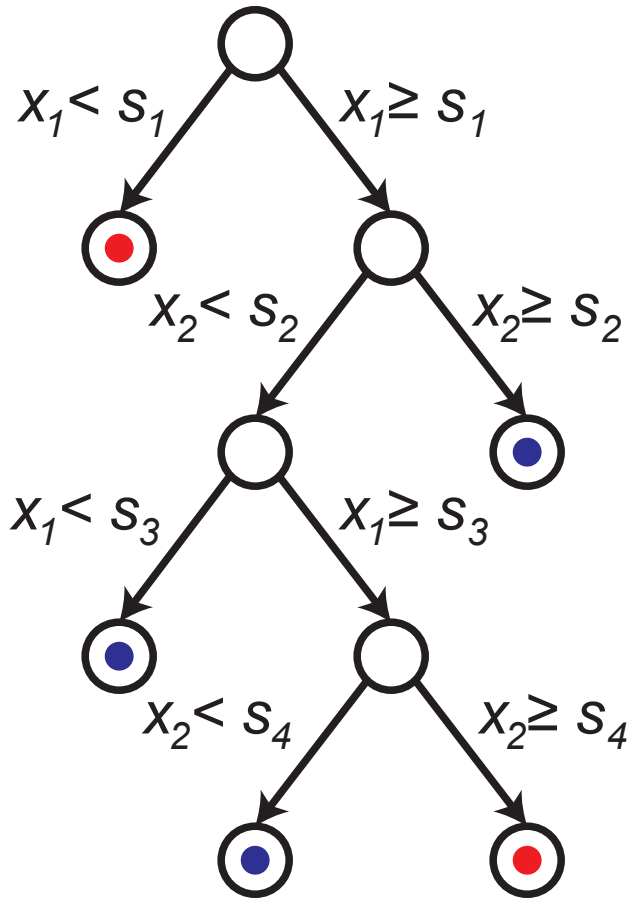
Contents

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:
- Suppose that **a partition** into M regions $R_1, R_2, ..., R_M$ **is given**.
- The **response is** then modeled as **a constant** c_m **in each region**:

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

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

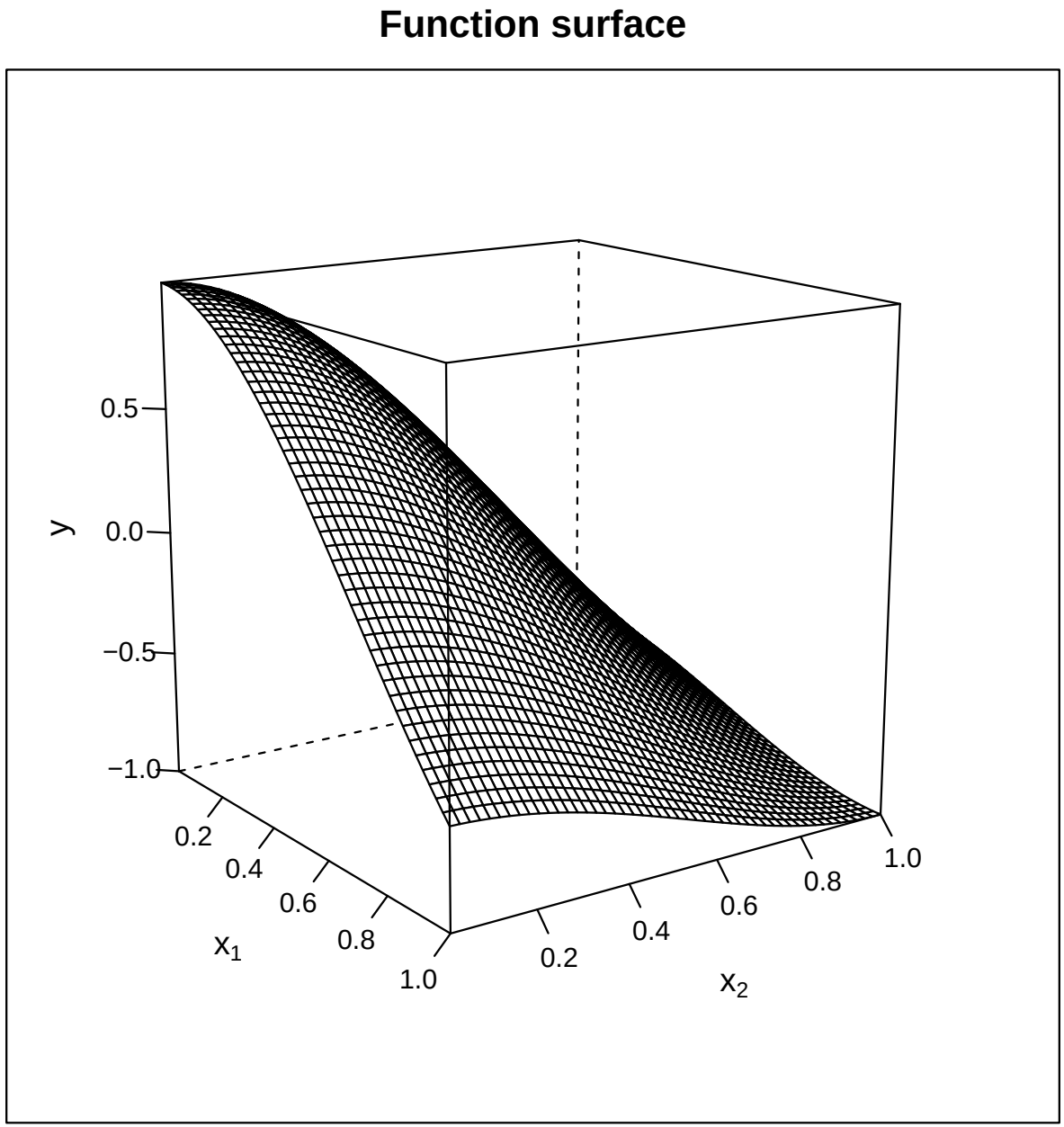
$$Q^{(m)}(T) = \frac{n^{(m_L)}}{n^{(m)}} \sum_{\boldsymbol{x}_i \in R_{m_L}} (y_i - \hat{c}_{m_L})^2 + \frac{n^{(m_R)}}{n^{(m)}} \sum_{\boldsymbol{x}_i \in R_{m_R}} (y_i - \hat{c}_{m_R})^2 \,.$$

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

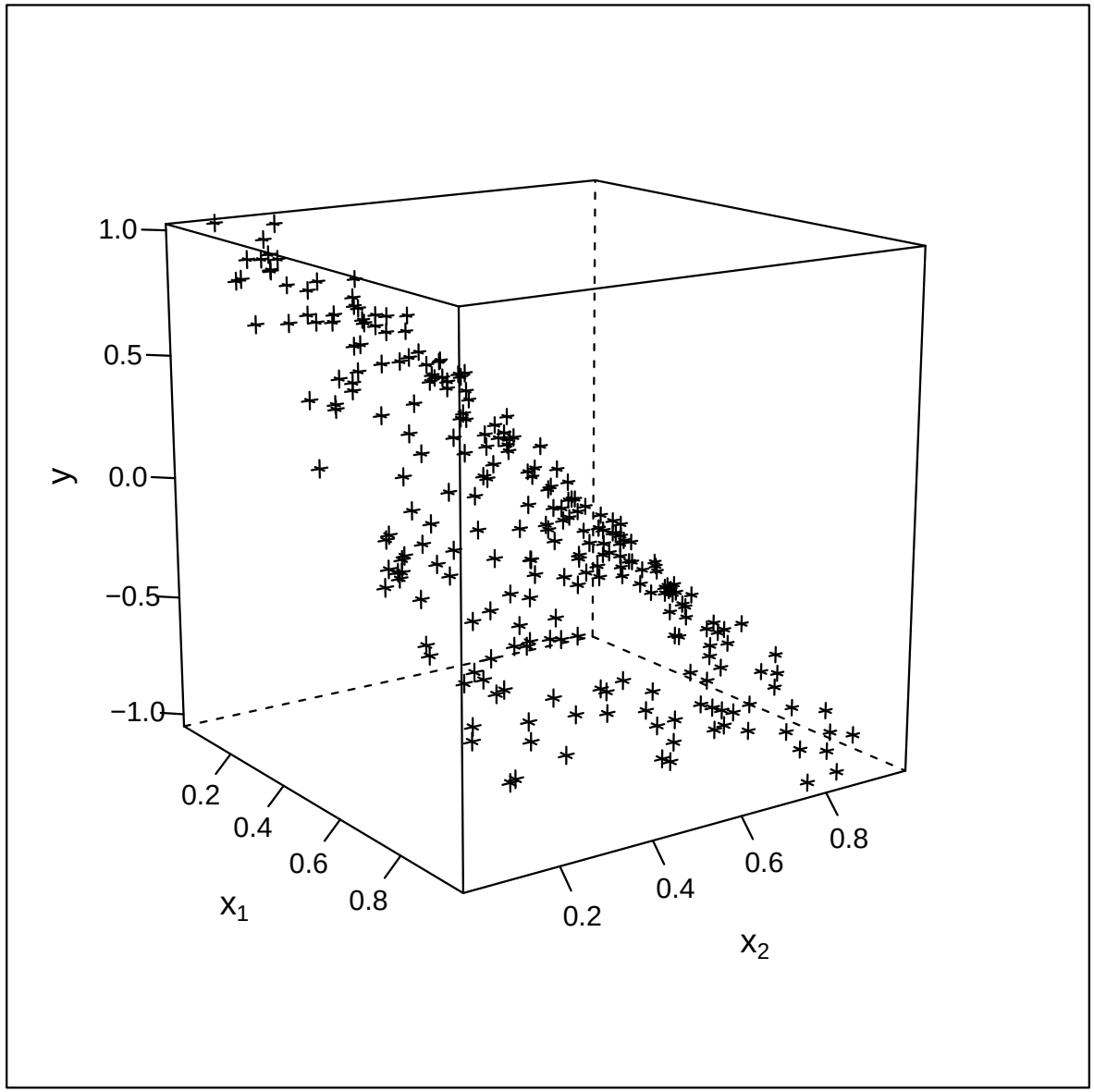
$$\hat{c}_m = \frac{1}{n^{(m)}} \sum_{\boldsymbol{x}_i \in R_m} y_i \,,$$

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

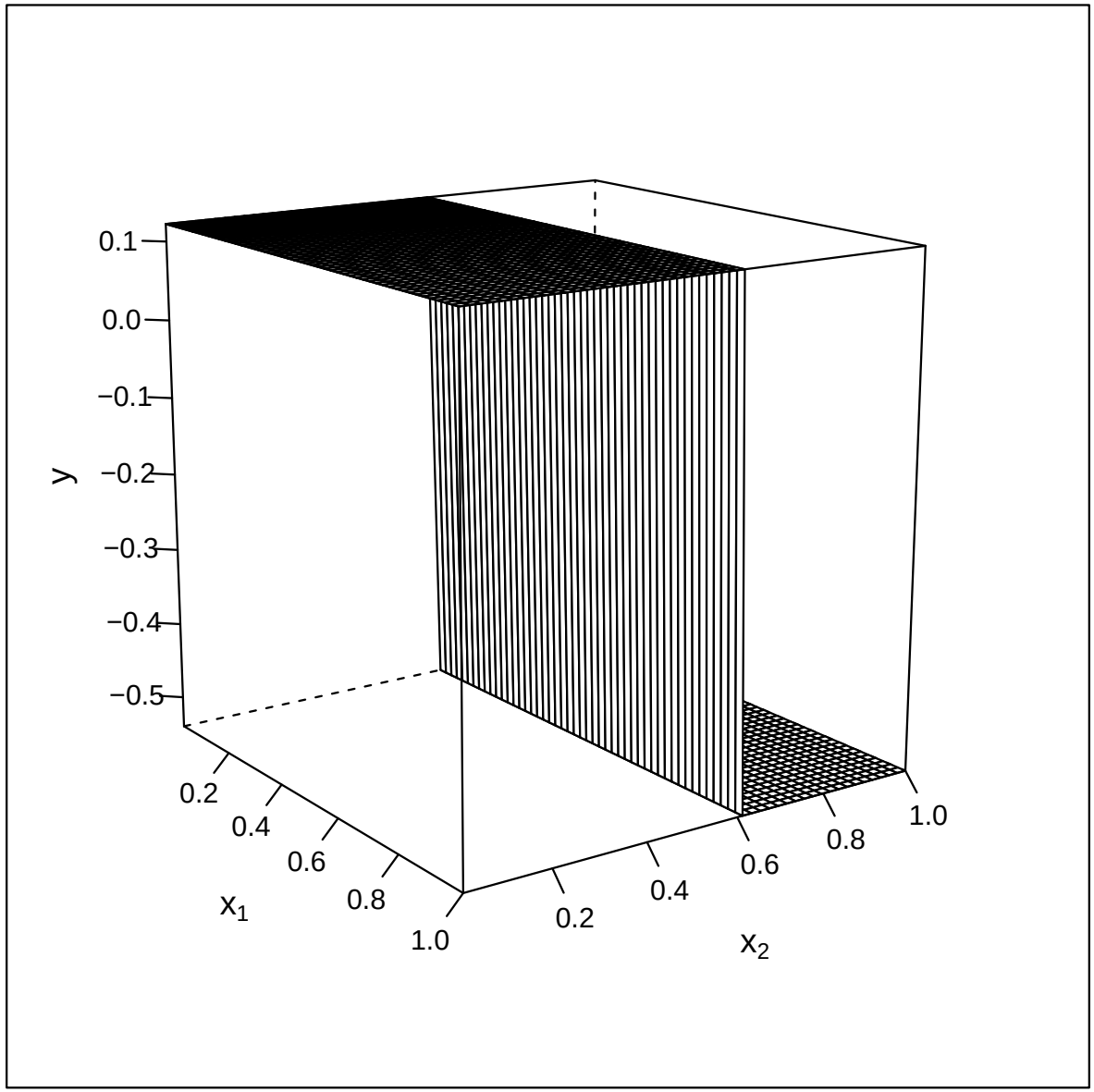
Regression tree (illustration)



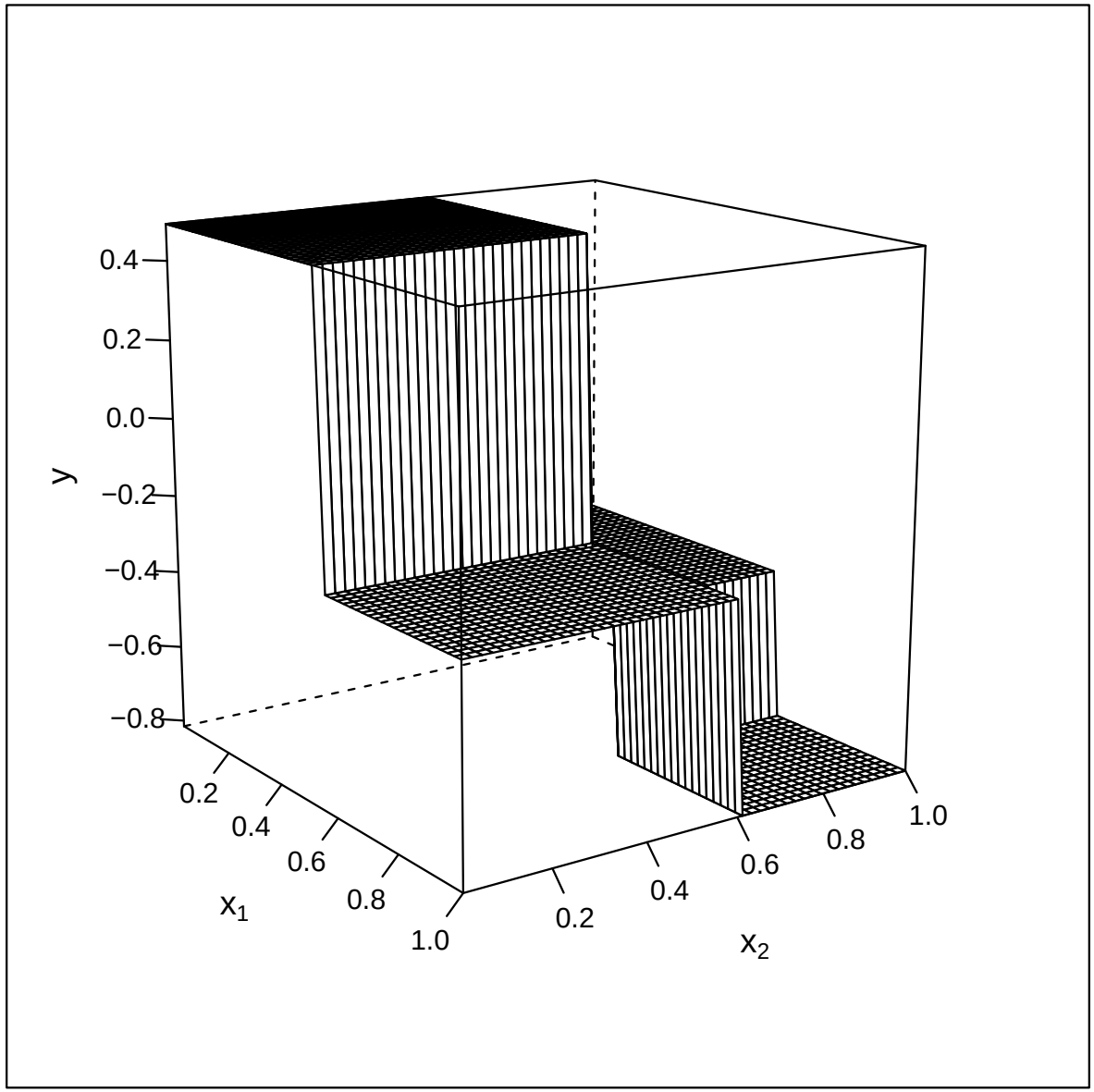
Regression tree (illustration)



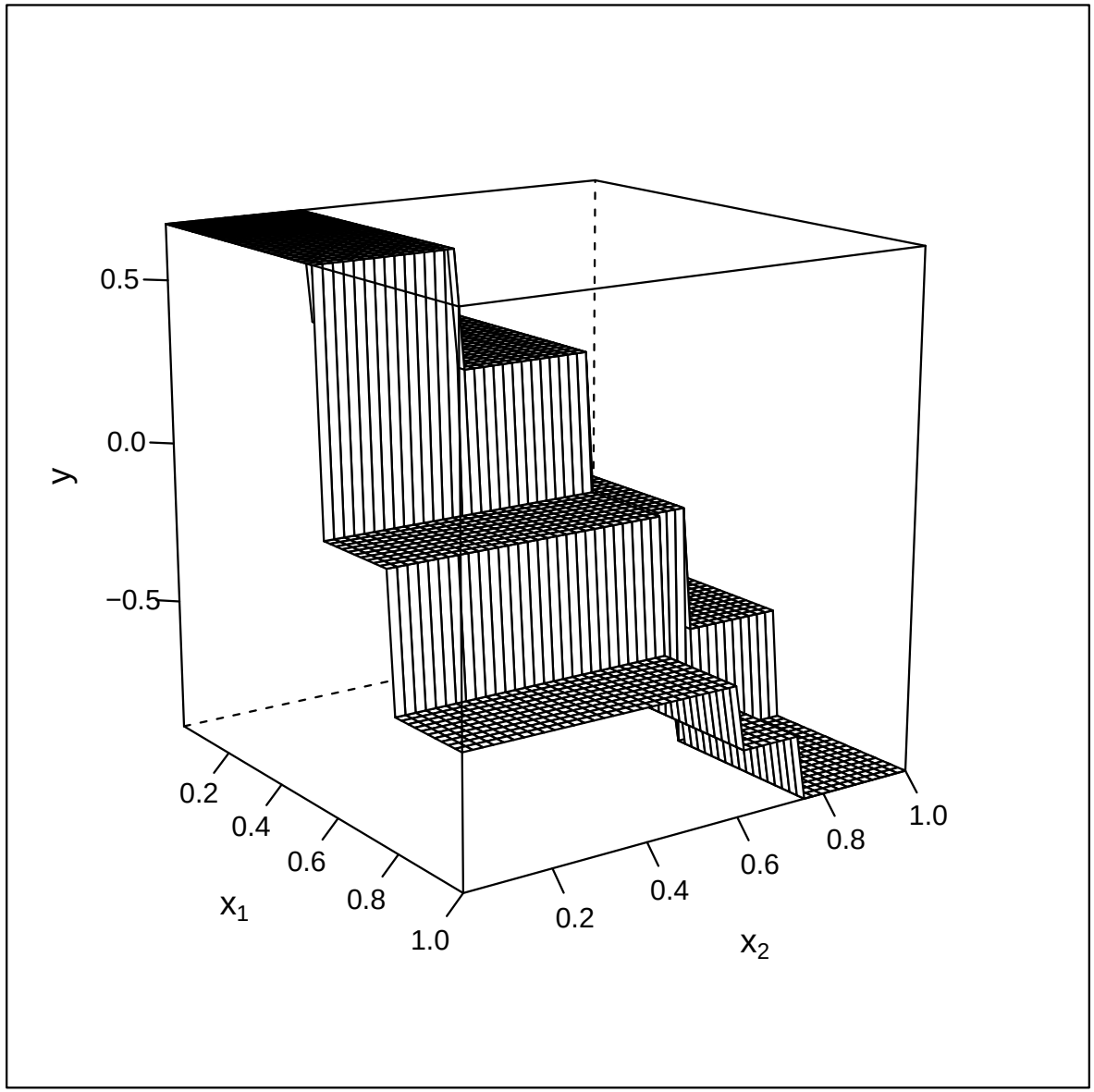
Regression tree (illustration)



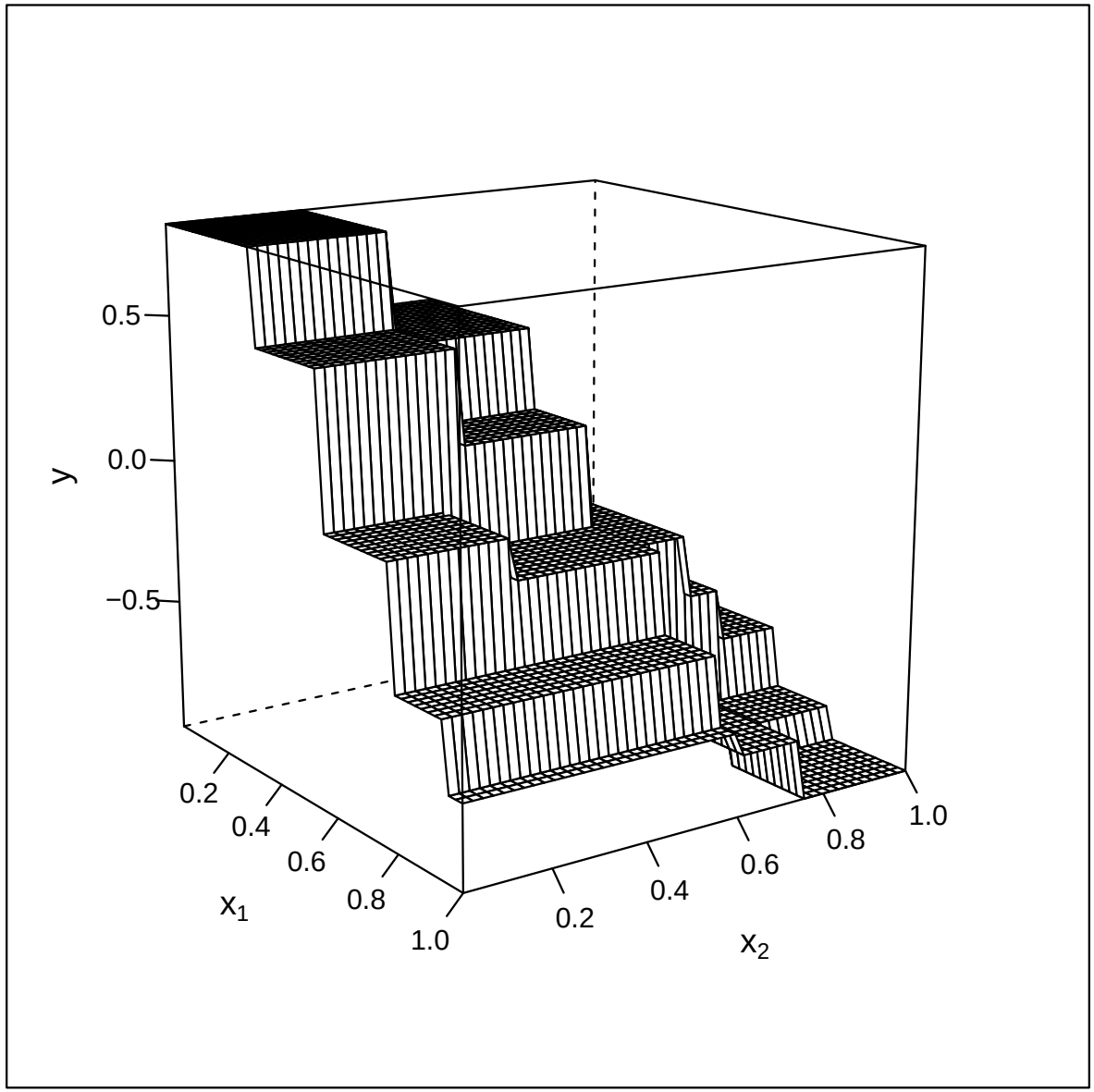
Regression tree (illustration)



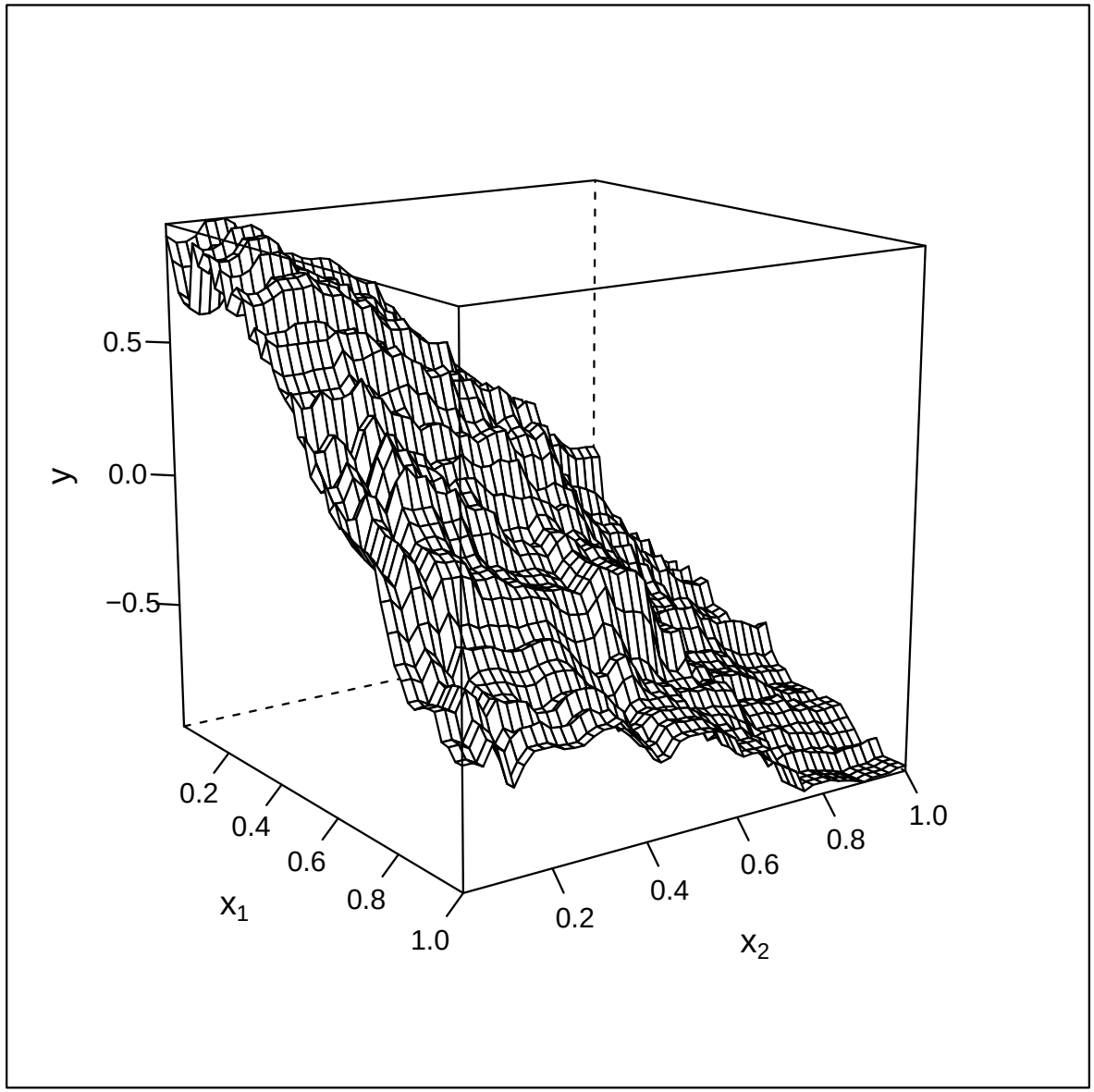
Regression tree (illustration)



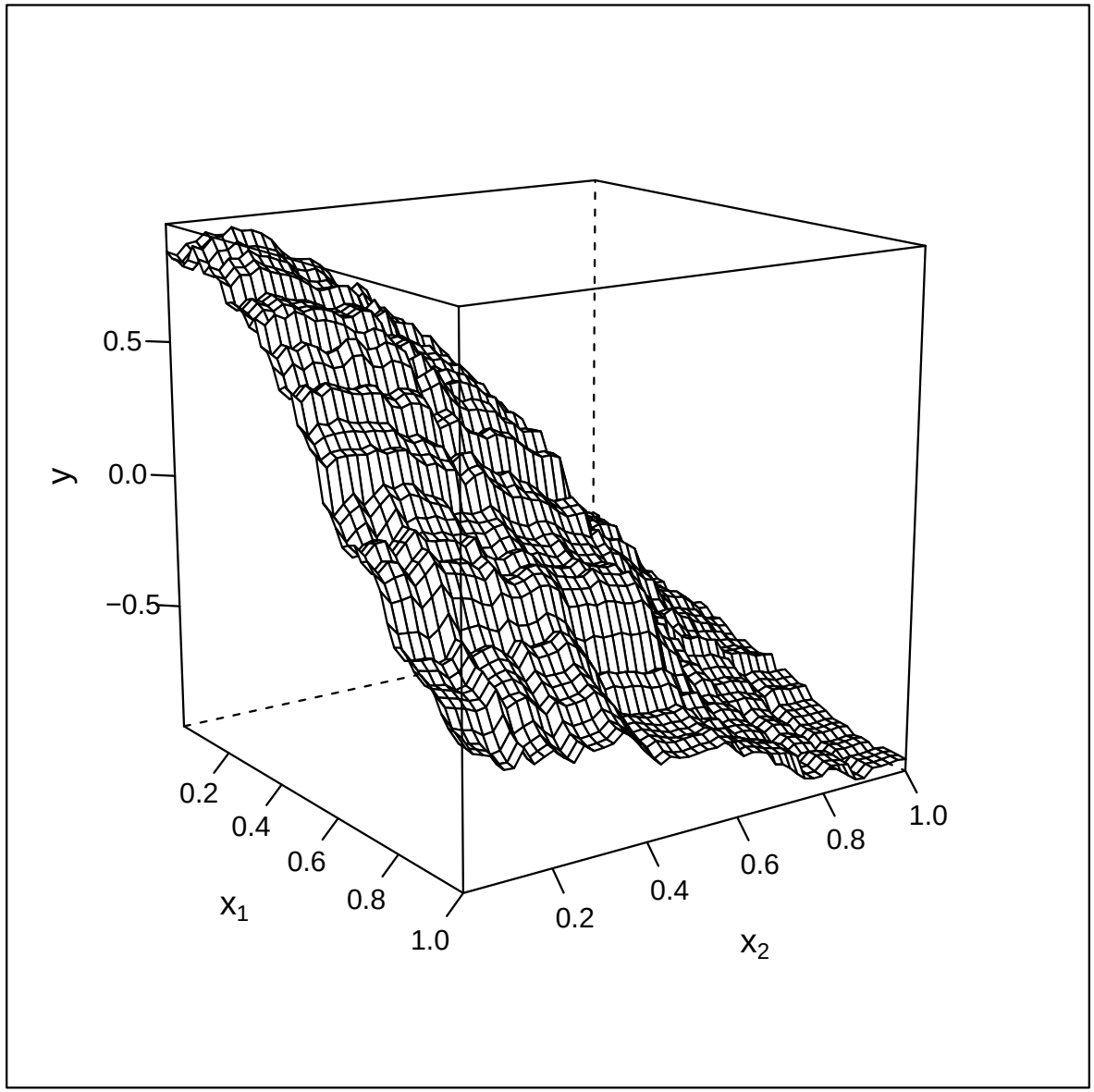
Regression tree (illustration)



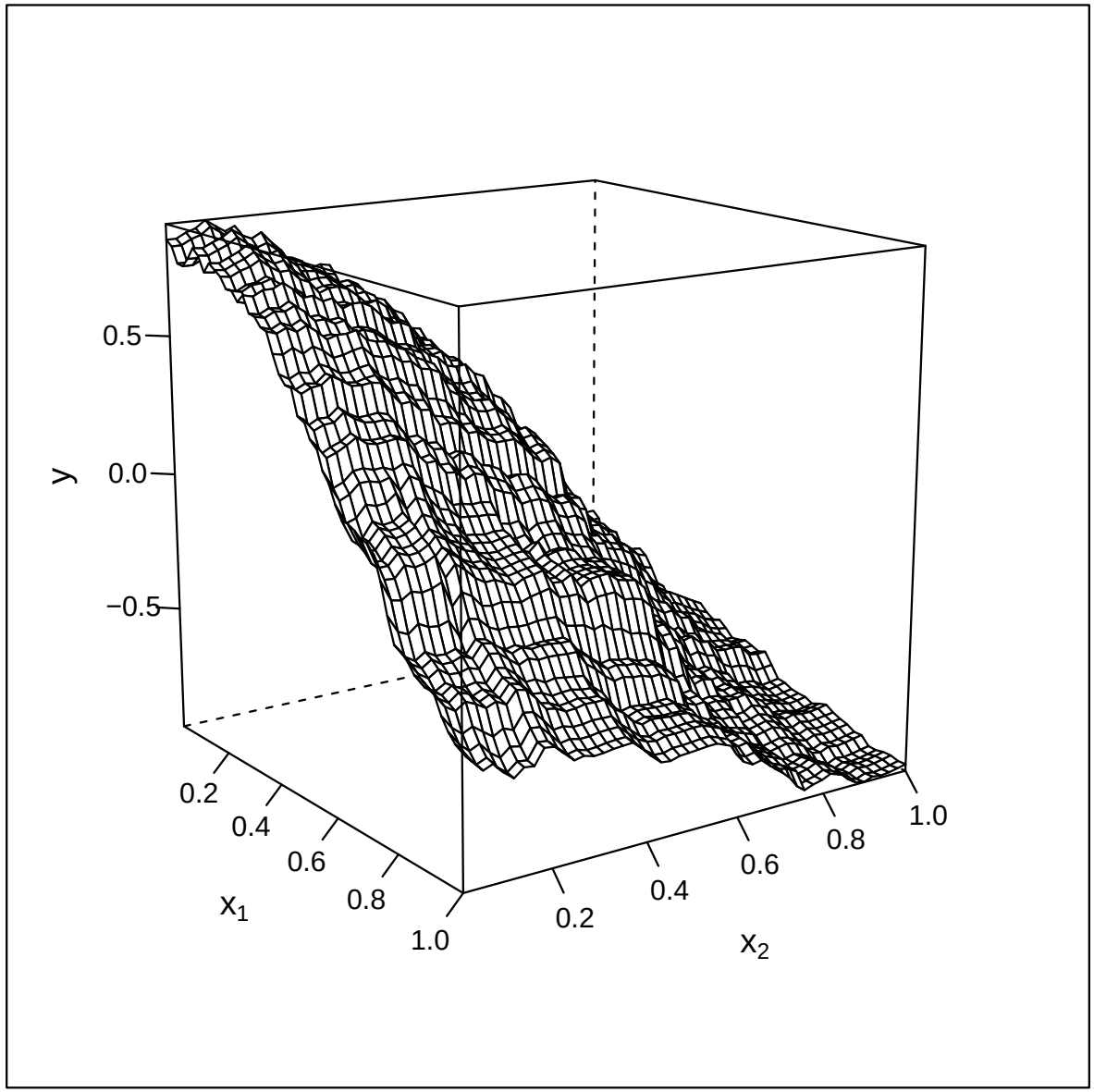
Regression tree (illustration)



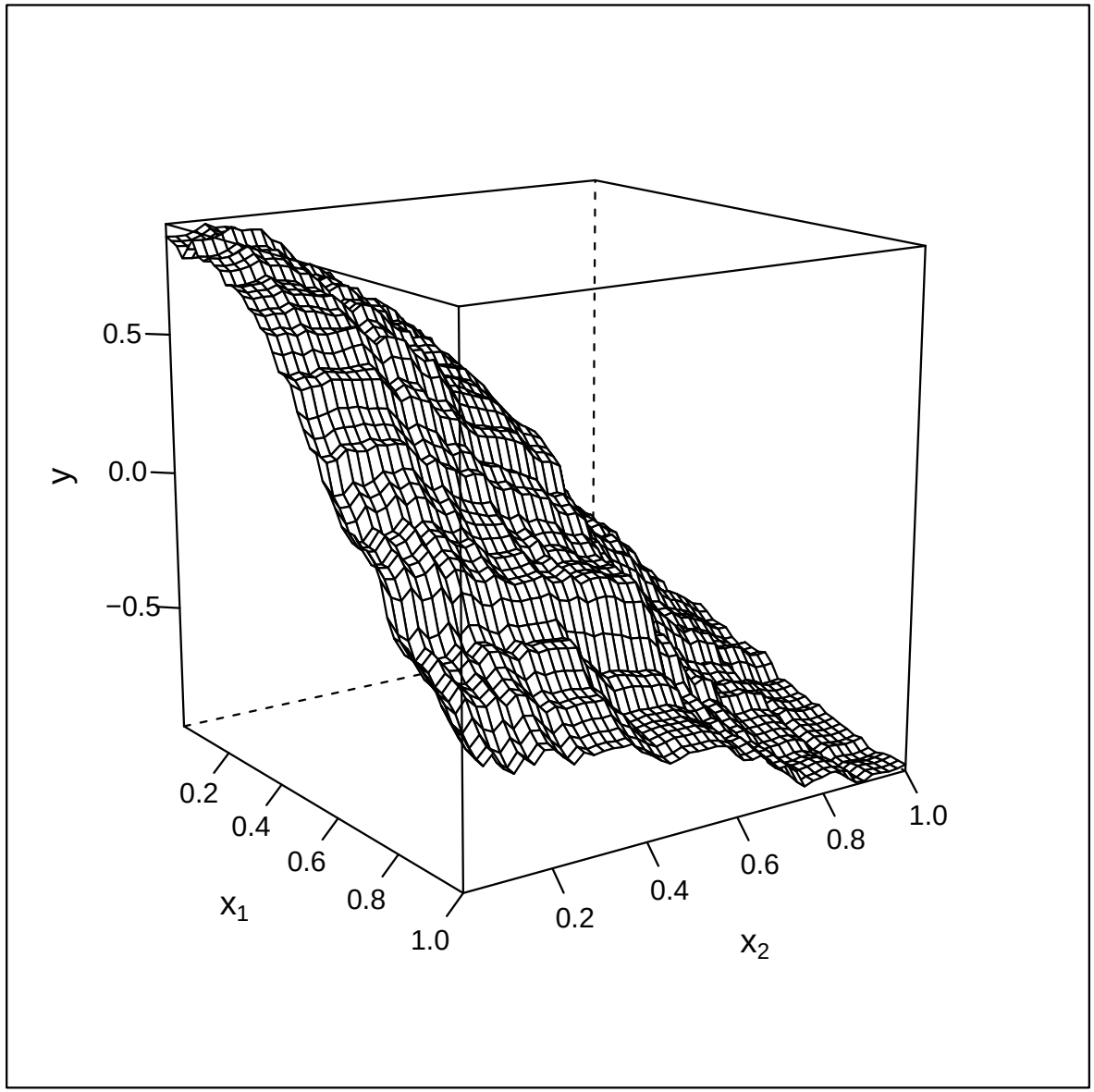
Regression tree (illustration)



Regression tree (illustration)



Regression tree (illustration)



Boosting regression trees in R

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

Input parameters (arguments):

- loss function ℓ (`distribution`),
- number of iterations B (`n.trees`),
- depth of the grown three K (`interaction.depth`),
- regularization parameter λ (`shrinkage`).

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

Constructing a classifier (training)

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

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

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

   - 2.2 Grow the regression tree of maximal depth K based on the sample

$$(\boldsymbol{x}_1, v_1),\ (\boldsymbol{x}_2, v_2),\ ...,\ (\boldsymbol{x}_n, v_n)\,.$$

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

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

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

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

Contents

Connection to AdaBoost

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

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

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

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

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

- Hence:

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

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

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

LogitBoost

- Suppose (here) that Y takes values from $\{0, 1\}$.
- The conditional random variable $Y|X = \boldsymbol{x}$ follows the Bernoulli distribution with parameter $p(\boldsymbol{x}) = \mathbb{P}[Y = 1|X = \boldsymbol{x}]$, and its **likelihood as a function of** $p(\boldsymbol{x})$ for an observation $(\boldsymbol{x}, y)$ can be written as:
$$p(\boldsymbol{x})^y \left(1 - p(\boldsymbol{x})\right)^{1-y}.$$
- The **logistic regression model** assumes:
$$p(\boldsymbol{x}) = \frac{1}{1 + e^{-(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x})}} = \frac{e^{\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}}}{1 + e^{\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}}},$$
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 $(1, \boldsymbol{x}^T)^T$ in $p(\boldsymbol{x})$.

LogitBoost

- One can rewrite:

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

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

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

- **Choice of the loss function:** the negative log-likelihood (which will be minimized):

$$-\Big(y \log p(\boldsymbol{x}) + (1-y) \log\big(1 - p(\boldsymbol{x})\big)\Big) = \log(1 + e^{-2\tilde{y} f(\boldsymbol{x})})$$

  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:

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

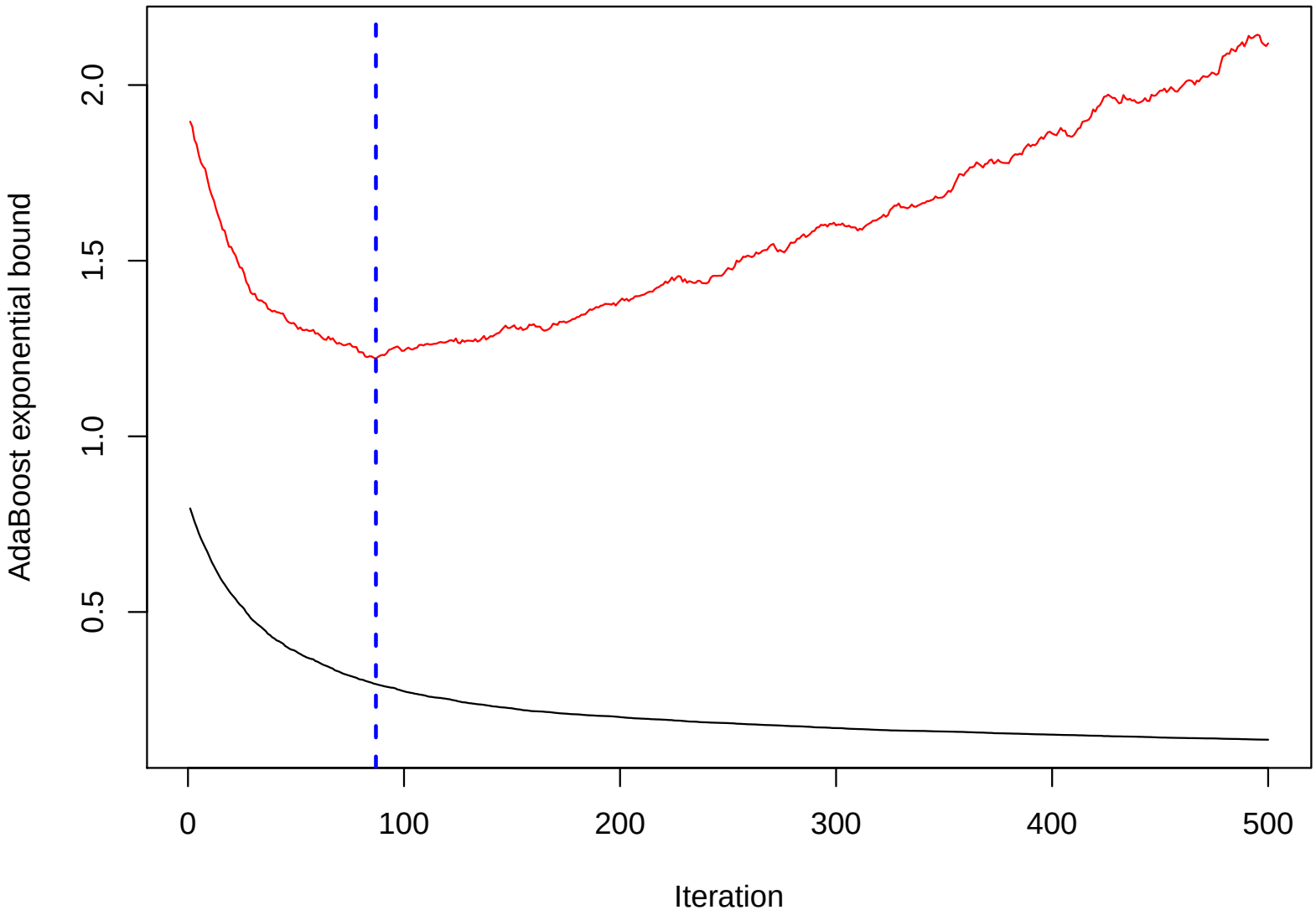
  is called **LogitBoost**.

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

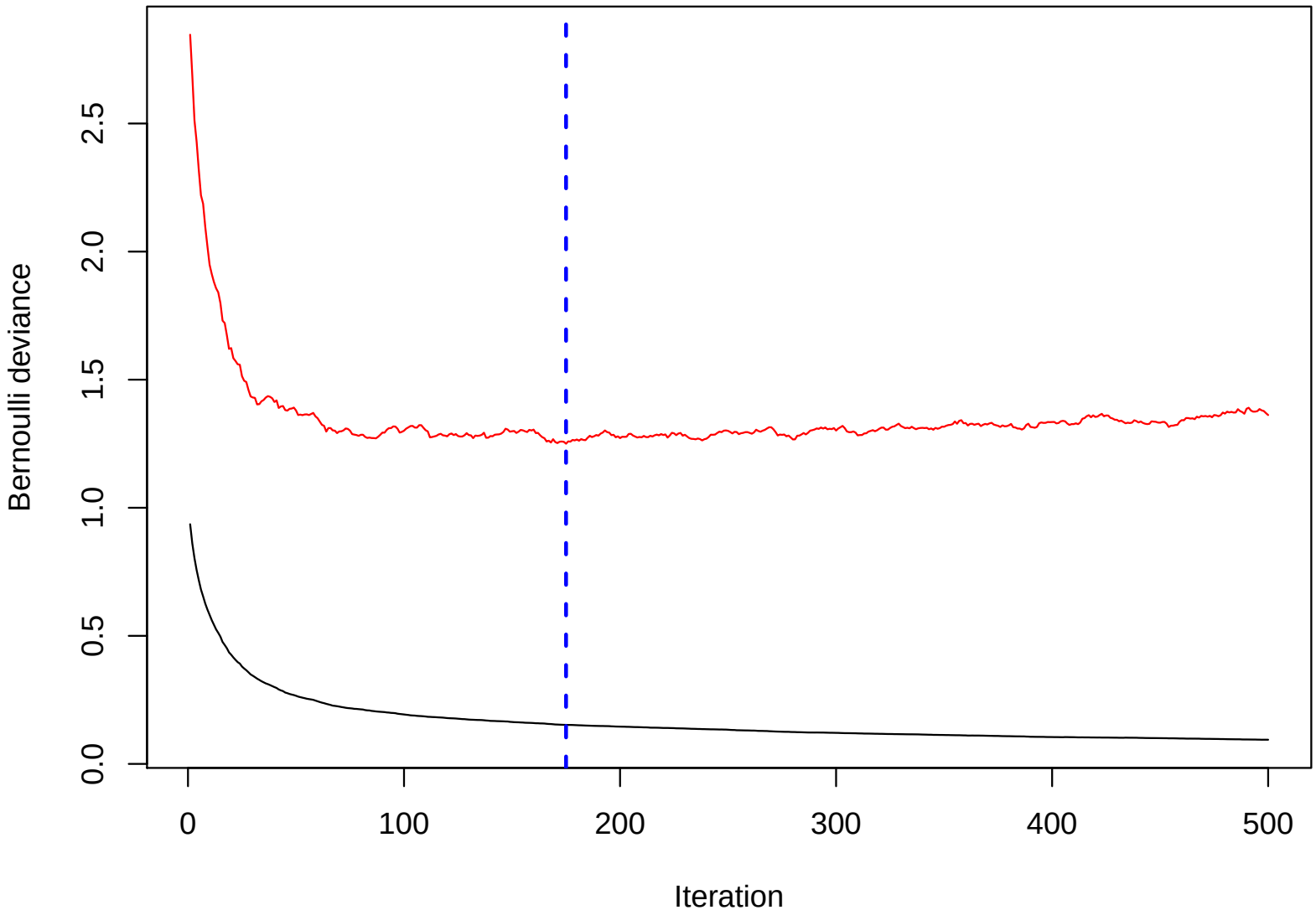
$$f^*(\boldsymbol{x}) = \frac{1}{2} \log \frac{\mathbb{P}[Y = 1 | X = \boldsymbol{x}]}{\mathbb{P}[Y = -1 | X = \boldsymbol{x}]} .$$

- 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 funcion 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.