STK-IN4300
Statistical Learning Methods in Data Science

Riccardo De Bin

debin@math.uio.no

Outline of the lecture

- Gradient Boosting
  - review
  - L_2 boosting with linear learner
- Tree-based boosting
- eXtreme Gradient Boosting

Gradient Boosting: from the last lecture

In the last lecture:

- boosting as implementation of "wisdom of the crowds";
- repeatedly apply a weak learner to modification of the data;
- from AdaBoost to gradient boosting;
- forward stagewise additive modelling;
- importance of shrinkage.

Gradient Boosting: general gradient boosting

Gradient boosting algorithm:

1. initialize the estimate, e.g., $f_0(x) = 0$;
2. for $m = 1, \ldots, m_{\text{stop}}$,
   - 2.1 compute the negative gradient vector,
     $$u_m = - \left. \frac{\partial L(y, f(x))}{\partial f(x)} \right|_{f(x)=\hat{f}_{m-1}(x)};$$
   - 2.2 fit the base learner to the negative gradient vector, $h_m(u_m, x)$;
   - 2.3 update the estimate, $f_m(x) = f_{m-1}(x) + \nu h_m(u_m, x)$.
3. final estimate, $\hat{f}_{m_{\text{stop}}}(x) = \sum_{m=1}^{m_{\text{stop}}} \nu h_m(u_m, x)$

Note:

- X must be centered;
- $\hat{f}_{m_{\text{stop}}}(x)$ is a GAM.

L_2 boosting with linear learner: algorithm

L_2Boost algorithm:

1. initialize the estimate with least squares, $\hat{f}_0(x) = h(x, \hat{\theta}_{y,X})$,
   - $\hat{\theta}_{y,X} = \text{argmin}_\theta \sum_{i=1}^N (Y_i - h(x, \theta))^2$;
2. for $m = 1, \ldots, m_{\text{stop}}$,
   - 2.1 compute the residuals, $u_m = y_i - \hat{f}_{m-1}(x)$;
   - 2.2 fit the base learner to the residuals by, e.g., least squares;
   - 2.3 update the estimate, $f_m(x) = f_{m-1}(x) + \nu h_m(u_m, x)$.
3. final estimate, $\hat{f}_{m_{\text{stop}}}(x) = \sum_{m=1}^{m_{\text{stop}}} h_m(u_m, x)$

L_2 boosting with linear learner: boosting operator

Consider a linear regression example,

$$y_i = f(x_i) + \epsilon_i, i = 1, \ldots, N,$$

in which:

- ϵ_i i.i.d. with $E[\epsilon_i] = 0$, $\text{Var}[\epsilon_i] = \sigma^2$;
- we use a linear base learner $\mathcal{S} : \mathbb{R}^N \to \mathbb{R}^N$ ($\mathcal{S}y = \hat{y}$);
  - e.g., $\mathcal{S} = \nu X(X^T X)^{-1} X^T$.

Note that, using an L_2 loss function,

- $\hat{f}_m(x) = \hat{f}_{m-1}(x) + \mathcal{S}u_m$;
- $u_m = y - \hat{f}_{m-1}(x) = u_{m-1} - \mathcal{S}u_{m-1} = (I - \mathcal{S})u_{m-1}$;
- iterating, $u_m = (I - \mathcal{S})^m$, $m = 1, \ldots, m_{\text{stop}}$.

Because $\hat{f}_0(x) = \mathcal{S}y$, then $\hat{f}_{m_{\text{stop}}}(x) = \sum_{m=0}^{m_{\text{stop}}} \mathcal{S}(I - \mathcal{S})^m y$, i.e.,

$$\hat{f}_{m_{\text{stop}}}(x) = \underbrace{(I - (I - \mathcal{S})^{m+1})}_{\text{boosting operator } \mathcal{B}_m} y.$$

L_2 boosting with linear learner: properties

Consider a linear learner $\mathcal{S}$ (e.g., least square) with eigenvalues λ_k, $k = 1, \ldots, N$. Then

Proposition 2 (Bühlmann & Yu, 2003): The eigenvalues of the L_2Boost operator $\mathcal{B}_m$ are

$$\left\{1 - (1 - \lambda_k)^{m_{\text{stop}}+1}, k = 1, \ldots, N\right\}.$$

If $\mathcal{S} = \mathcal{S}^T$ (i.e., symmetric), then $\mathcal{B}_m$ can be diagonalized with an orthonormal transformation,

$$\mathcal{B}_m = U D_m U^T, \qquad D_m = \text{diag}(1 - (1 - \lambda_k)^{m_{\text{stop}}+1})$$

where $UU^T = U^T U = I$.

L_2 boosting with linear learner: properties

We can now compute (Bühlmann & Yu, 2003, Proposition 3):

- $\text{bias}^2(m, \mathcal{S}; f) = N^{-1} \sum_{i=1}^{N} (E[\hat{f}_m(x_i)] - f)^2$
$= N^{-1} f^T U \text{diag}((1 - \lambda_k)^{2m+2}) U^T f;$
- $\text{Var}(m, \mathcal{S}; \sigma^2) = N^{-1} \sum_{i=1}^{N} (\text{Var}[\hat{f}_m(x_i)])$
$= \sigma^2 N^{-1} \sum_{i=1}^{N} (1 - (1 - \lambda_k)^{m+1})^2;$

and

- $\text{MSE}(m, \mathcal{S}; f, \sigma^2) = \text{bias}^2(m, \mathcal{S}; f) + \text{Var}(m, \mathcal{S}; \sigma^2).$

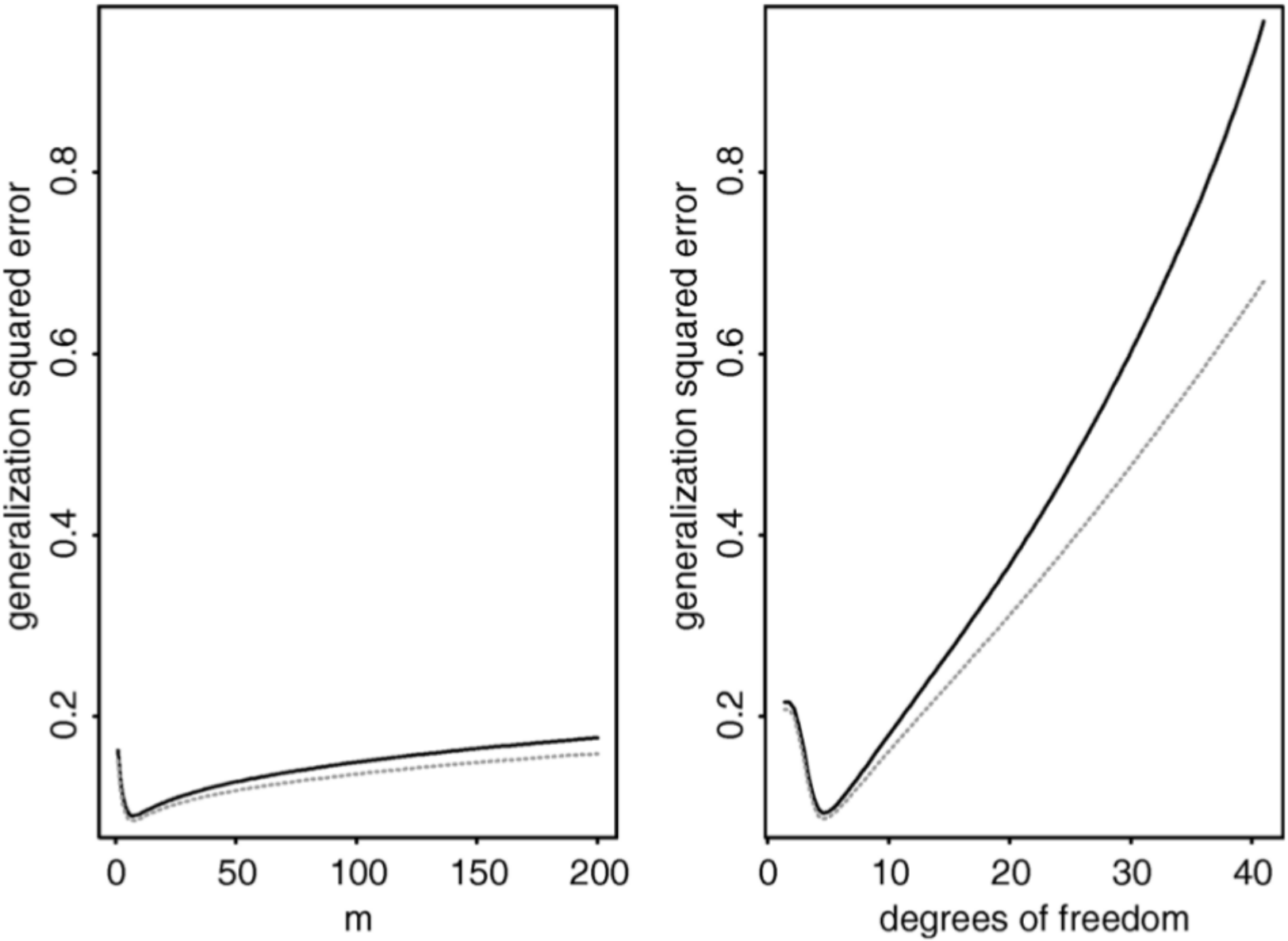
L_2 boosting with linear learner: properties

Assuming $0 < \lambda_k \leqslant 1$, $k = 1, \ldots, N$, note that:

- $\text{bias}^2(m, \mathcal{S}; f)$ decays exponentially fast for m increasing;
- $\text{Var}(m, \mathcal{S}; \sigma^2)$ increases exponentially slow for m increasing;
- $\lim_{m \to \infty} \text{MSE}(m, \mathcal{S}; f, \sigma^2) = \sigma^2$;
- if $\exists k : \lambda_k < 1$ (i.e., $\mathcal{S} \neq I$), then $\exists m : \text{MSE}(m, \mathcal{S}; f, \sigma^2) < \sigma^2$;
- if $\forall k : \lambda_k < 1, \frac{\mu_k^2}{\sigma^2} > \frac{1}{(1-\lambda_k)^2} - 1$, then $MSE_{\mathcal{B}_m} < MSE_{\mathcal{S}}$, where $\mu = U^T f$ (μ represents f in the coordinate system of the eigenvectors of $\mathcal{S}$).

(for the proof, see Bühlmann & Yu, 2003, Theorem 1)

L_2 boosting with linear learner: properties



L_2 boosting with linear learner: properties

About $\frac{\mu_k^2}{\sigma^2} > \frac{1}{(1-\lambda_k)^2} - 1$:

- a large left side means that f is relatively complex compared with the noise level σ^2;
- a small right side means that λ_k is small, i.e. the learner shrinks strongly in the direction of the k-th eigenvector;
- therefore, to have boosting bringing improvements:
  - there must be a large signal to noise ratio;
  - the value of λ_k must be sufficiently small;

$$\downarrow$$

use a weak learner!!!

L_2 boosting with linear learner: properties

There is a further intersting theorem in Bühlmann & Yu (2003),

Theorem 2: Under the assumption seen till here and $0 < \lambda_k \leqslant 1$, $k = 1, \ldots, N$, and assuming that $E[|\epsilon_1|^p] < \infty$ for $p \in \mathbb{N}$,

$$N^{-1} \sum_{i=1}^{N} E[(\hat{f}_m(x_i) - f(x_i))^p] = E[\epsilon_1^p] + O(e^{-Cm}), \quad m \to \infty$$

where $C > 0$ does not depend on m (but on N and p).

This theorem can be used to argue that boosting for classification is resistant to overfitting (for $m \to \infty$, exponentially small overfitting).

Gradient Boosting: boosting in high-dimensions

The boosting algorithm is working in high-dimension frameworks:

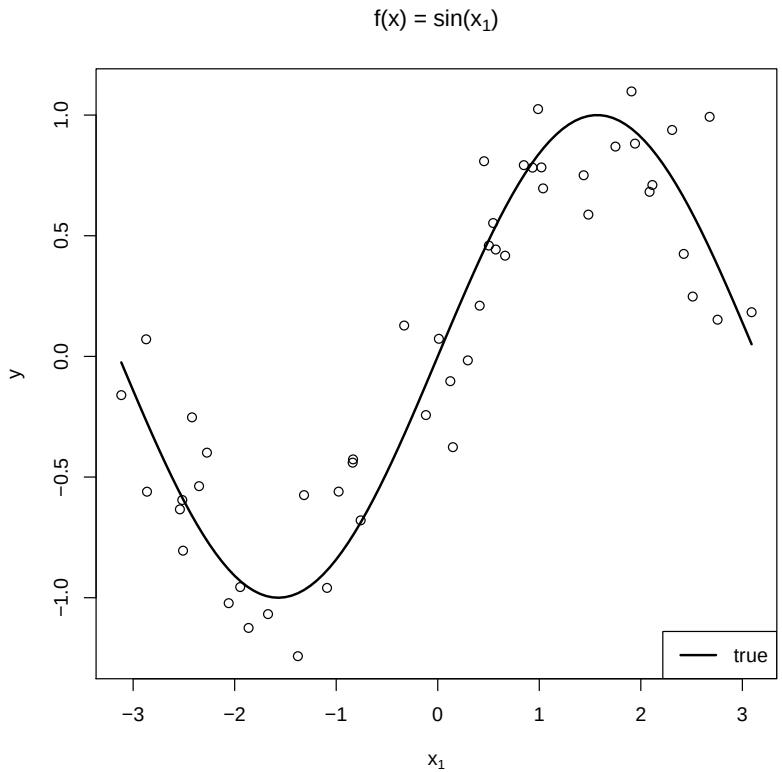
- forward stagewise additive modelling;
- at each step, only one dimension (component) of X is updated at each iteration;
- in a parametric setting, only one $\hat{\beta}_j$ is updated;
- an additional step in which it is decided which component to update must be computed at each iteration.

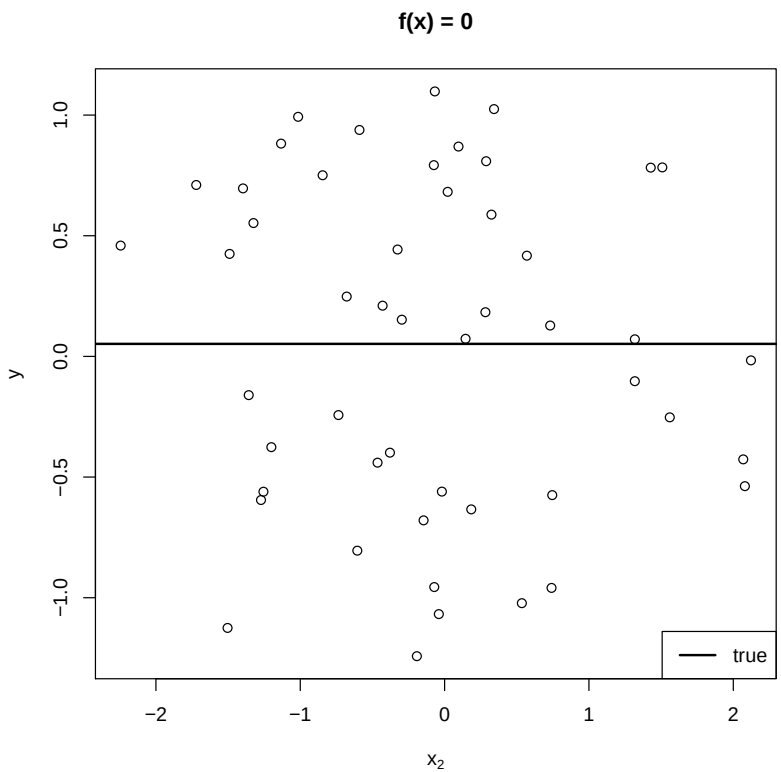
Gradient Boosting: component-wise boosting algorithm

Component-wise boosting algorithm:

1. initialize the estimate, e.g., $\hat{f}_j^{[0]}(x) \equiv 0, j = 1, \ldots, p$;
2. for $m = 1, \ldots, m_{\text{stop}}$,
   - compute the negative gradient vector, $u = -\left.\frac{\partial L(y, f(x))}{\partial f(x)}\right|_{f(x)=\hat{f}^{[m-1]}(x)}$;
   - fit the base learner to the negative gradient vector, $\hat{h}_j(u, x_j)$, for the j-th component only;
   - select the best update j^* (usually that minimizes the loss);
   - update the estimate, $\hat{f}_{j*}^{[m]}(x) = \hat{f}_{j*}^{[m-1]} + \nu \hat{h}_{j*}(u, x_{j*})$;
   - all the other componets do not change.
3. final estimate, $\hat{f}_{m_{\text{stop}}}(x) = \sum_{j=1}^{p} \hat{f}_j^{[m_{\text{stop}}]}(x)$.

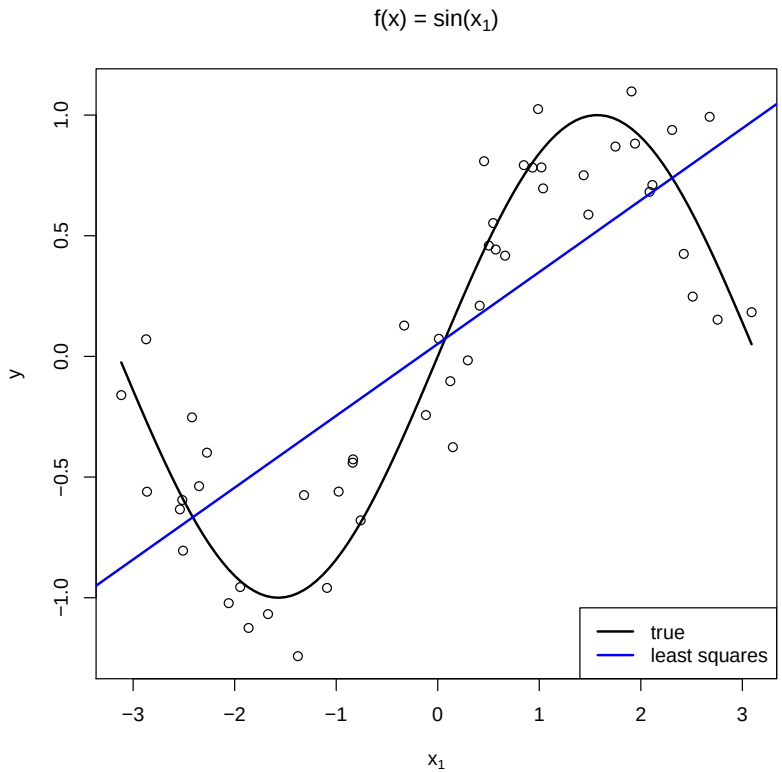
Component-wise Gradient Boosting: algorithm

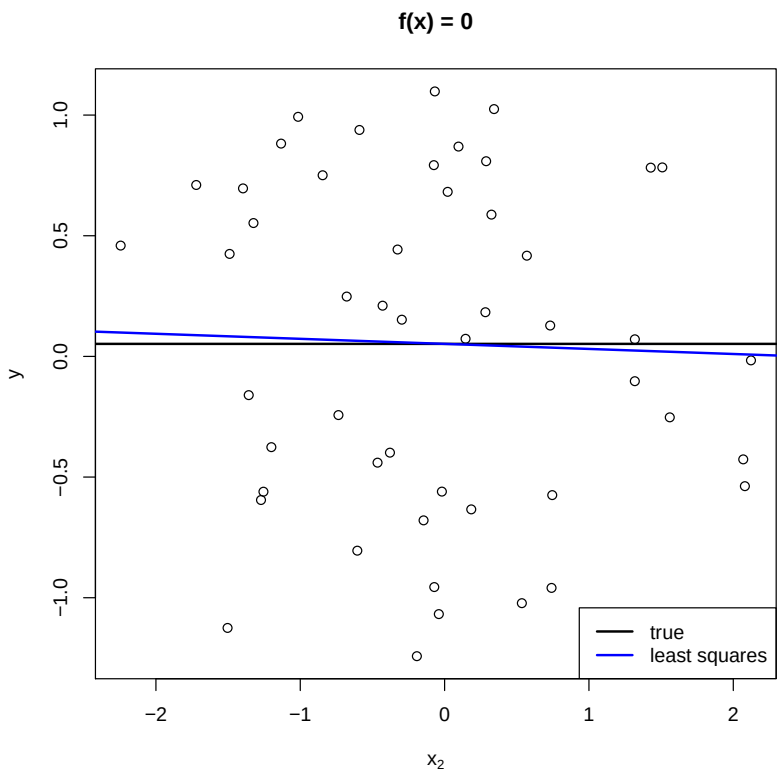




(data generated from $y = \sin(x_1) + 0 \cdot x_2 + \epsilon$, $\epsilon \sim N(0, 0.25^2)$)

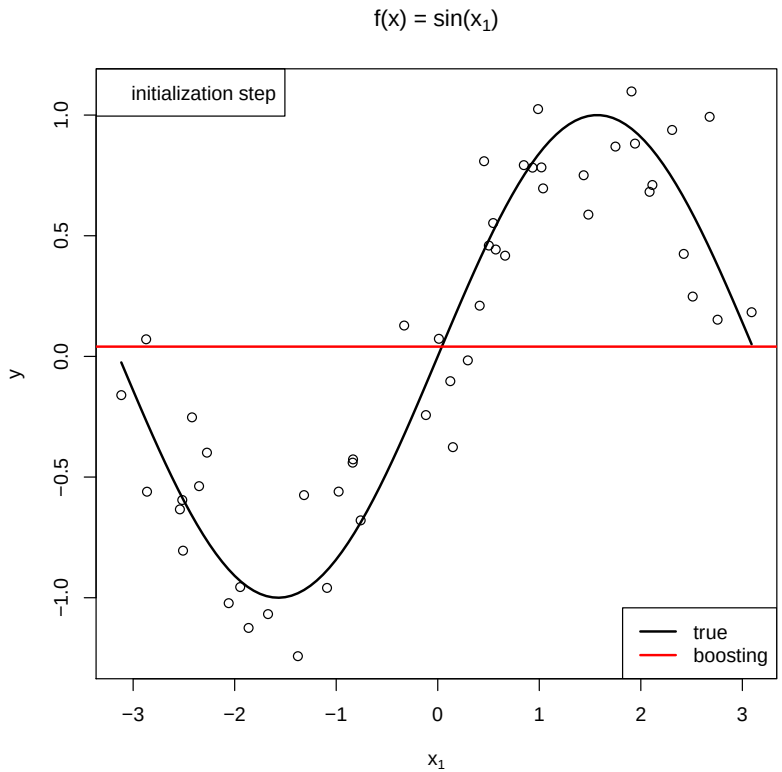
Component-wise Gradient Boosting: algorithm

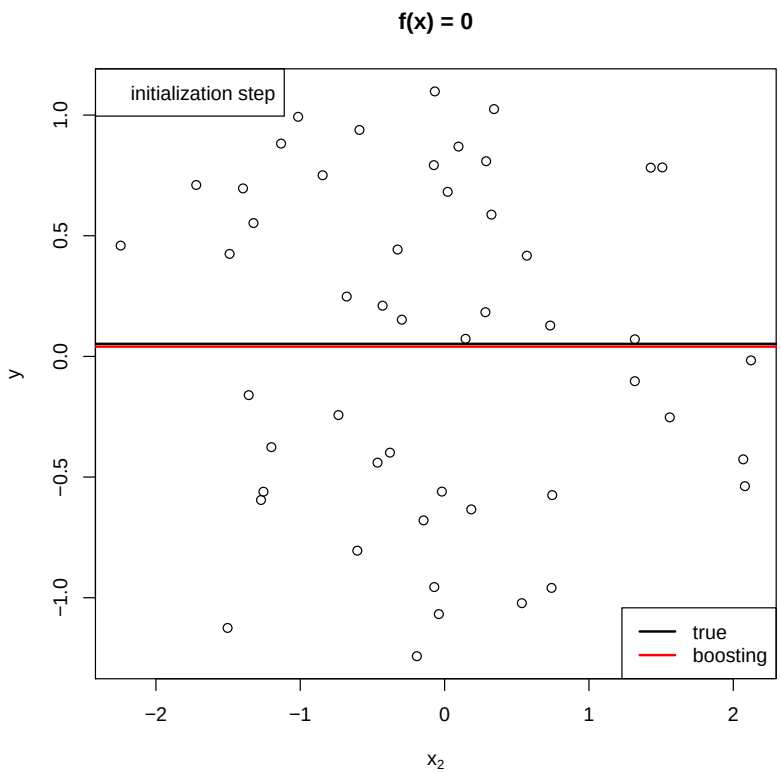




(when fitting a linear model: $\hat{y} = 0.05 + 0.30 \cdot x_1 - 0.02 \cdot x_2$)

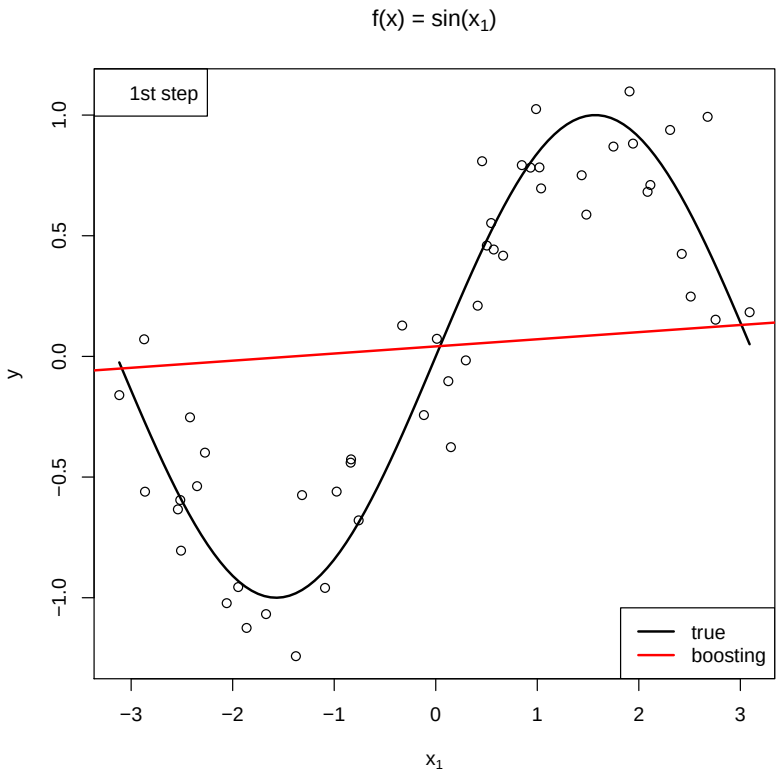
Component-wise Gradient Boosting: algorithm with linear effects

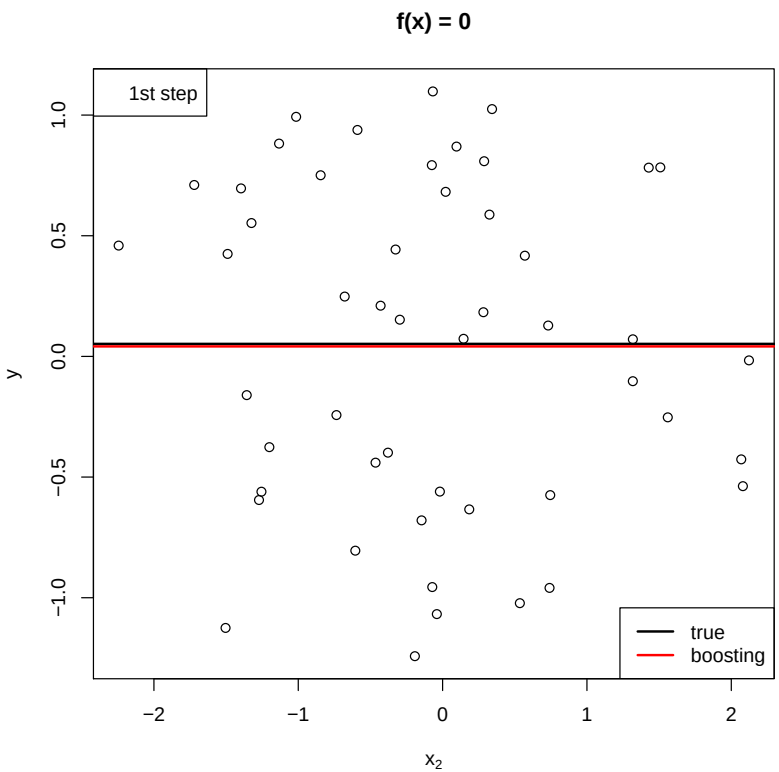




(starting from the null model: $\hat{y} = 0.040$)

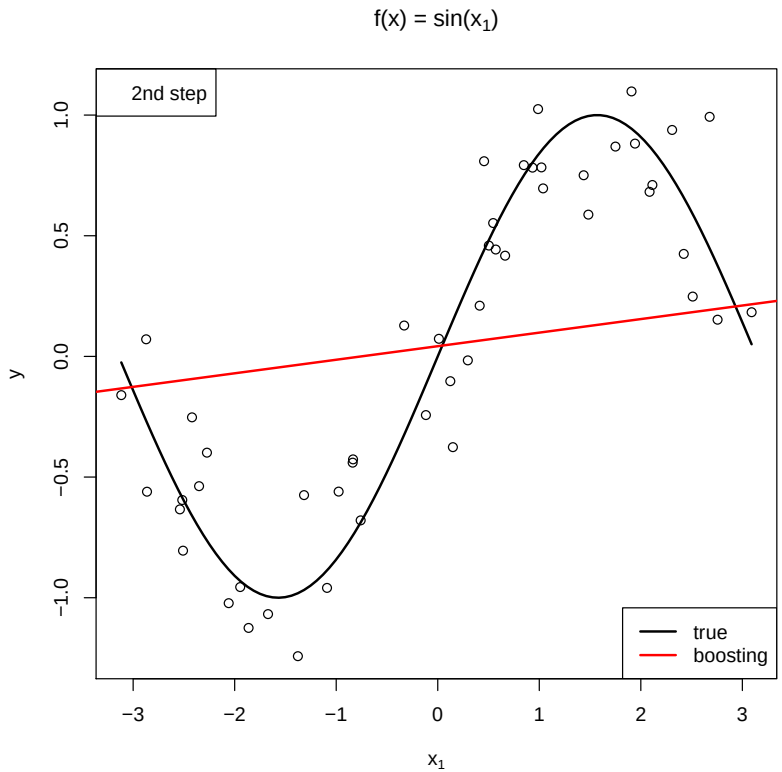
Component-wise Gradient Boosting: algorithm with linear effects

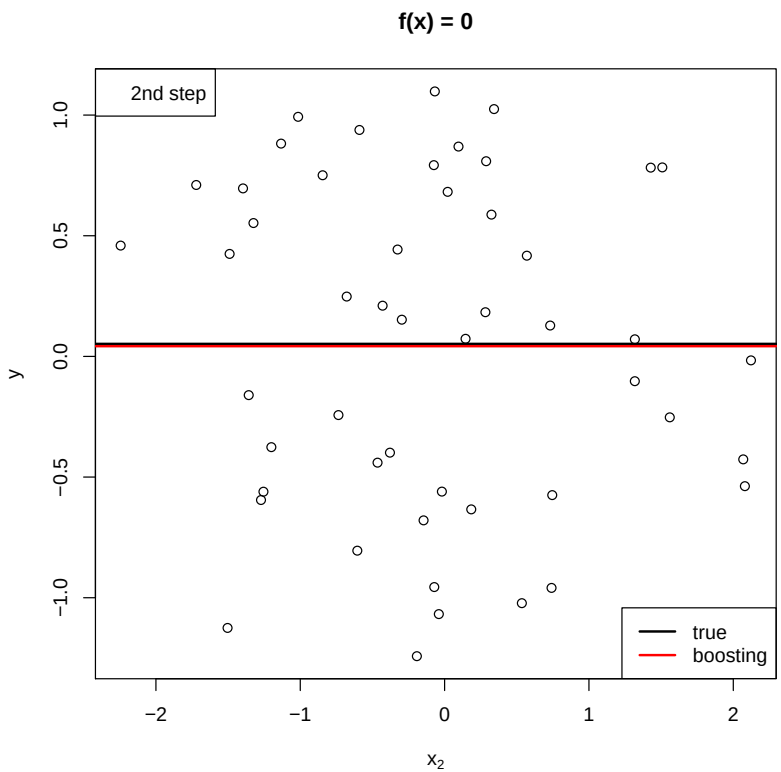




(1st iteration: $\hat{y} = 0.040 + 0.030 \cdot x_1$)

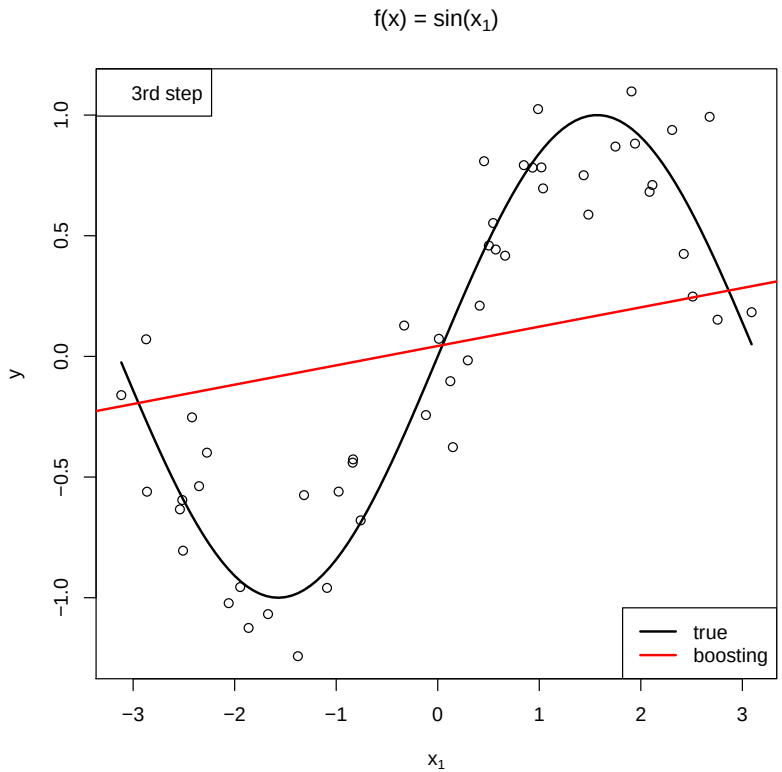
Component-wise Gradient Boosting: algorithm with linear effects

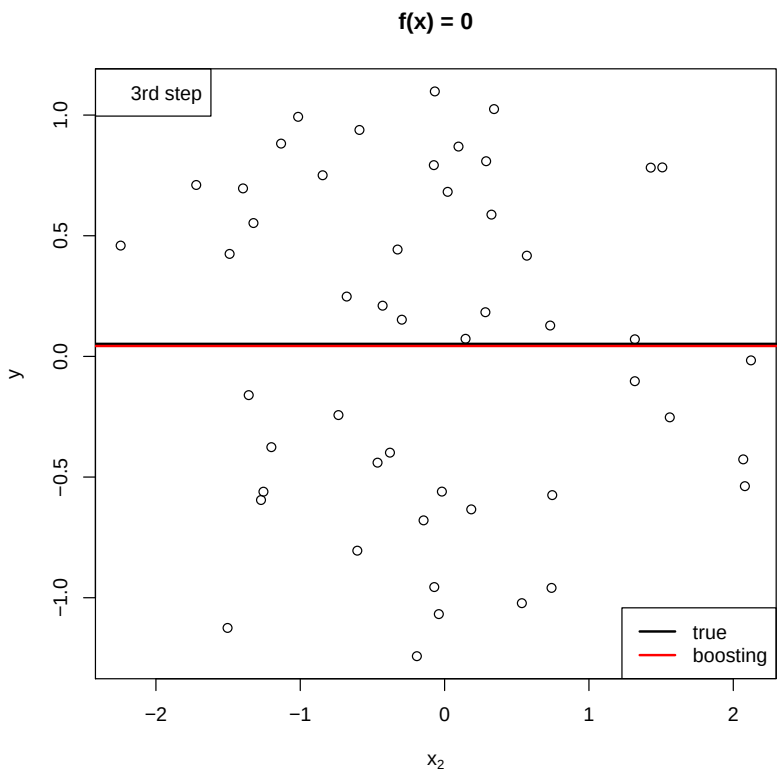




(2nd iteration: $\hat{y} = 0.041 + 0.056 \cdot x_1$)

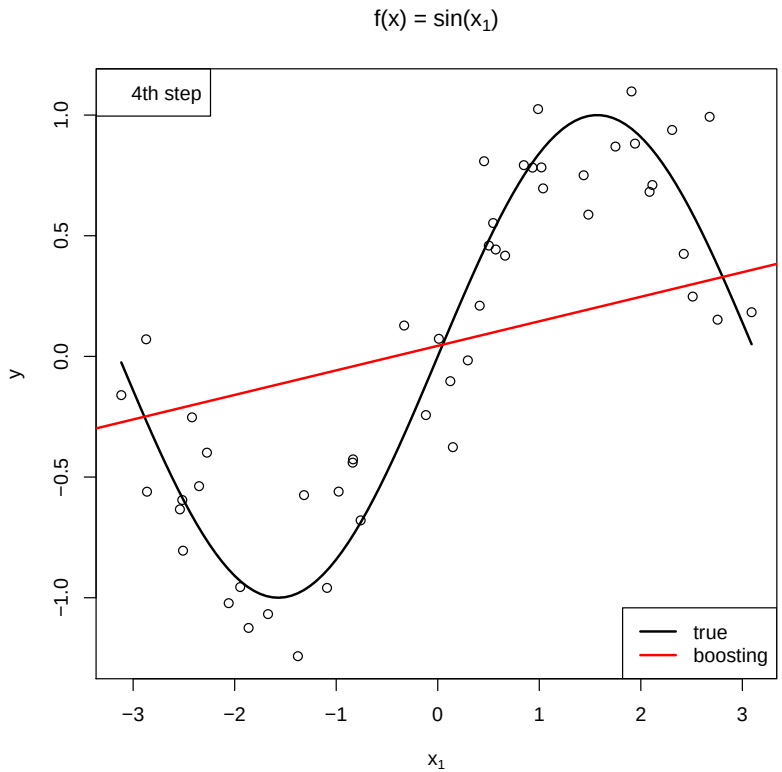
Component-wise Gradient Boosting: algorithm with linear effects

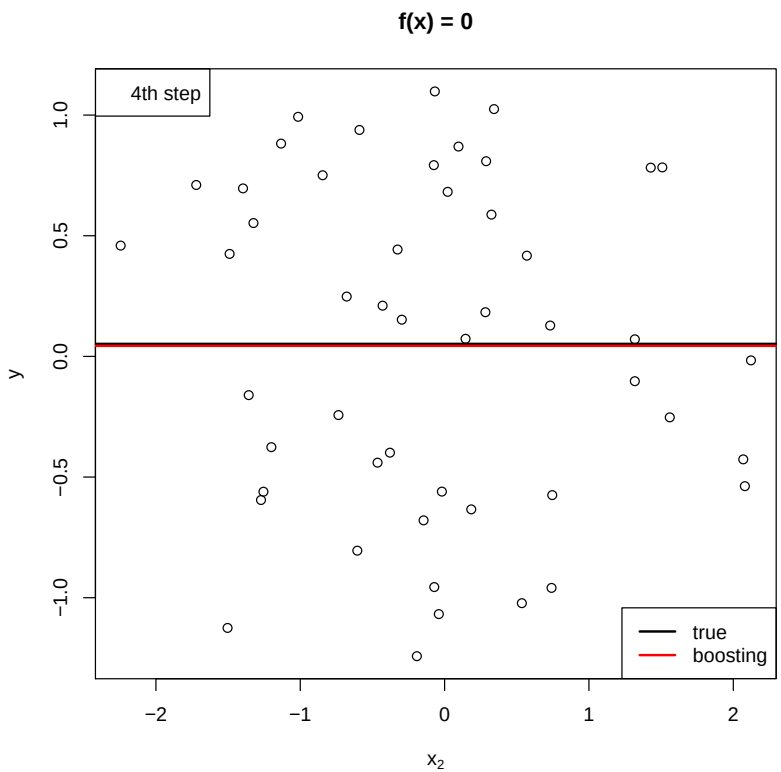




(3rd iteration: $\hat{y} = 0.043 + 0.080 \cdot x_1$)

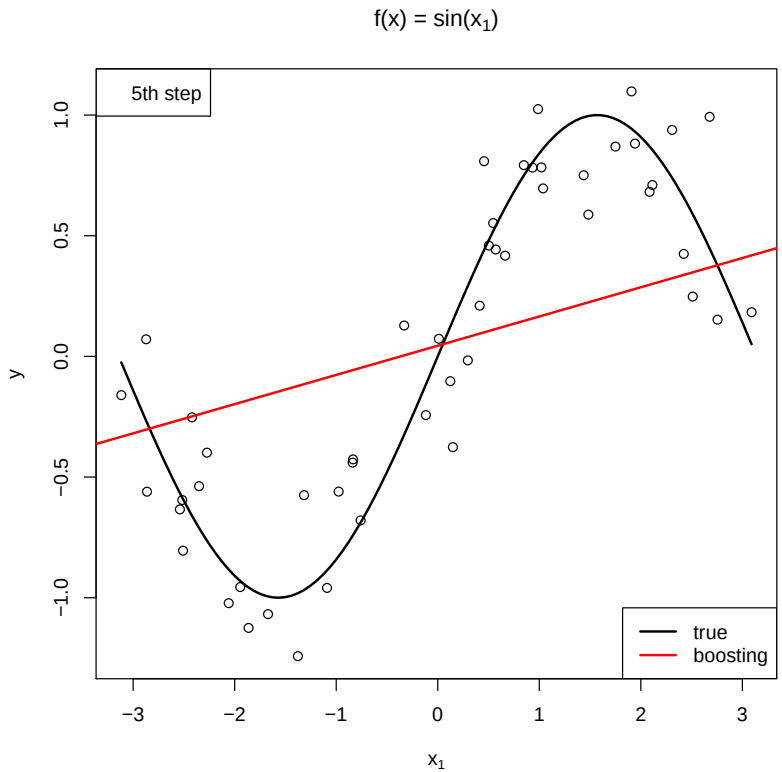
Component-wise Gradient Boosting: algorithm with linear effects

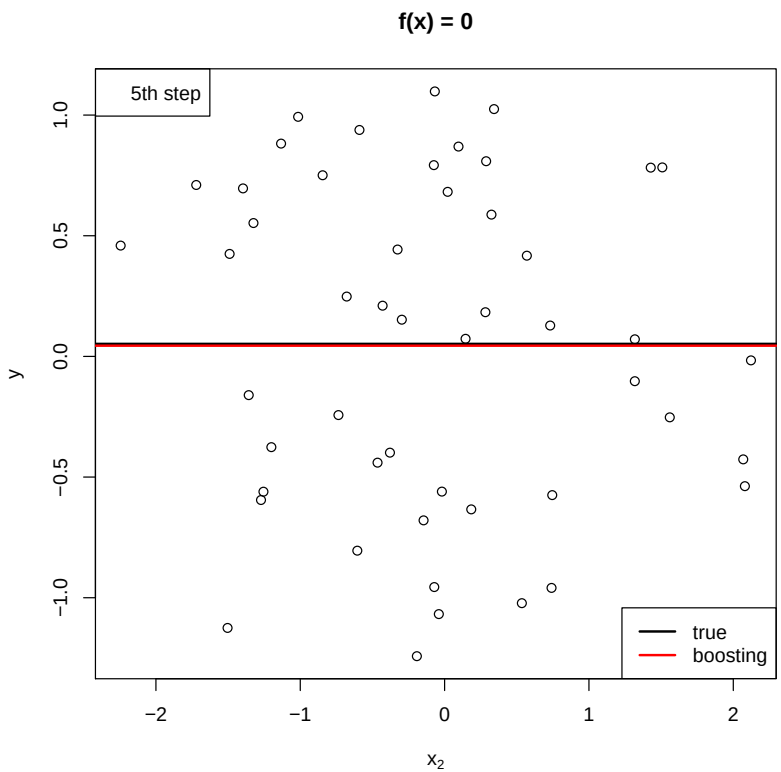




(4th iteration: $\hat{y} = 0.043 + 0.102 \cdot x_1$)

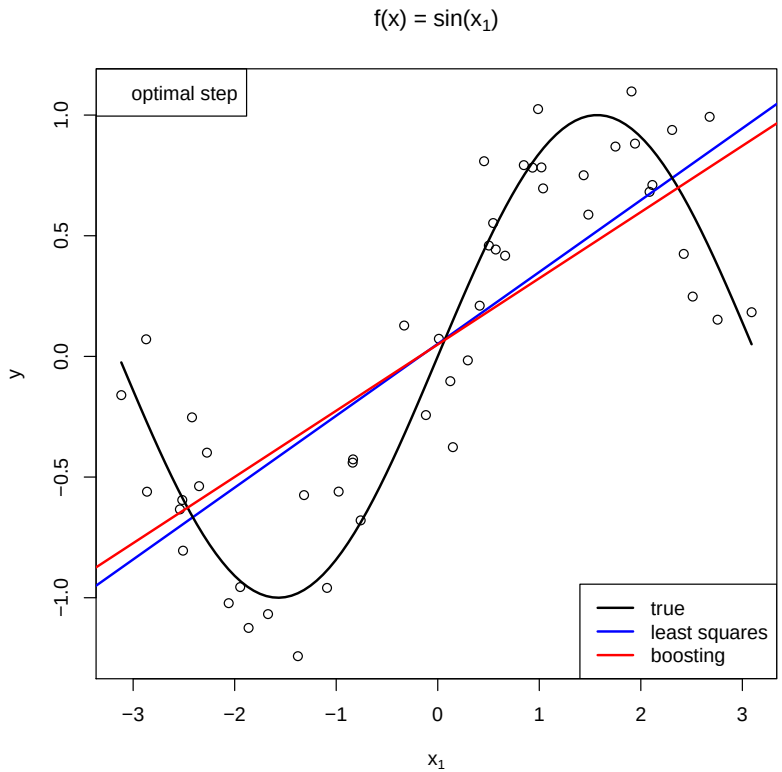
Component-wise Gradient Boosting: algorithm with linear effects

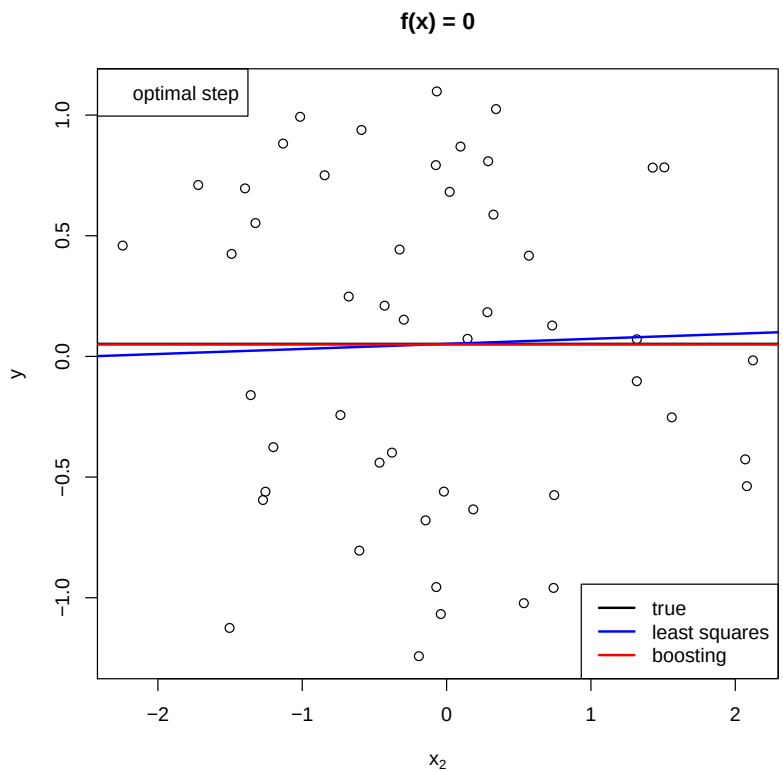




(5th iteration: $\hat{y} = 0.044 + 0.121 \cdot x_1$)

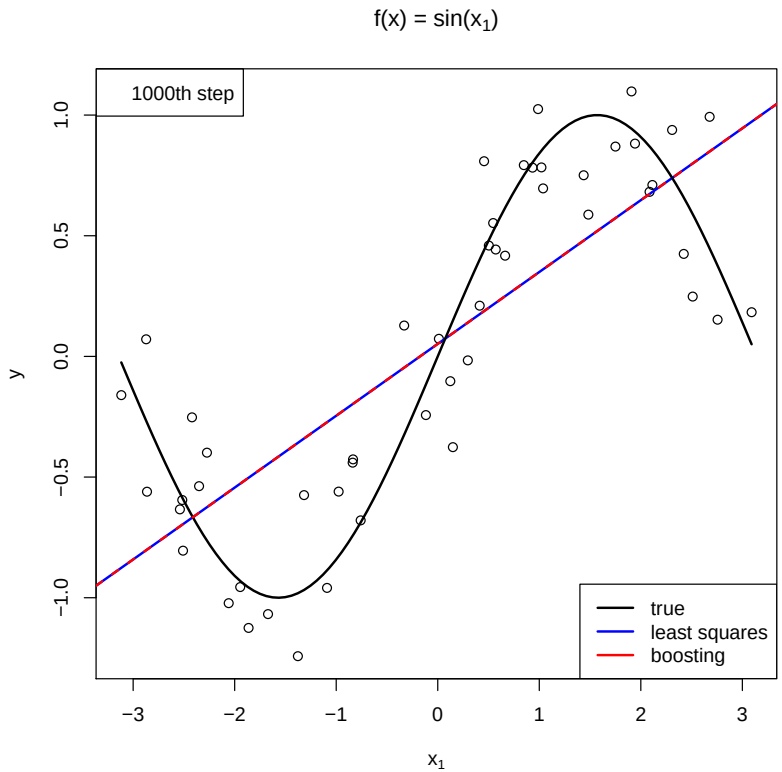
Component-wise Gradient Boosting: algorithm with linear effects

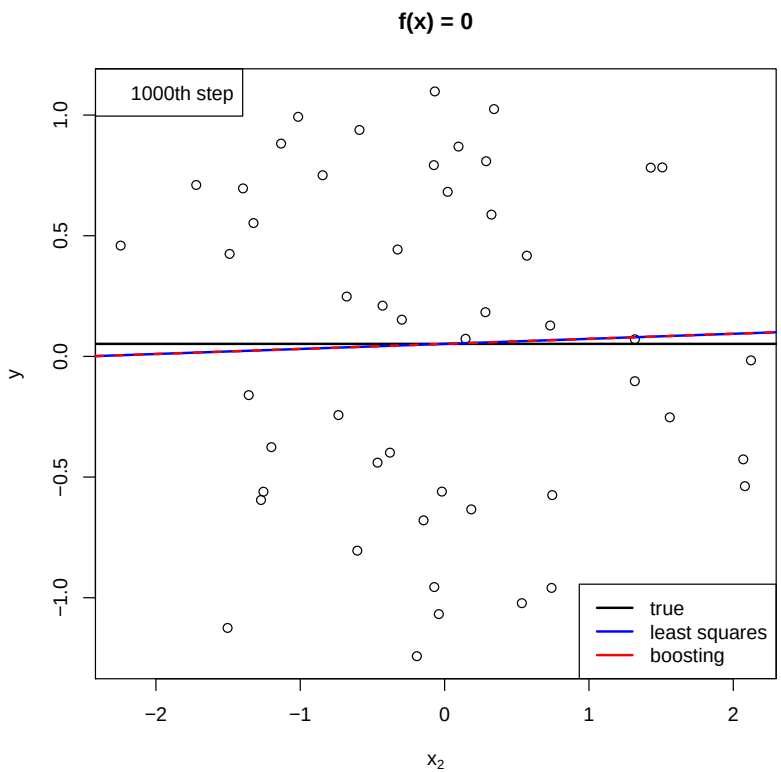




(best iteration: $\hat{y} = 0.048 + 0.274 \cdot x_1$)

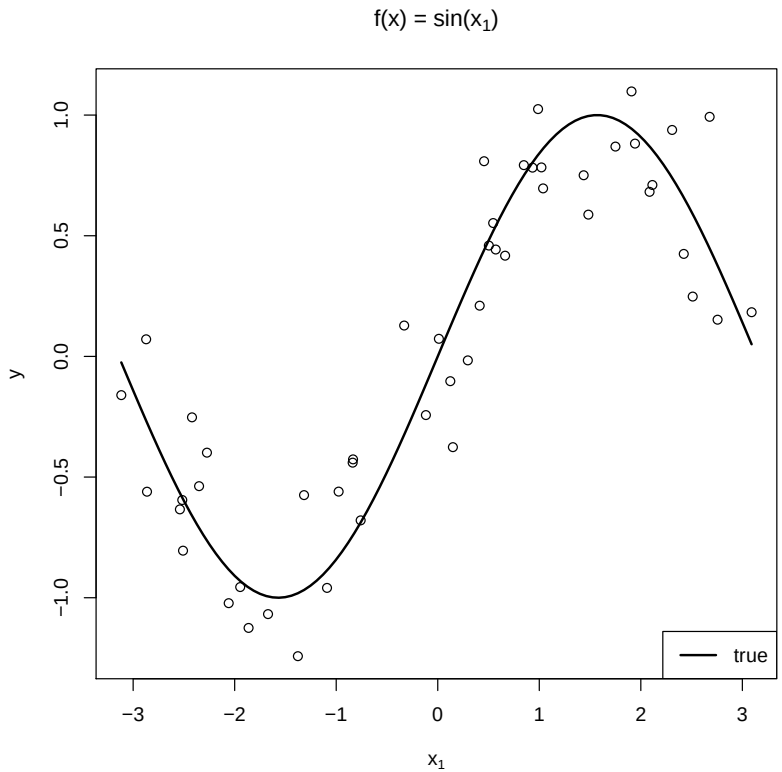
Component-wise Gradient Boosting: algorithm with linear effects

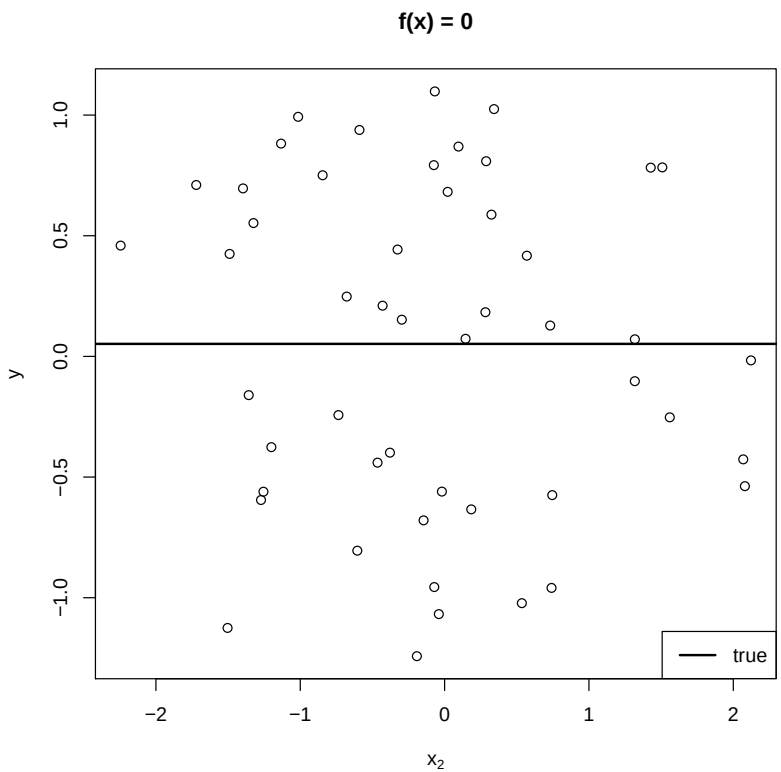




(1000th iteration: $\hat{y} = 0.050 + 0.030 \cdot x_1 + 0.020 \cdot x_1$)

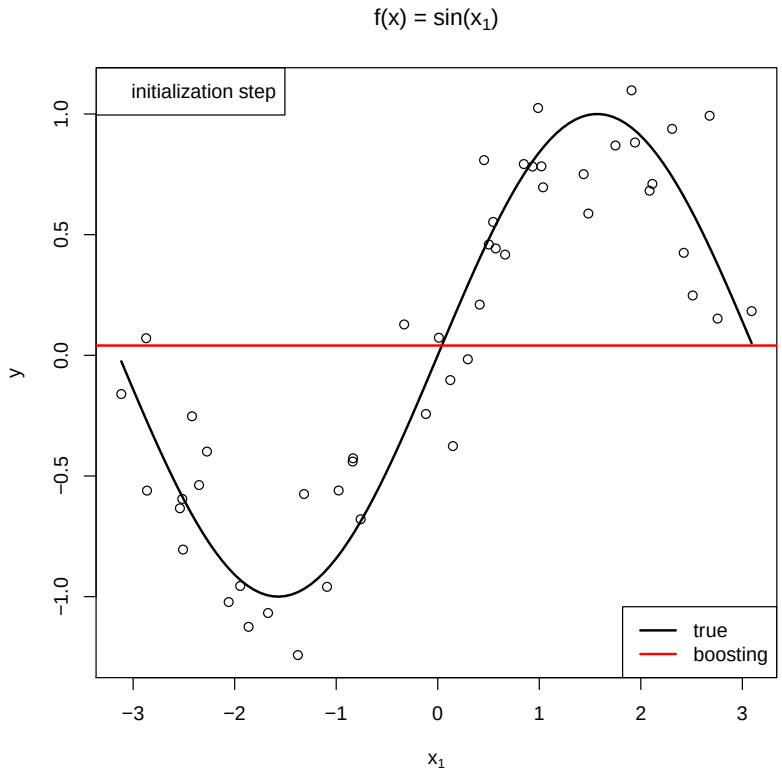
Component-wise Gradient Boosting: algorithm with splines

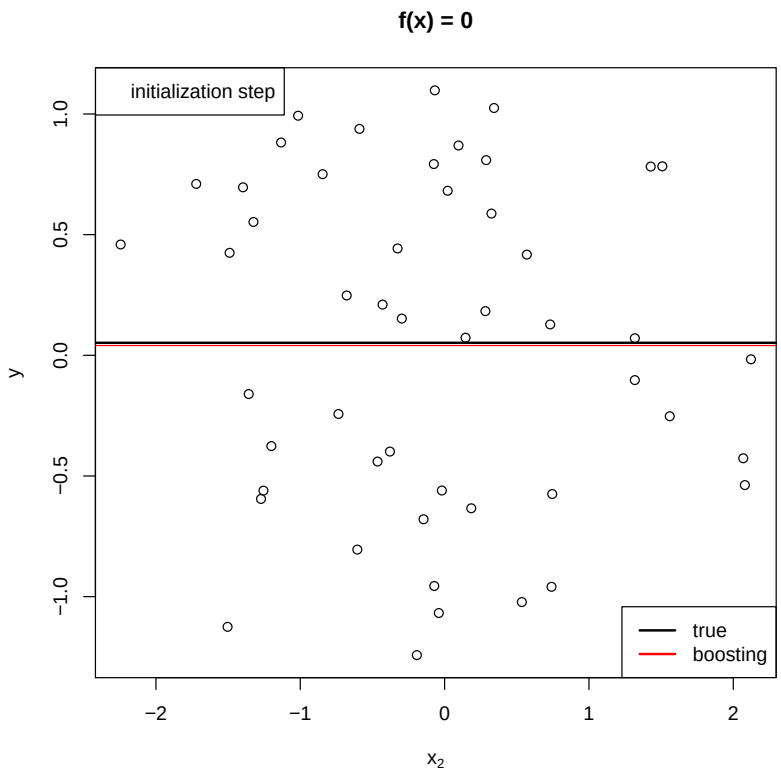




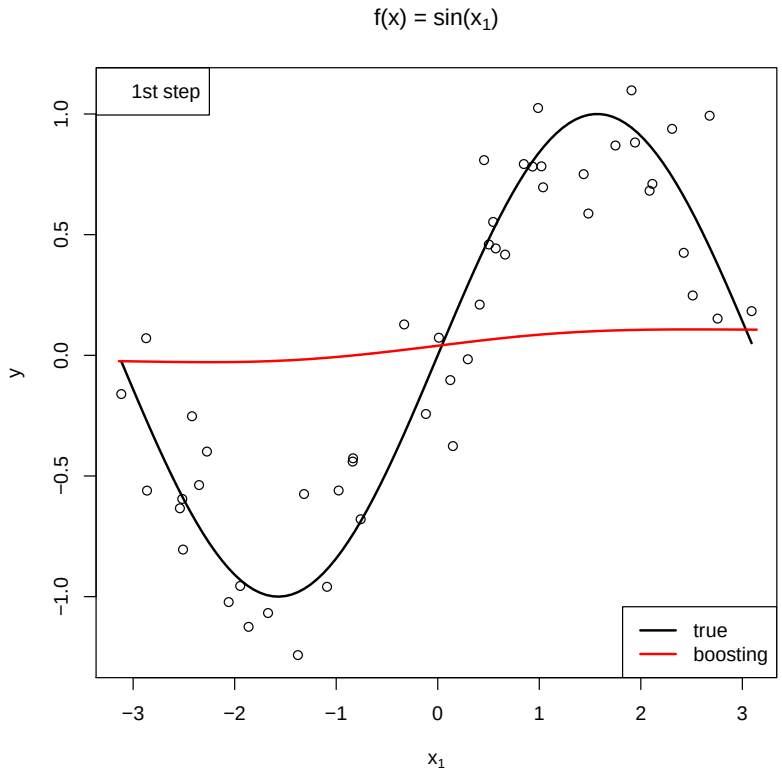
(data generated from $y = \sin(x_1) + 0 \cdot x_2 + \epsilon$, $\epsilon \sim N(0, 0.25^2)$)

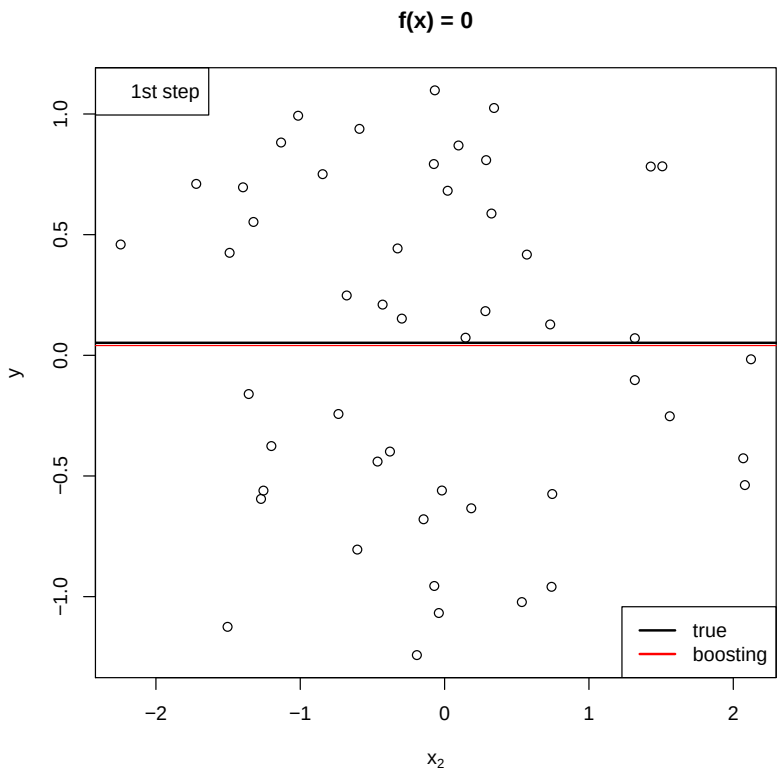
Component-wise Gradient Boosting: algorithm with splines



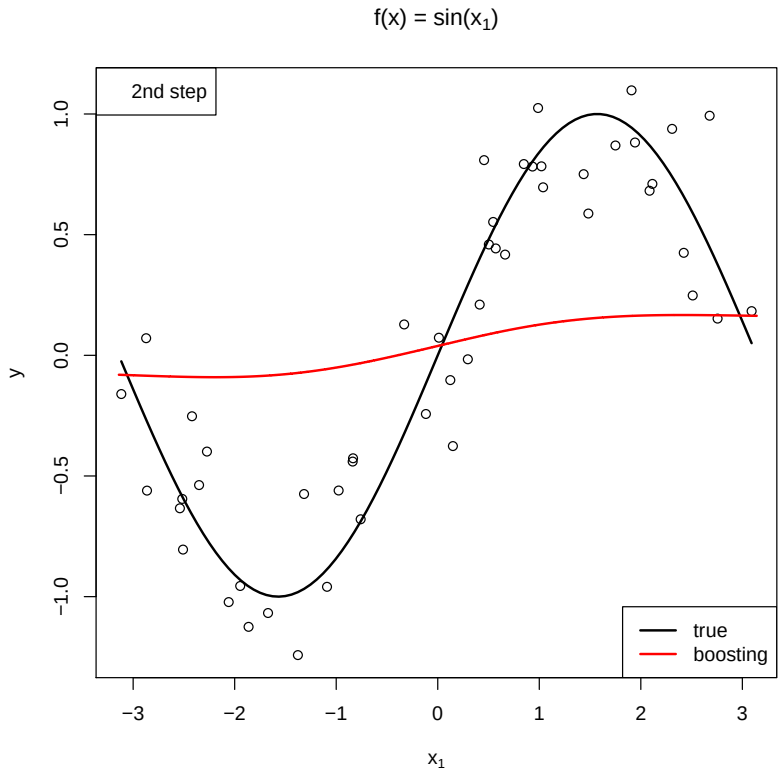


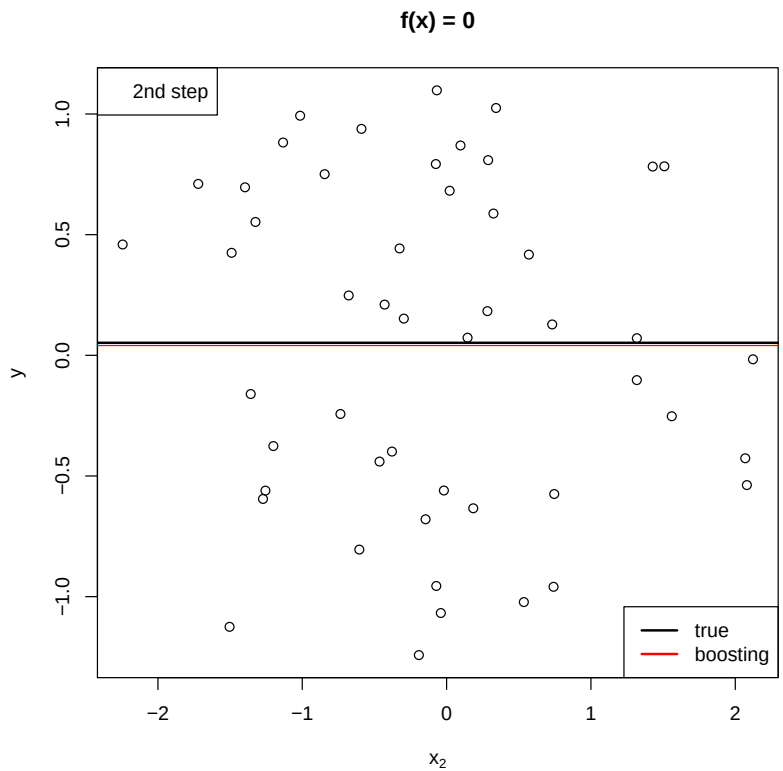
Component-wise Gradient Boosting: algorithm with splines



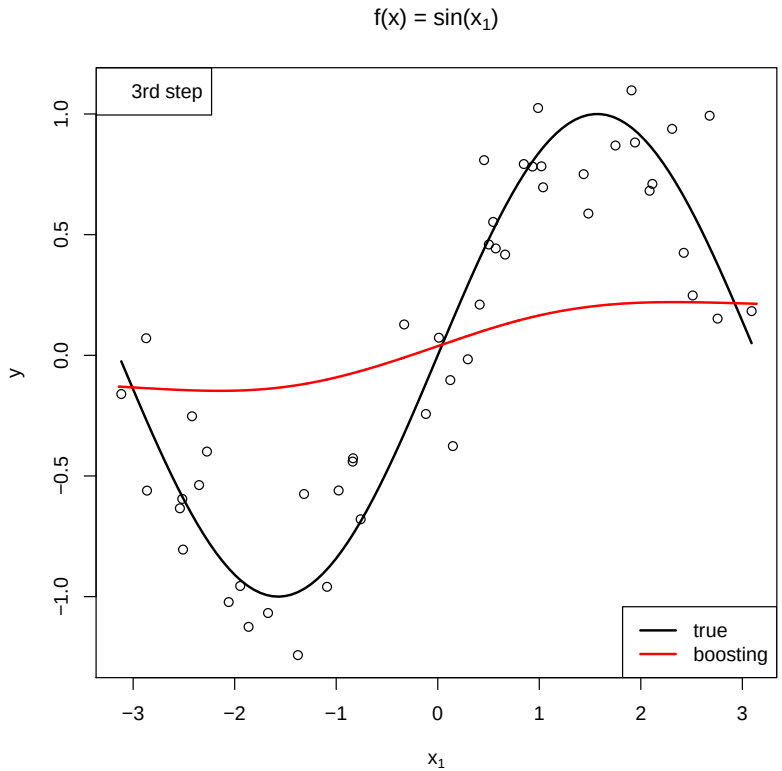


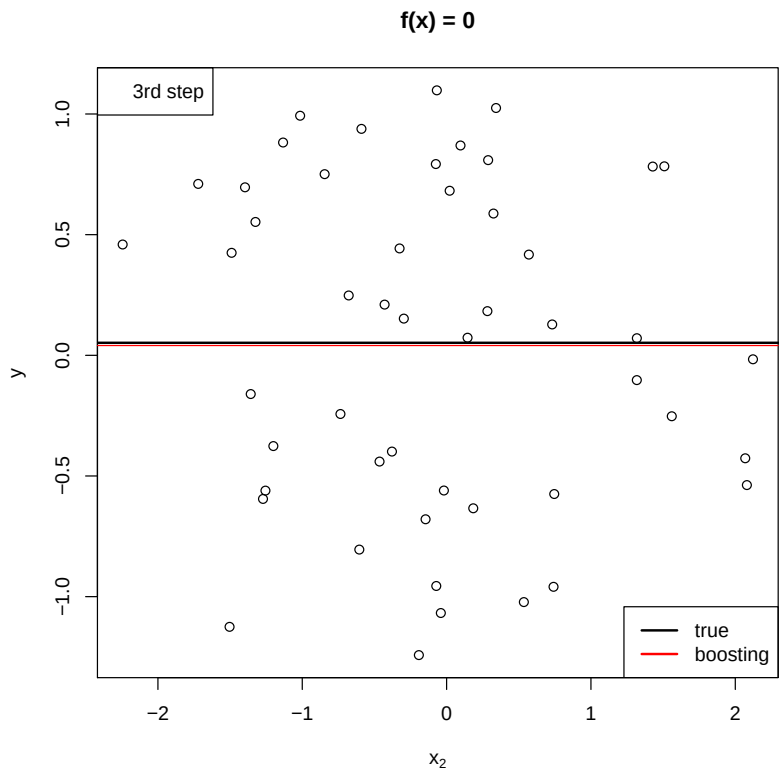
Component-wise Gradient Boosting: algorithm with splines



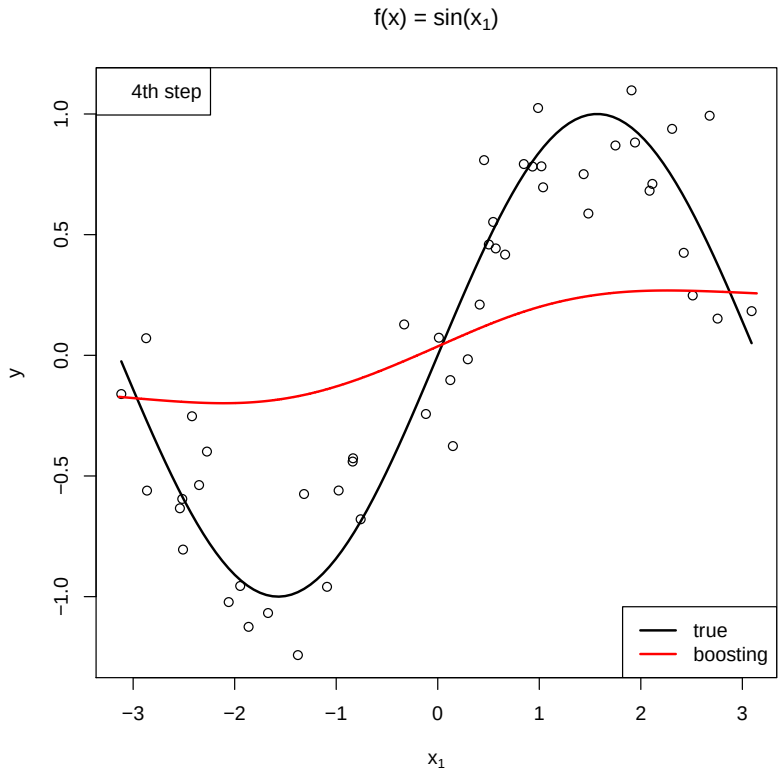


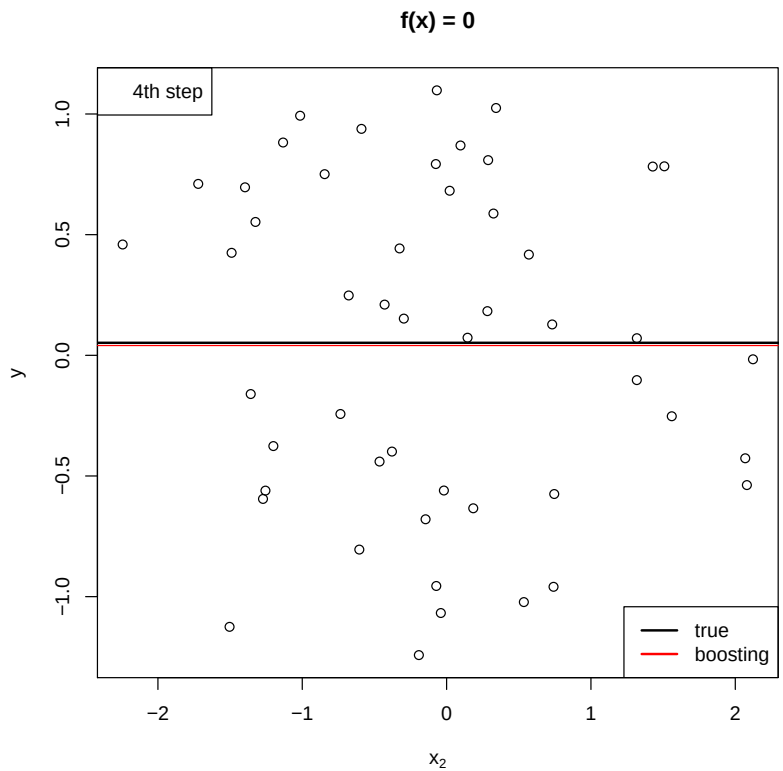
CComponent-wise Gradient Boosting: algorithm with splines



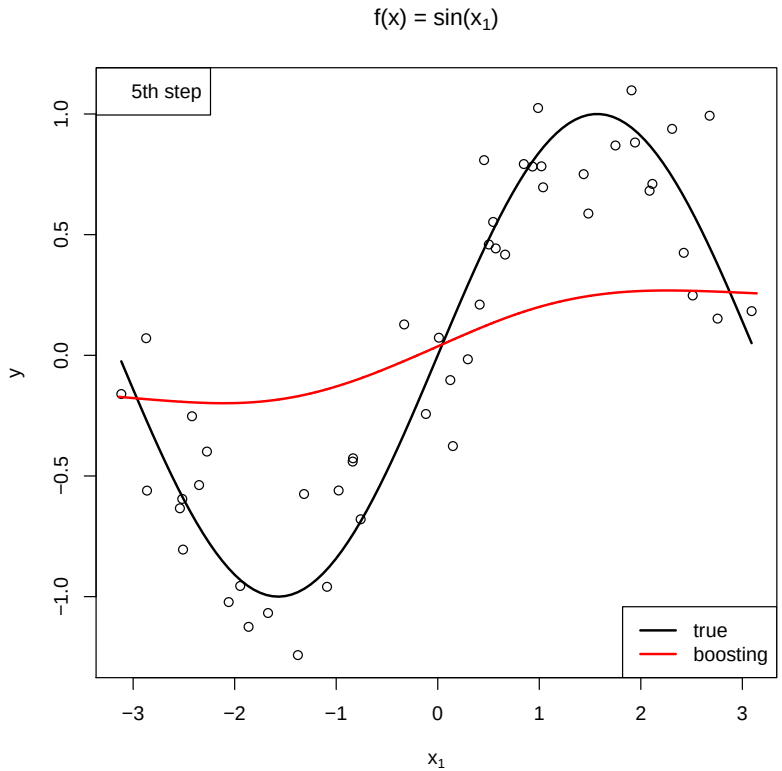


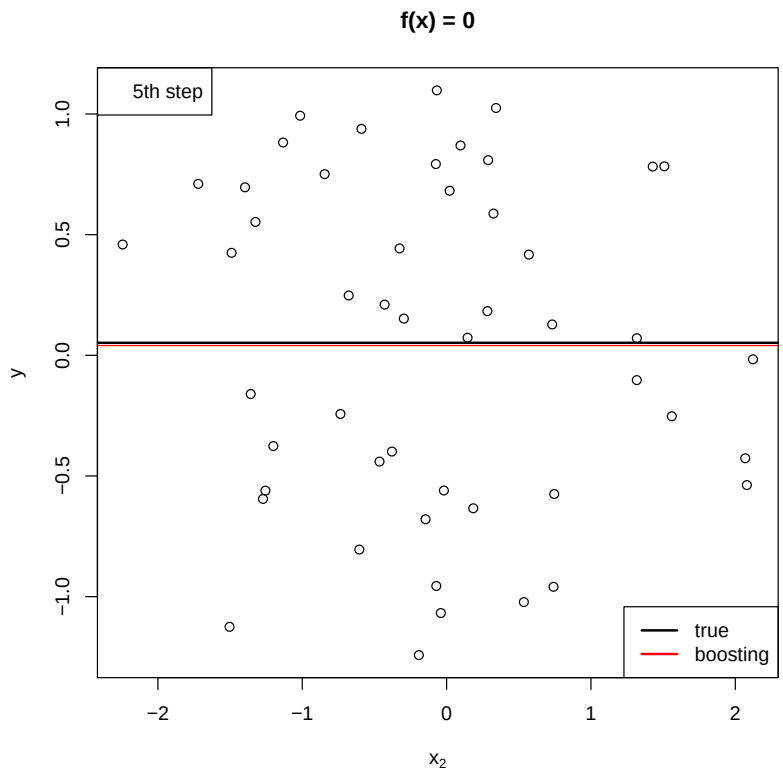
Component-wise Gradient Boosting: algorithm with splines



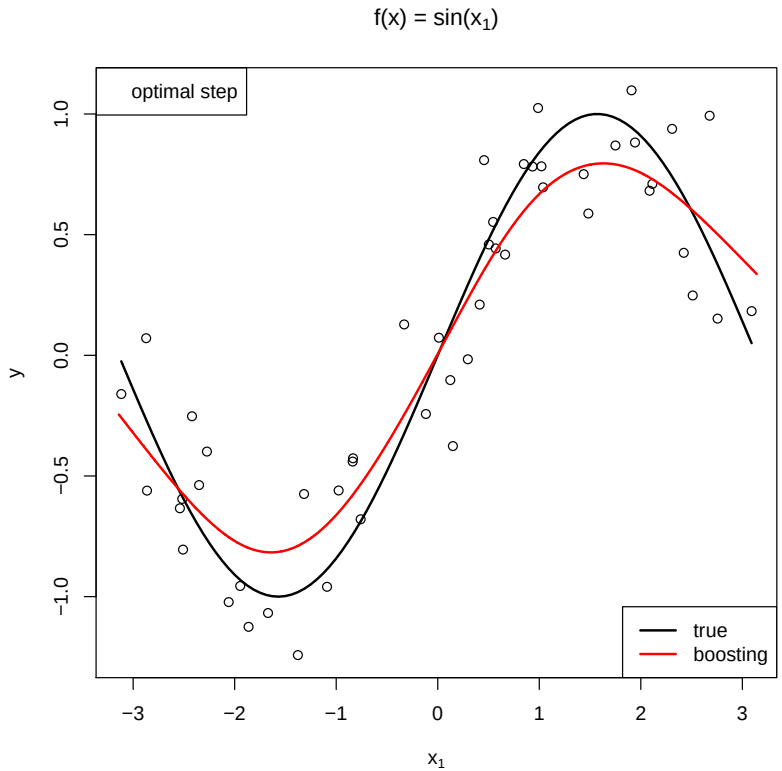


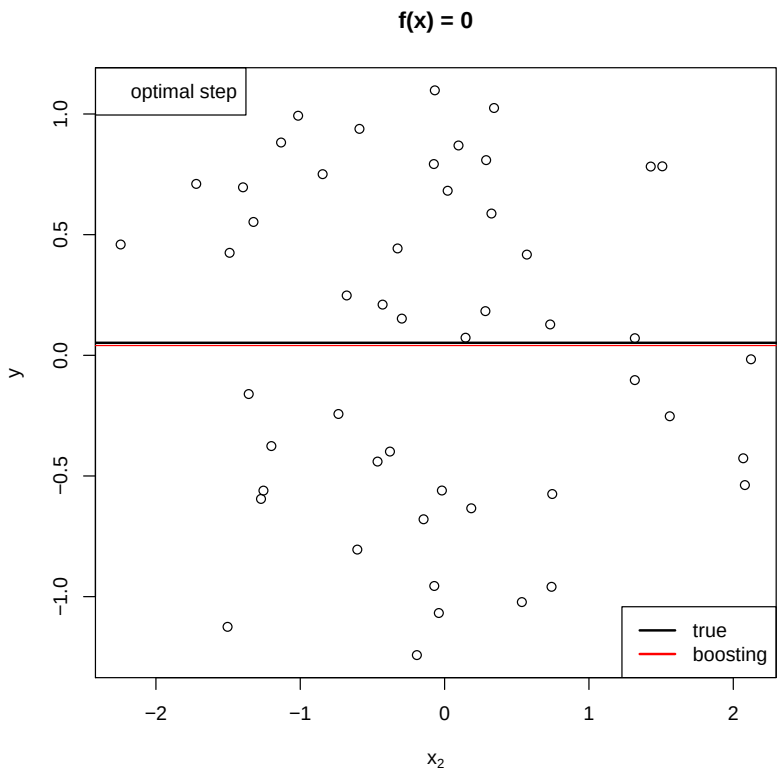
Component-wise Gradient Boosting: algorithm with splines



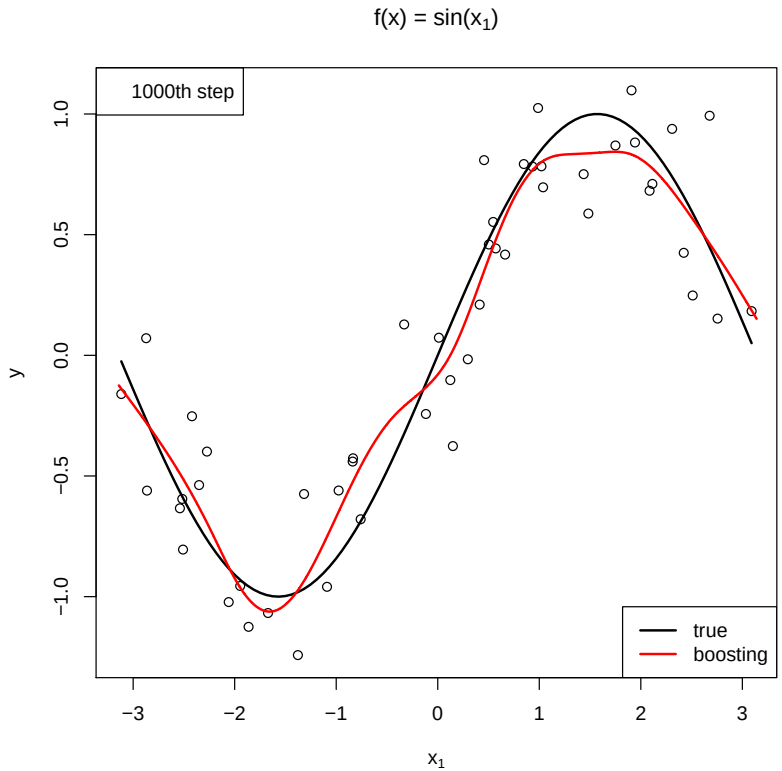


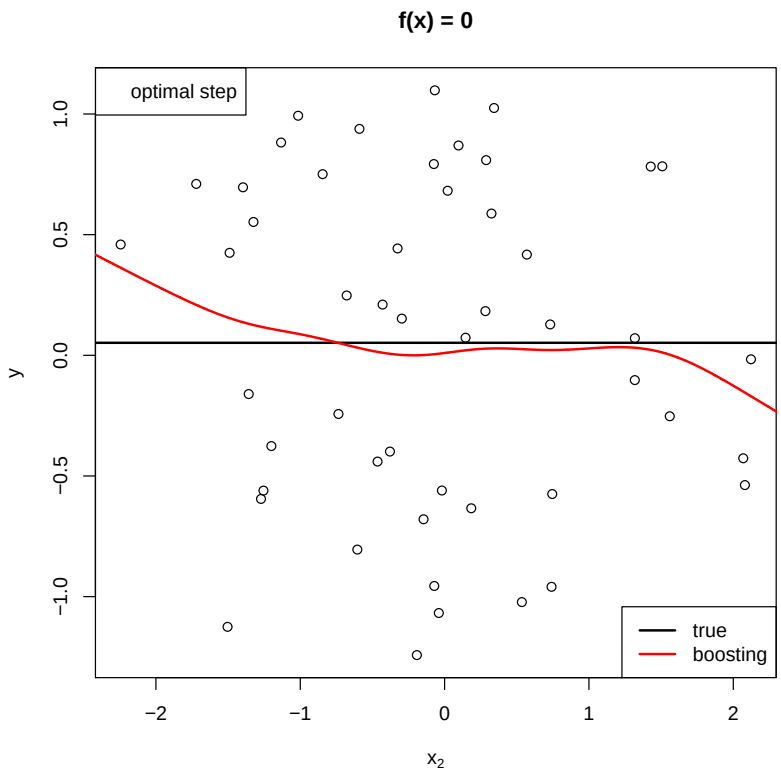
Component-wise Gradient Boosting: algorithm with splines



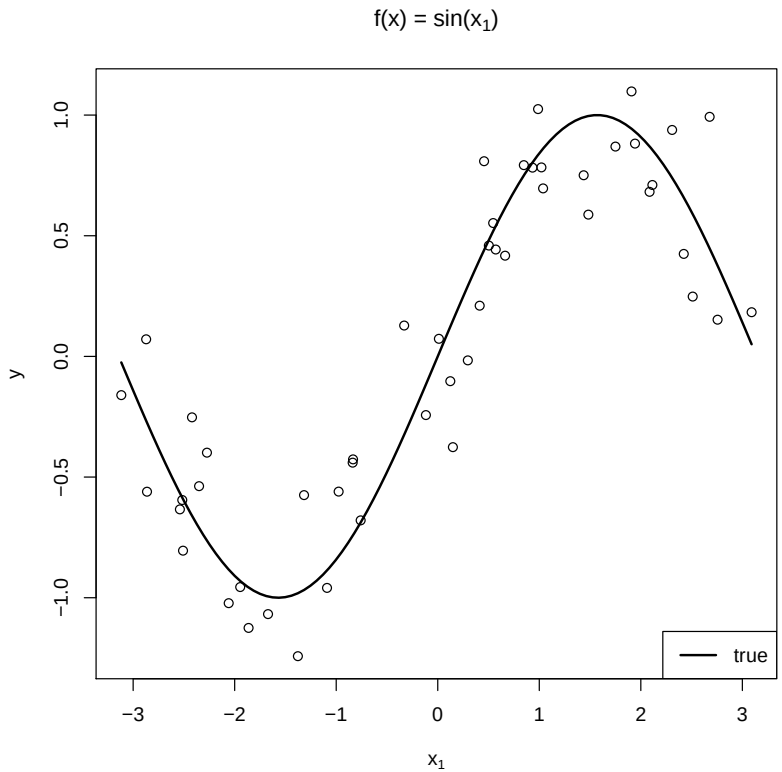


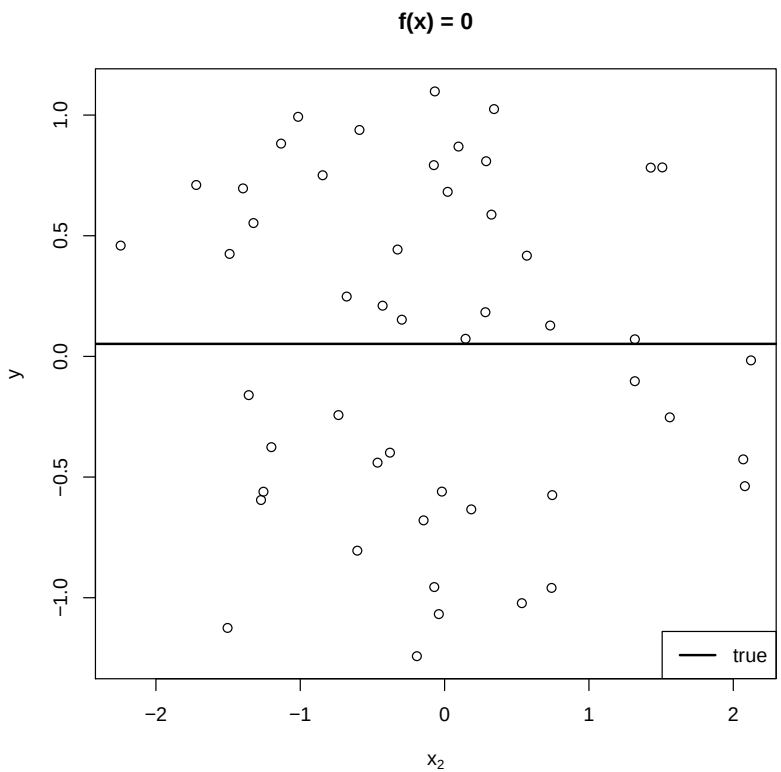
Component-wise Gradient Boosting: algorithm with splines





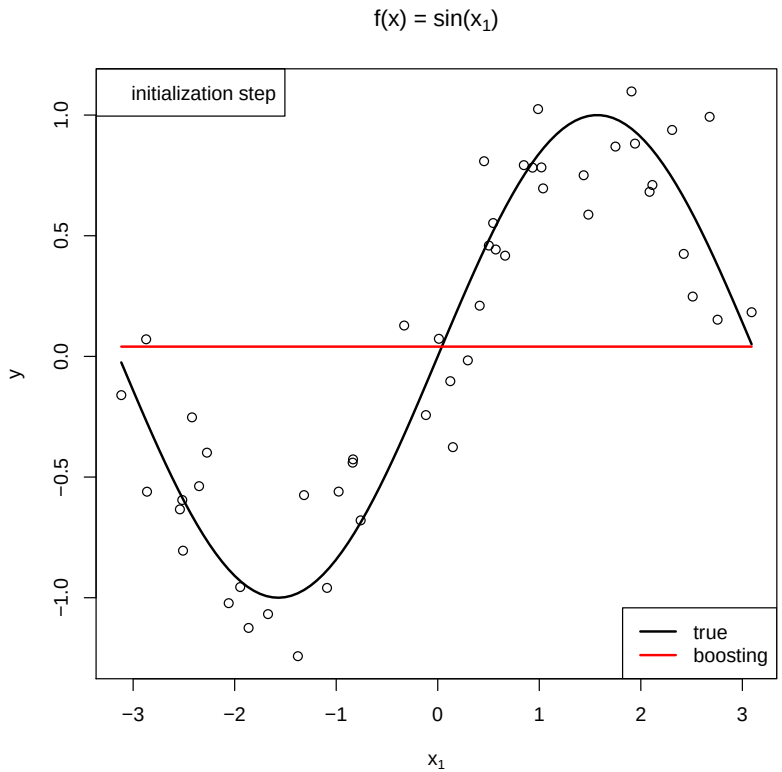
Component-wise Gradient Boosting: algorithm with stumps

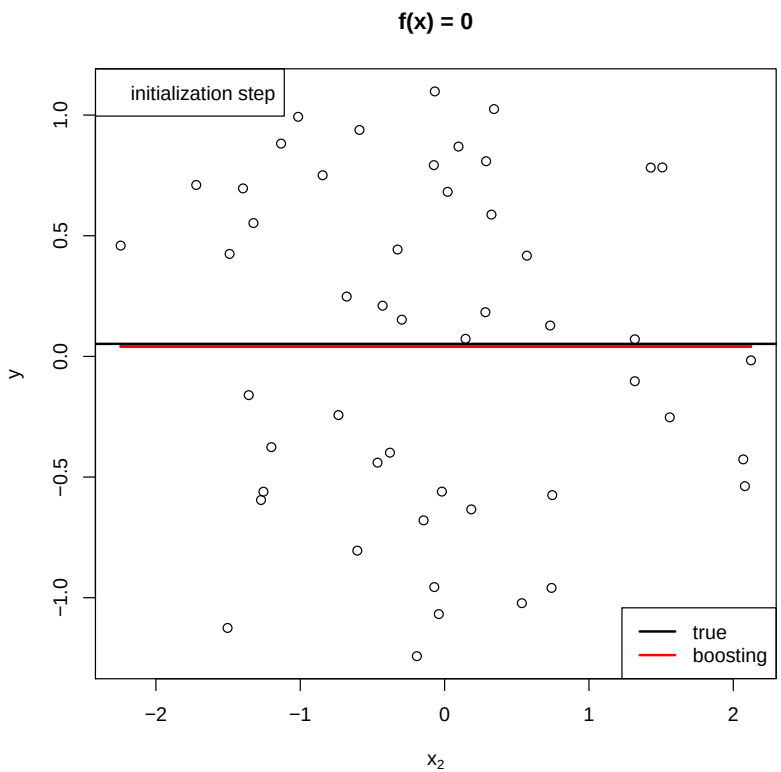




(data generated from $y = \sin(x_1) + 0 \cdot x_2 + \epsilon$, $\epsilon \sim N(0, 0.25^2)$)

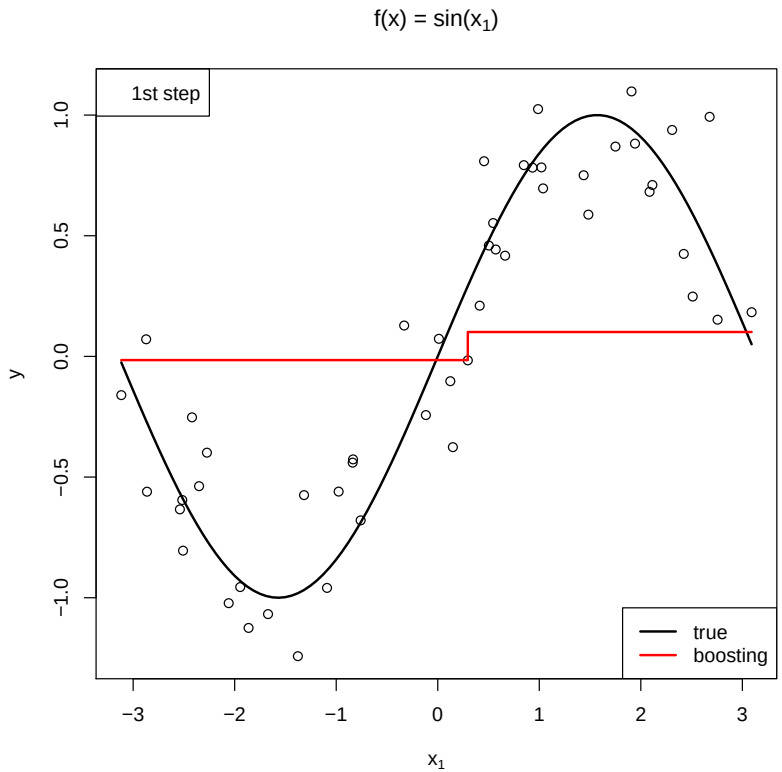
Component-wise Gradient Boosting: algorithm with stumps

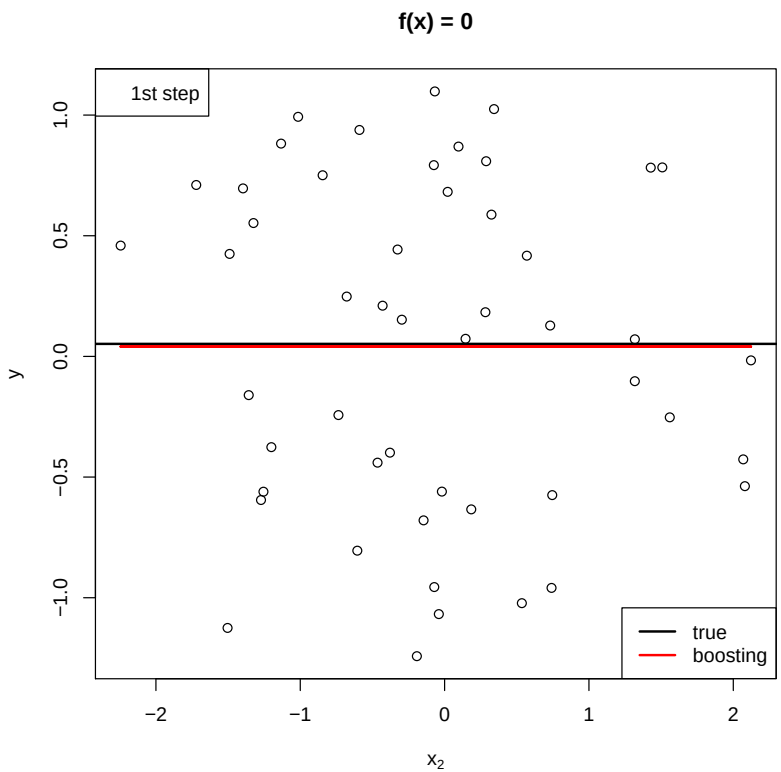




(starting from the null model: $\hat{y} = 0.040$)

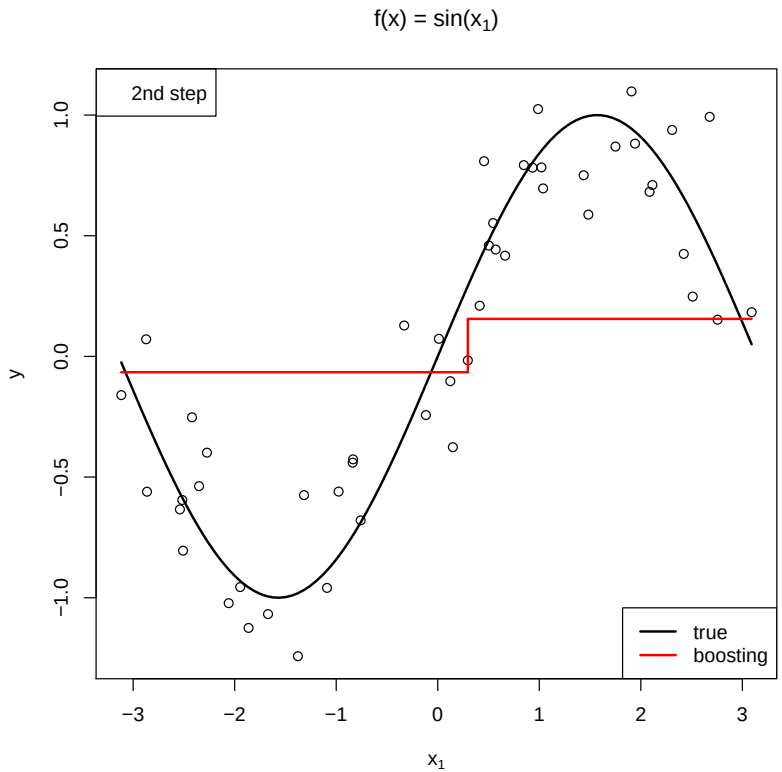
Component-wise Gradient Boosting: algorithm with stumps

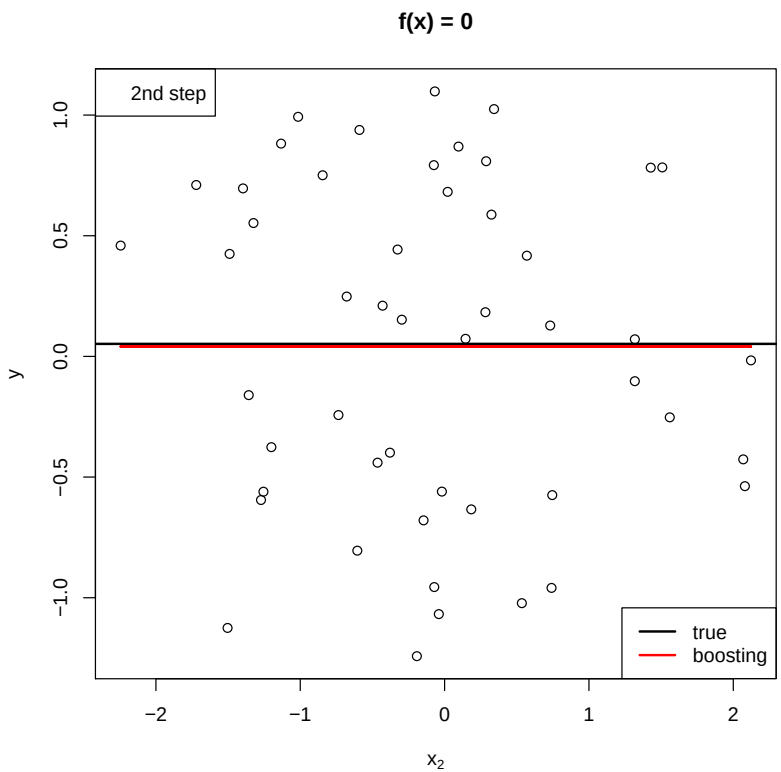




(1st iter.: $\hat{y} = 0.040 - 0.056 \cdot I(x_1 \leqslant 0.29) + 0.060 \cdot I(x_1 > 0.29)$)

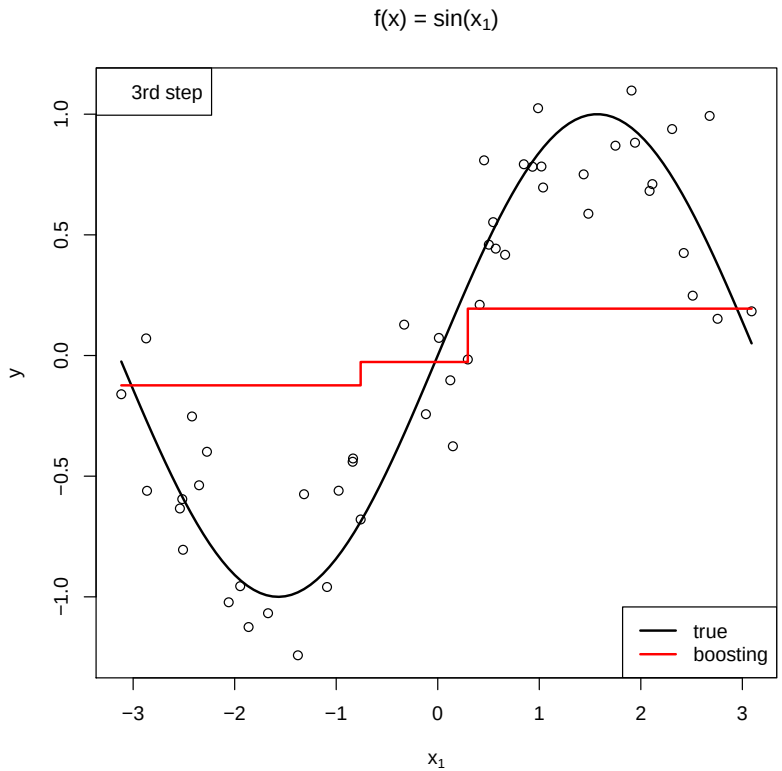
Component-wise Gradient Boosting: algorithm with stumps

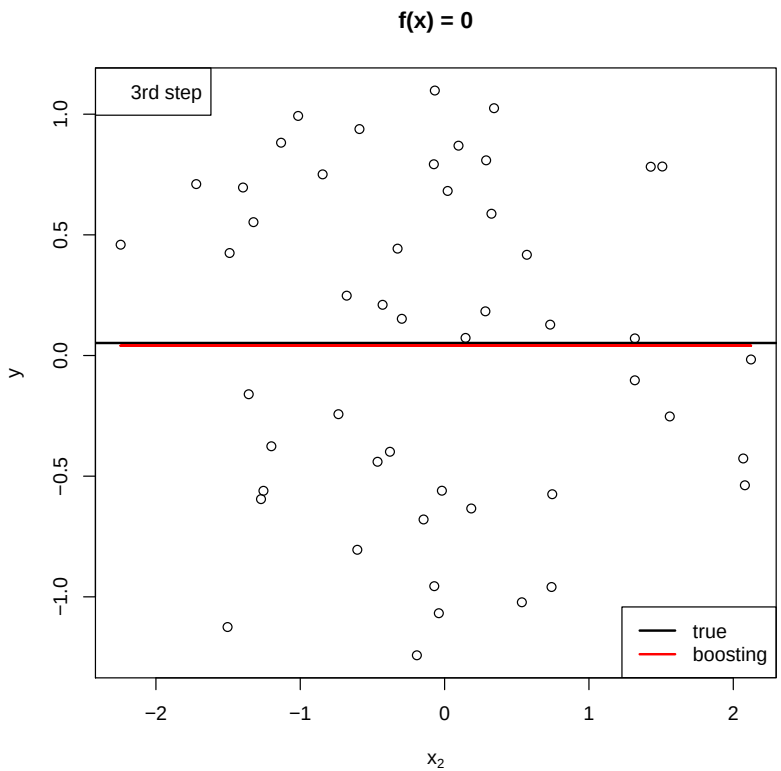




(2nd iter.: $\hat{y} = 0.041 - 0.106 \cdot I(x_1 \leqslant 0.29) + 0.115 \cdot I(x_1 > 0.29)$)

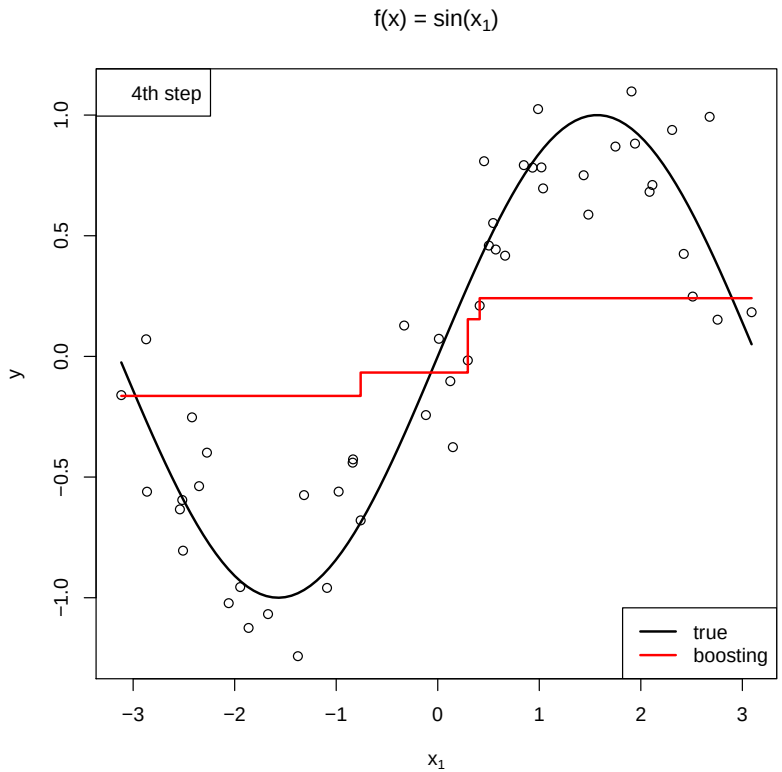
Component-wise Gradient Boosting: algorithm with stumps

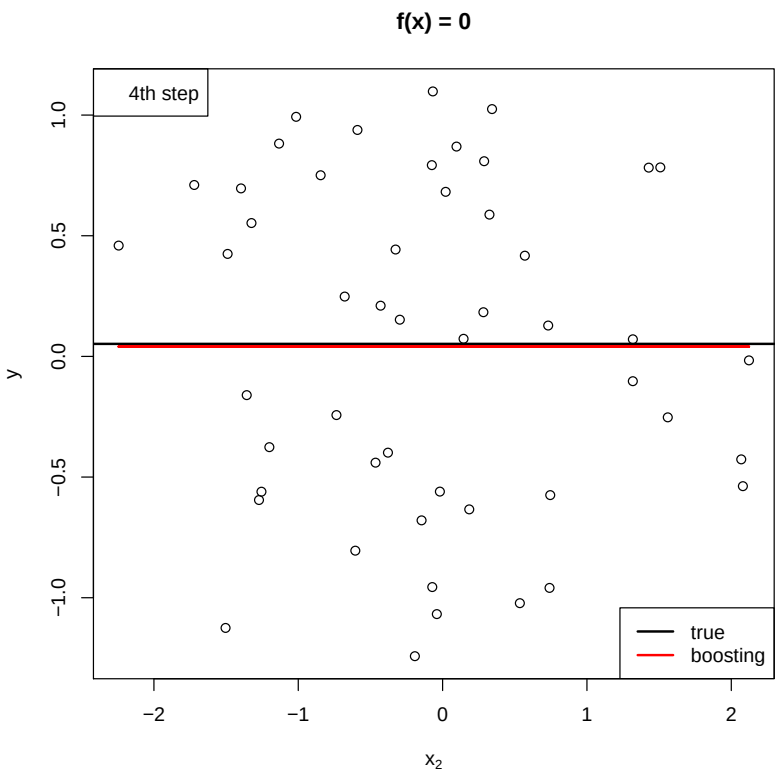




(3rd iter.: $\hat{y} = \hat{a}_0 + \hat{a}_1 \cdot I(x_1 \leqslant -0.75) + \hat{a}_2 \cdot I(-0.75 < x_1 \leqslant 0.29) + \hat{a}_3 \cdot I(x_1 > 0.29)$)

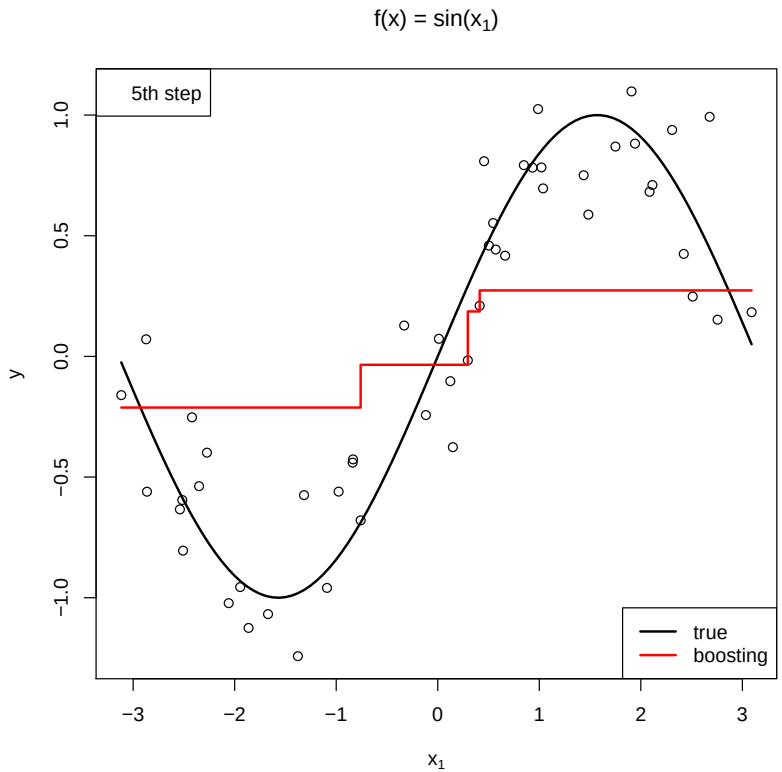
Component-wise Gradient Boosting: algorithm with stumps

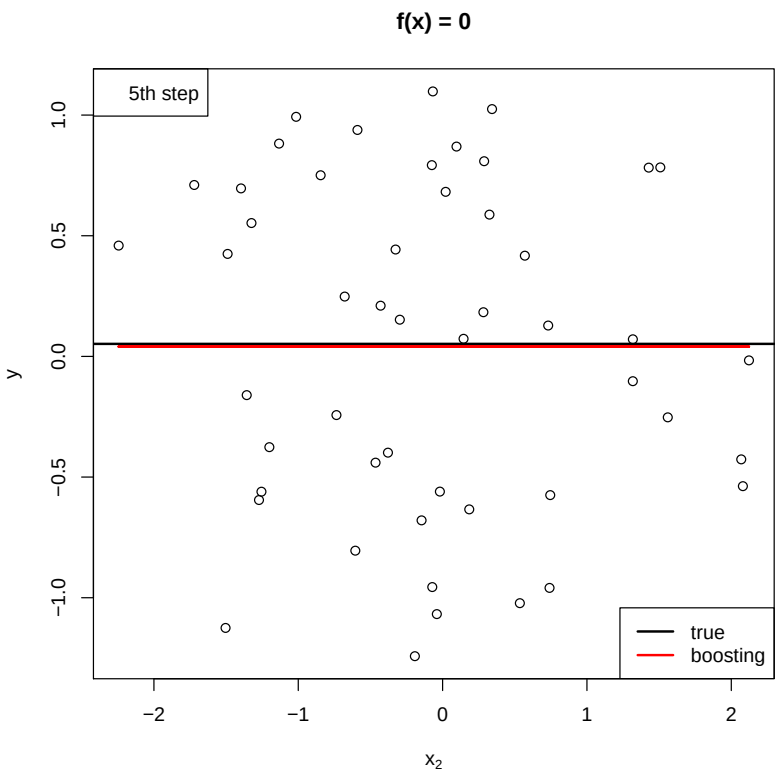




(4th iteration: . . .)

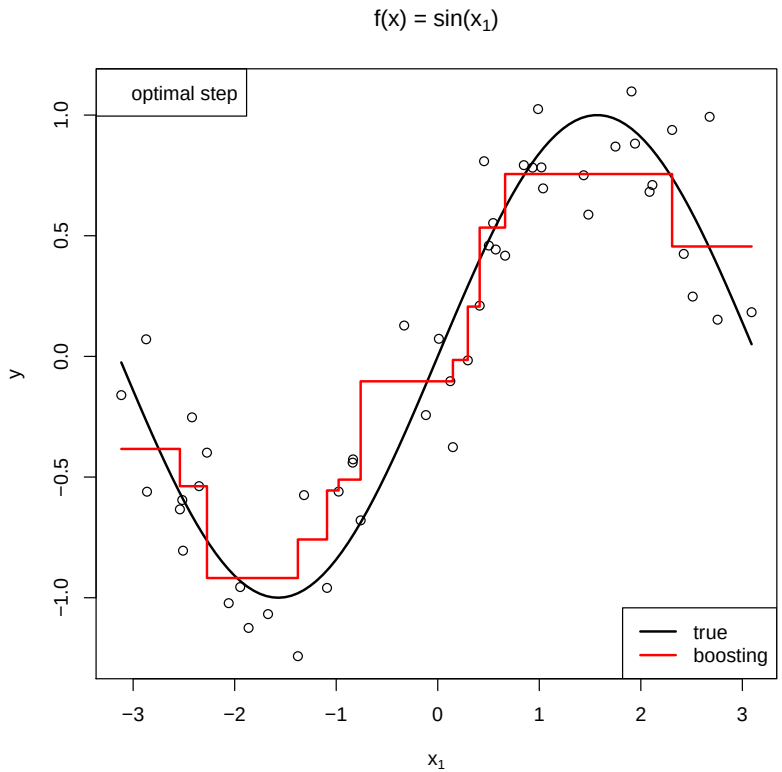
Component-wise Gradient Boosting: algorithm with stumps

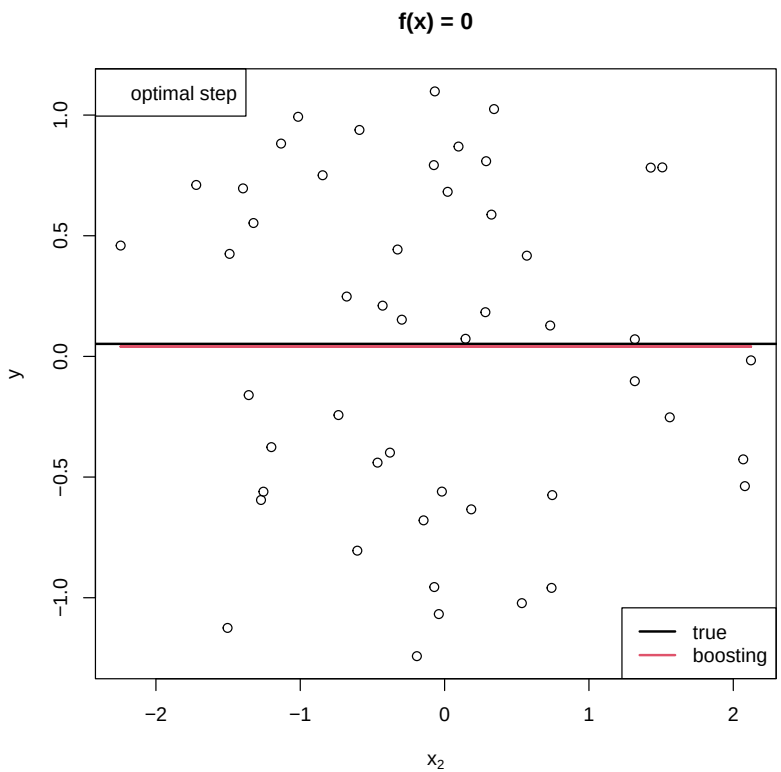




(5th iteration: . . .)

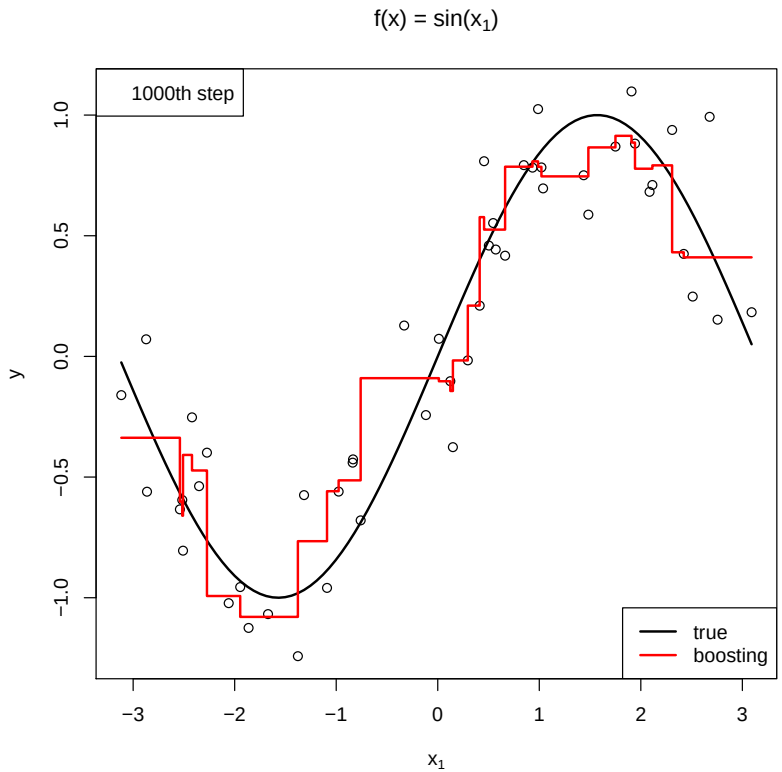
Component-wise Gradient Boosting: algorithm with stumps

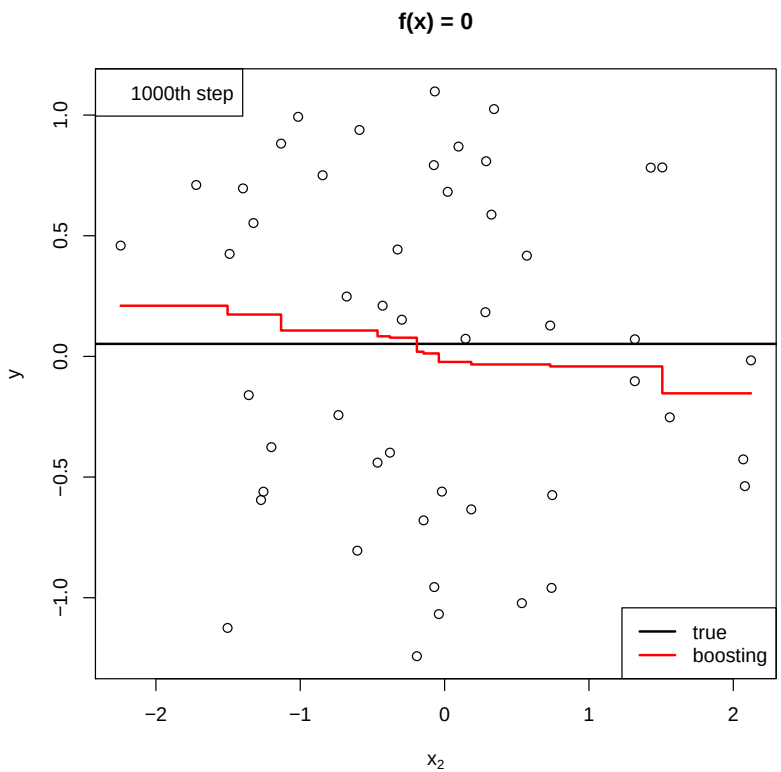




(best iteration: . . .)

Component-wise Gradient Boosting: algorithm with stumps





(1000th iteration: . . .)

Boosting: tuning parameters

- The update step is regulated by the shrinkage parameter ν;
- as long as its magnitude is reasonable, the choice of the penalty parameter does not influence the procedure;
- the choice of the number of iterations m_{stop} is highly relevant;
- m_{stop} (complexity parameter) influences variable selection properties and model sparsity;
- m_{stop} controls the amount of shrinkage;
  - m_{stop} too small results in a model which is not able to describe the data variability;
  - an excessively large m_{stop} causes overfitting and causes the selection of irrelevant variables.
- there is no standard approach $\rightarrow$ repeated cross-validation (Seibold et al., 2018).

Tree-based boosting: introduction

The base (weak) learner in a boosting algorithm can be a tree:

- largely used in practice;
- very powerful and fast algorithm;
- we lose part of the statistical interpretation;
- we can still compute the variable importance (see lecture 9).

Tree-based boosting: algorithm

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^N L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute

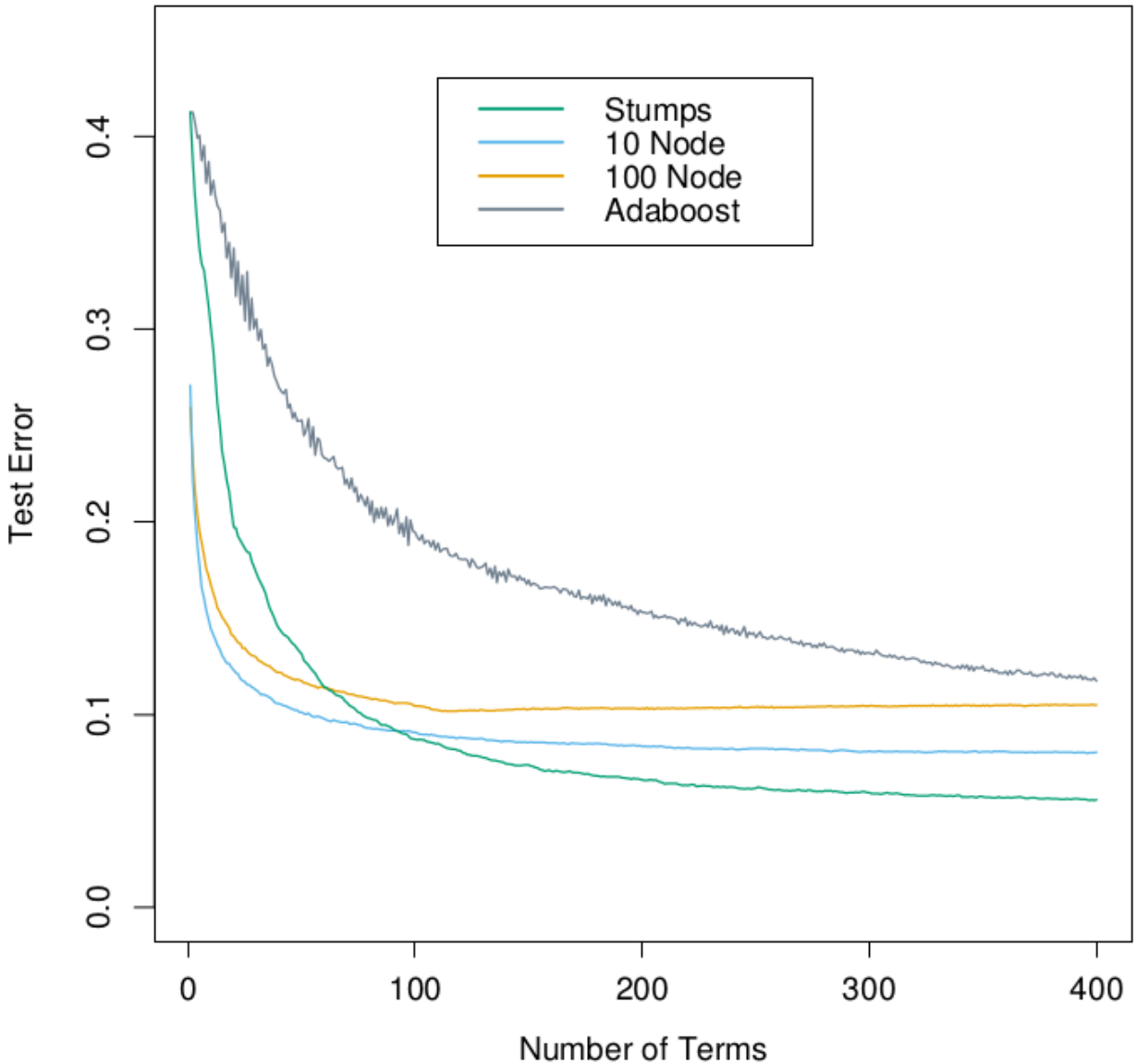
$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$

   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute

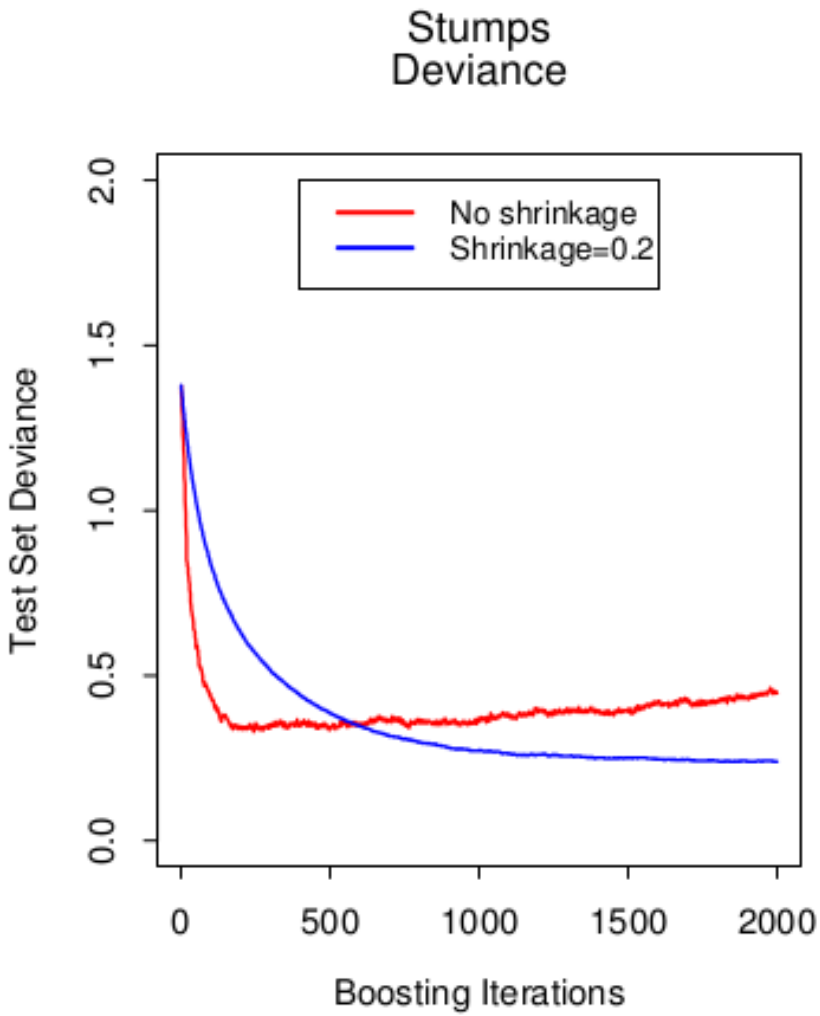
$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$

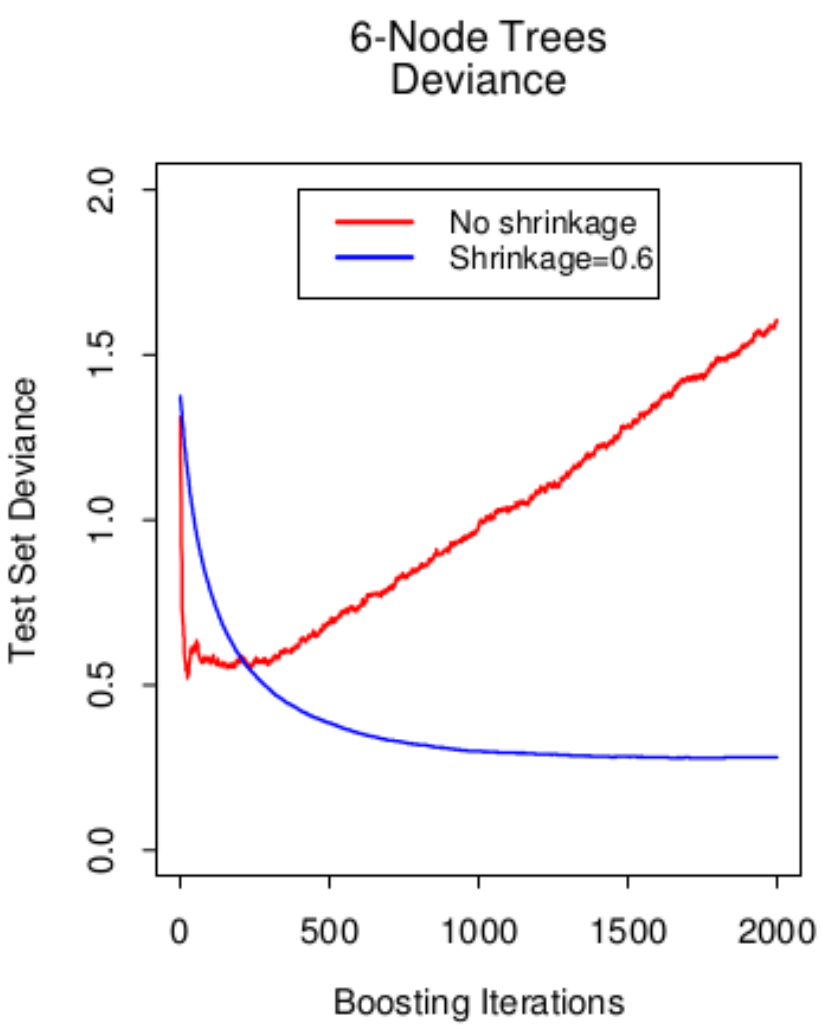
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

Tree-based boosting: importance of "weakness"

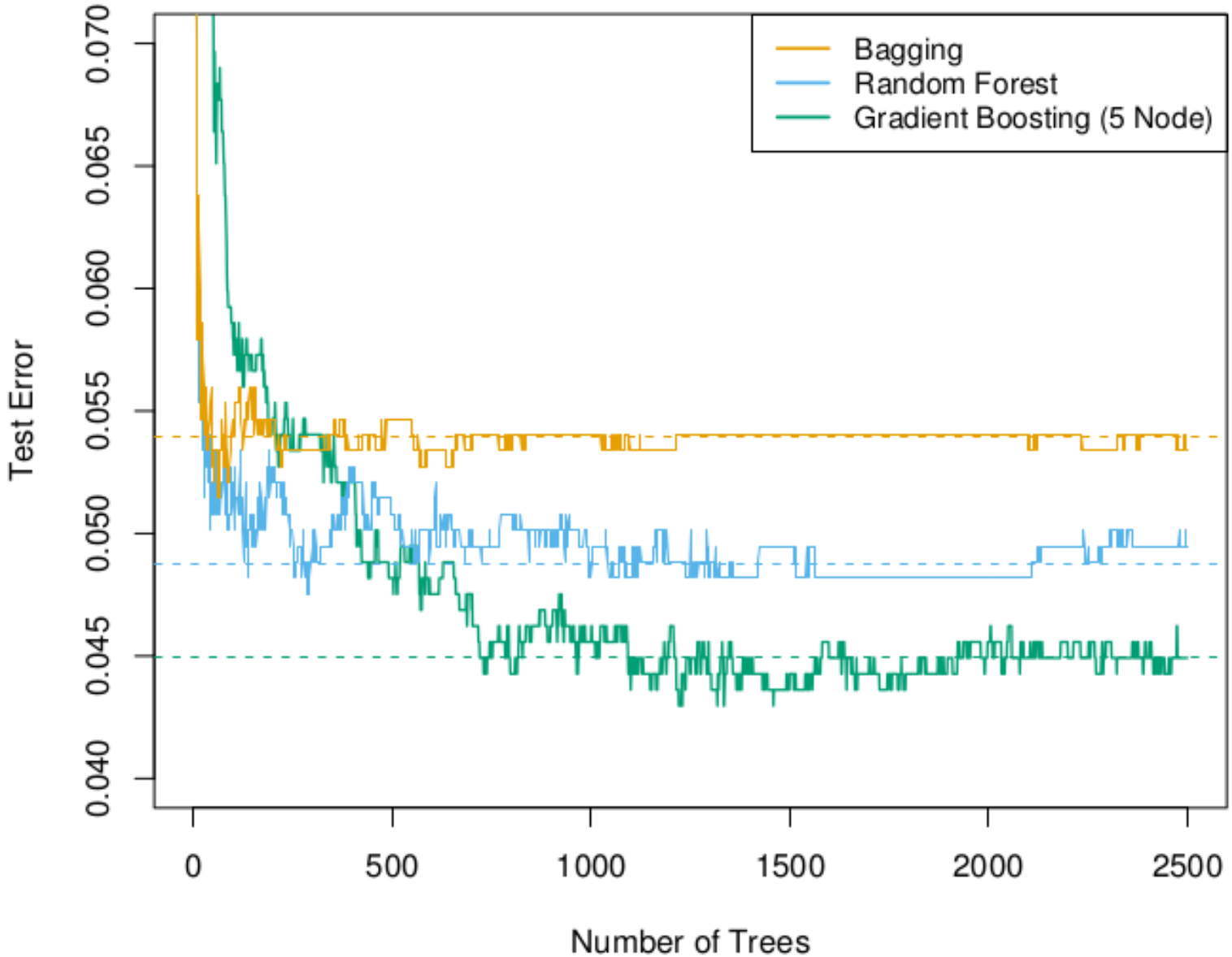


Tree-based boosting: importance of "shrinkage"





Tree-based boosting: comparison



eXtreme Gradient Boosting: introduction

Described by Chen & Guestrin (2016):

- released in 2014;
- already in 2015, among the 29 challenge winning solutions published at Kaggle's blog, 17 solutions used XGBoost;
- focus on scalability;
- handles very well collinearity and missing data;
- several mathematical and computational improvements;
- mainly trees as base-learners.

eXtreme Gradient Boosting: details

At each iteration a tree $f_k(\boldsymbol{x})$ is derived by minimizing

$$\text{obj}(f_k(\boldsymbol{x})) = \sum_i^N L\left(y_i, \hat{f}(\boldsymbol{x}_i)^{[k-1]} + f_k(\boldsymbol{x}_i)\right) + \Omega(f_k(\boldsymbol{x}))$$

where

- $(\boldsymbol{x}_i, y_i)$ is the i-th observation;
- $\sum_i^N L\left(y_i, \hat{f}(\boldsymbol{x}_i)^{[k-1]} + f_k(\boldsymbol{x}_i)\right)$ is the empirical loss;
- $\hat{f}(\boldsymbol{x}_i)^{[k-1]}$ is the current estimate of the model;
- $\Omega(f_k(\boldsymbol{x}))$ is a penalty term that penalizes the tree complexity.

Once the best tree $f_k(\boldsymbol{x})$ is obtained, its contribution is added to the current model,

$$\hat{f}(\boldsymbol{x})^{[k]} = \hat{f}(\boldsymbol{x})^{[k-1]} + \nu f_k(\boldsymbol{x}).$$

eXtreme Gradient Boosting: details

The optimization is performed on the second order approximation of $\text{obj}(f_k(\boldsymbol{x}))$:

$$\text{obj}(f_k(\boldsymbol{x})) \approx \sum_{i=1}^{n} \left[L\left(y_i, \hat{f}(\boldsymbol{x}_i)^{[k-1]}\right) + g_i f_k(\boldsymbol{x}_i) + \tfrac{1}{2} h_i f_k^2(\boldsymbol{x}_i) \right] + \Omega(f_k(\boldsymbol{x}_i)),$$

where

- $g_i = \partial_{\hat{f}(\boldsymbol{x}_i)^{[k-1]}} L(y_i, \hat{f}(\boldsymbol{x}_i)^{[k-1]})$;
- $h_i = \partial^2_{\hat{f}(\boldsymbol{x}_i)^{[k-1]}} L(y_i, \hat{f}(\boldsymbol{x}_i)^{[k-1]})$;

The construction of the decision trees only depends on the loss function through g_i and h_i;

- it eases the computations.

eXtreme Gradient Boosting: details

The penalty term is also formulated

$$\Omega(f_k(\boldsymbol{x})) = \gamma T_k + \frac{1}{2}\lambda||w_k||^2$$

to help the computations (of the trees):

- γ is associated to the computation of the split points0,
  - penalizes the number of the tree leafs T;
- λ to that of the leaf weights,
  - penalizes the magnitude of the weights w;
  - $||\cdot||$ denotes the L_2 norm.

eXtreme Gradient Boosting: details

Other relevant features implemented in XGBoost include:

- variable subsampling;
  - only a random part of the variables are used in fitting the tree;
  - conveniently, the computations are also speeded up.
- weighted quantile sketch to speed up the choice of slit points;
- sparsity-aware split finding;

Plus some computational advances, e.g.,

- column block for parallel learning;
- cache-aware access;
- blocks for out-of-core computation,
  - block compression;
  - block sharding.

References I

Bühlmann, P. & Yu, B. (2003). Boosting with the L_2 loss: regression and classification. *Journal of the American Statistical Association* **98**, 324–339.

Chen, T. & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Seibold, H., Bernau, C., Boulesteix, A.-L. & De Bin, R. (2018). On the choice and influence of the number of boosting steps for high-dimensional linear cox-models. *Computational Statistics* **33**, 1195–1215.