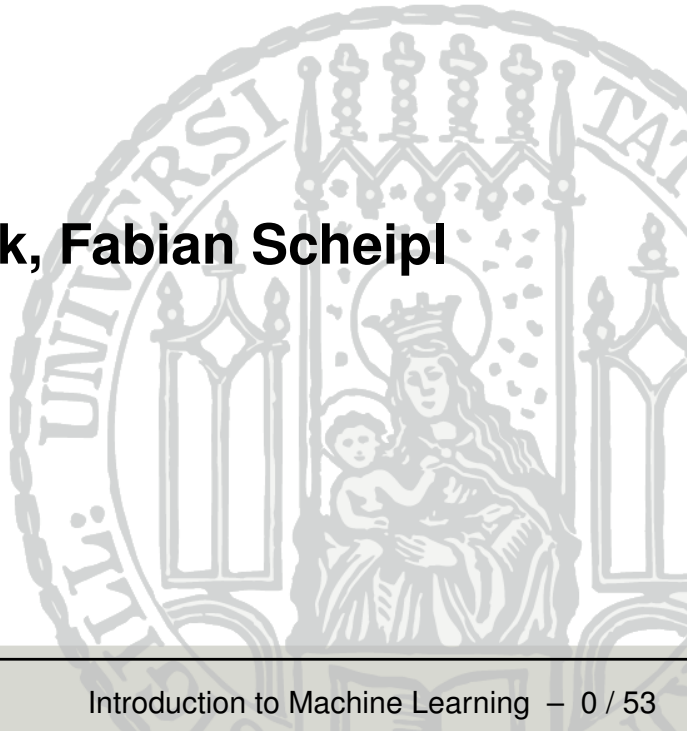
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Introduction to Machine Learning

12

Bernd Bischl, Christoph Molnar, Daniel Schalk, Fabian Scheipl

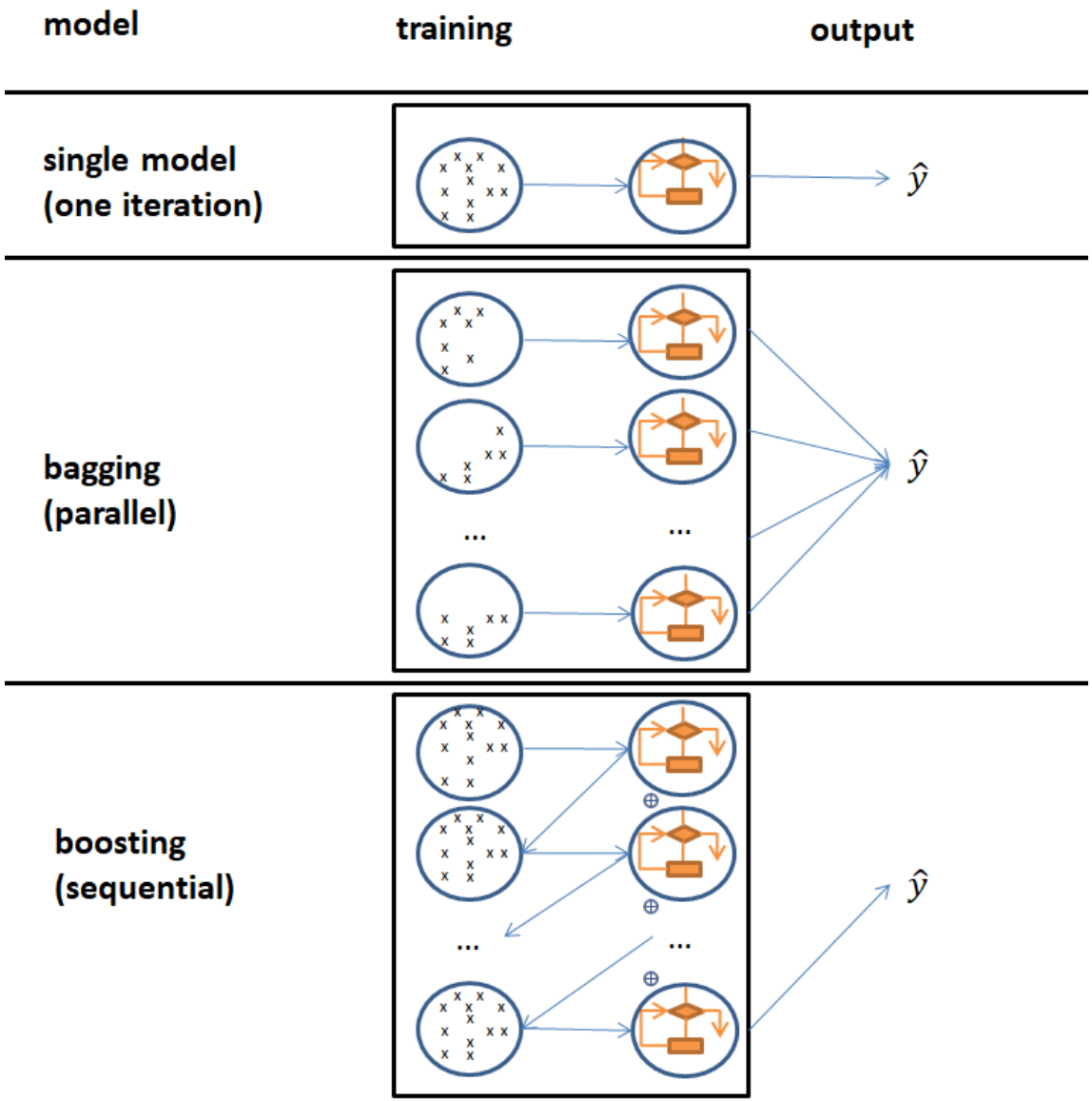
Department of Statistics – LMU Munich



INTRODUCTION TO BOOSTING

- Boosting is considered to be one of the most powerful learning ideas within the last twenty years.
- Originally designed for classification, but (especially gradient) boosting handles regression (and many others tasks) nowadays naturally.
- Homogeneous ensemble method (like bagging), but fundamentally different approach.
- **Idea:** Take a weak classifier and sequentially apply it to modified versions of the training data.
- We will begin by describing an older, simpler boosting algorithm, designed for binary classification, the popular “AdaBoost”.

BOOSTING VS. BAGGING



THE BOOSTING QUESTION

The first boosting algorithm ever was in fact no algorithm for practical purposes but the proof in a theoretical discussion:

"Does the existence of a weak learner for a certain problem imply the existence of a strong learner?" (Kearns 1988)

- *Weak learners* are defined as a prediction rule with a correct classification rate that is at least slightly better than random guessing (> 50% accuracy).
- We call a learner a *strong learner* "if there exists a polynomial-time algorithm that achieves low error with high confidence for all concepts in the class" (Schapire 1990)

In practice, it is typically easy to construct weak learners, but very difficult to get a strong one.

AdaBoost

THE BOOSTING ANSWER - ADABOOST

Any weak (base) learner can be iteratively boosted to become also a strong learner (Schapire and Freund 1990). The proof of this ground-breaking idea generated the first boosting algorithm.

- The *AdaBoost* (Adaptive Boosting) algorithm is a *boosting* method for binary classification by Freund and Schapire (1997).
- The base learner is sequentially applied to weighted training observations.
- After each base learner fit, currently misclassified observation receive a higher weight for the next iteration, so we focus more on the “harder” instances.

Leo Breiman (referring to the success of AdaBoost):
“Boosting is the best off-the-shelf classifier in the world.”

THE BOOSTING ANSWER - ADABOOST

- Assume target variable y encoded as $\{-1, 1\}$, and weak base learner (e.g. tree stumps) from a hypothesis space $\mathcal{B}$.
- Base learner models $b^{[m]}$ are binary classifiers that map to $\mathcal{Y} = \{-1, +1\}$. We might sometimes write $b(x, \theta^{[m]})$, instead.
- Predictions from all base models $b^{[m]}$ are combined to form:

$$f(\mathbf{x}) = \sum_{m=1}^{M} \beta^{[m]} b^{[m]}(x)$$

- Weights $\beta^{[m]}$ are computed by the boosting algorithm. Their effect is to give higher values to base learners with higher predictive accuracy.
- Number of iterations M main tuning parameter.
- The discrete prediction function is $h(x) = \text{sign}(f(\mathbf{x})) \in \{-1, 1\}$.

THE BOOSTING ANSWER - ADABOOST

AdaBoost

1: Initialize observation weights: $w^{[1](i)} = \frac{1}{n} \quad \forall i \in \{1, \ldots, n\}$

2: **for** $m = 1 \to M$ **do**

3: Fit classifier to training data with weights $w^{[m]}$ and get $\hat{b}^{[m]}$

4: Calculate weighted in-sample misclassification rate

$$\text{err}^{[m]} = \frac{\sum_{i=1}^{n} w^{[m](i)} \cdot \left[y^{(i)} \neq \hat{b}^{[m]}(\mathbf{x}^{(i)})\right]}{\sum_{i=1}^{n} w^{[m](i)}}$$

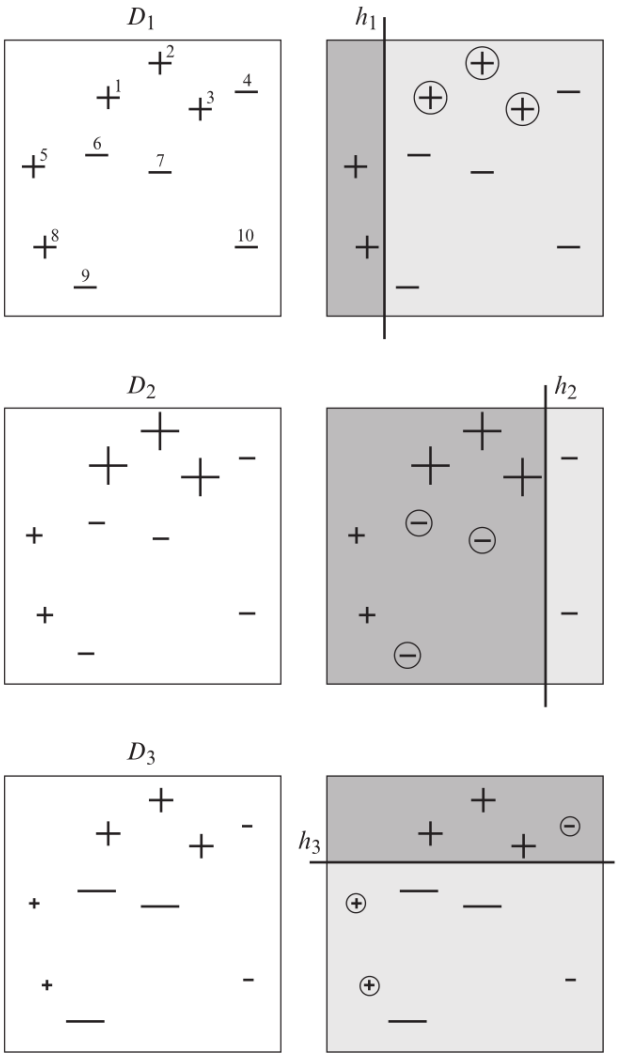
5: Compute: $\hat{\beta}^{[m]} = \frac{1}{2} \log\left(\frac{1-\text{err}^{[m]}}{\text{err}^{[m]}}\right)$

6: Set: $w^{[m+1](i)} = w^{[m](i)} \cdot \exp\left(\hat{\beta}^{[m]} \cdot \left[y^{(i)} \neq \hat{b}^{[m]}(\mathbf{x}^{(i)})\right]\right)$

7: **end for**

8: Output: $\hat{f}(\mathbf{x}) = \sum_{m=1}^{M} \hat{\beta}^{[m]} \hat{b}^{[m]}(x)$

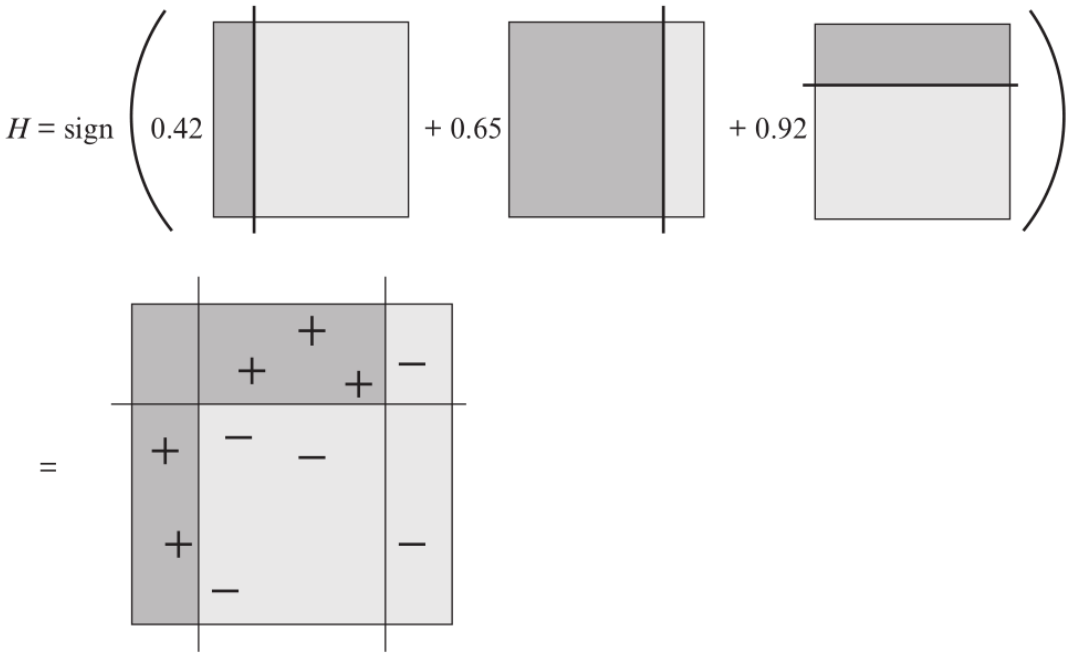
ADABOOST ILLUSTRATION



Schapire, Boosting, 2012.

- $n = 10$ observations and $M = 3$ iterations, tree stumps as base learners
- The label of every observation is represented by $+$ or $-$
- The size of the label represents the weight of an observation in the corresponding iteration
- Dark grey area: base model in iteration m predicts $+$
- Light grey area: base model in iteration m predicts $-$

ADABOOST ILLUSTRATION



Schapire, Boosting, 2012.

The three base models are combined into one classifier.
All observations are correctly classified.

ADABOOST WEIGHTS

The base-learner weights of AdaBoost can be treated as measure how much a single base-learner was able to learn:

- We expect a weak learner to be slightly better than random guessing: $\text{err}^{[m]} \leq 0.5$
  $\rightarrow$ First base-learner $\text{err}^{[1]} = 0.3$
- The ratio $(1 - \text{err}^{[m]})/\text{err}^{[m]}$ used to calculate the weights is therefore the proportion of correct vs. misclassified examples
  $\rightarrow$ First base-learner $0.7/0.3 \approx 2.33$
- The logarithm scales the ratio to a more intuitive weighting scheme:
  - $0.5 \log((1 - \text{err}^{[m]})/\text{err}^{[m]}) = 0 \Leftrightarrow$ base-learner m is not able to learn anything $\rightarrow$ $\text{err}^{[m]} = 0.5$
  - $0.5 \log((1 - \text{err}^{[m]})/\text{err}^{[m]}) > 0 \Leftrightarrow$ base-learner m is able to learn something $\rightarrow$ $\text{err}^{[m]} < 0.5$

  $\rightarrow$ Weight of the first base-learner: $0.5 \log(2.33) \approx 0.42$

ADABOOST WEIGHTS

The next step is to update the observation weights w_i for for each observation i.

- For iteration $m + 1$, this is done by

$$w^{[m+1](i)} = w^{[m](i)} \cdot \exp\left(\hat{\beta}^{[m]} \cdot \left[y^{(i)} \neq \hat{b}^{[m]}(\mathbf{x}^{(i)})\right]\right)$$

- If the base learner predicts the i-th observation correctly, $\exp\left(\hat{\beta}^{[m]} \cdot \left[y^{(i)} \neq \hat{b}^{[m]}(\mathbf{x}^{(i)})\right]\right)$ reduces to $\exp(0) = 1$, which means that the weight for observation i stays unchanged since $w^{[m+1](i)} = w^{[m](i)} \cdot 1 = w^{[m](i)}$.
- Considering the example, we have 7 correctly observations for which the initial observation weights $1/n = 0.1$ are kept untouched.

ADABOOST WEIGHTS

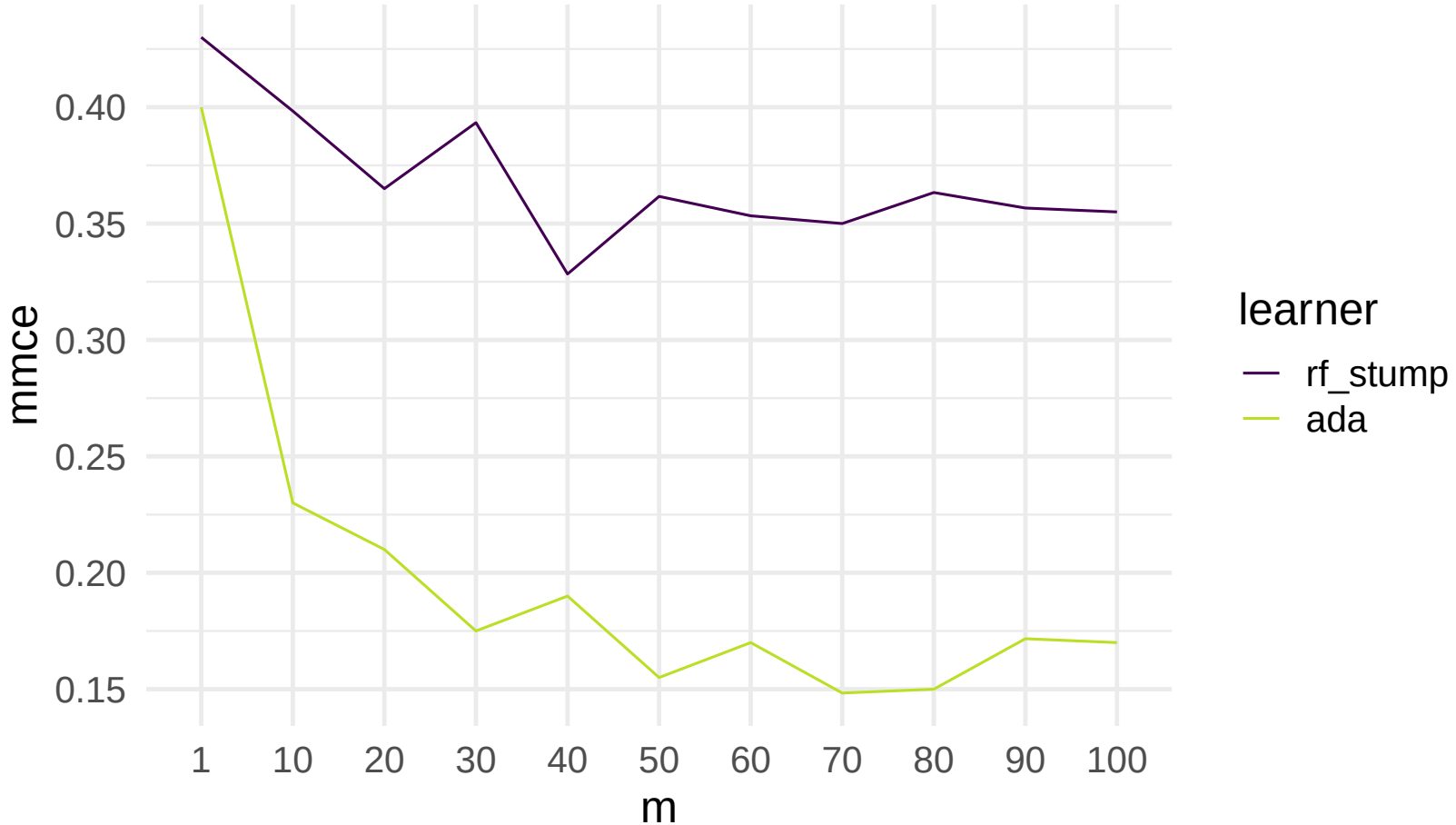
- The 3 wrongly classified observations get new observation weights

$$
\begin{aligned}
w^{[2](i)} &= w^{[1](i)} \cdot \exp\left(\hat{\beta}^{[1]} \cdot \left[y^{(i)} \neq \hat{b}^{[1]}(\mathbf{x}^{(i)})\right]\right) \\
&= \frac{1}{10} \cdot \exp\left(0.42 \cdot 1\right) \\
&\approx 0.11
\end{aligned}
$$

- The resulting new weights for the missclassified observations are now slightly greater compared to the initial weights of 0.1.
- This forces the next base learner $b^{[2]}(\mathbf{x}^{(i)})$ to concentrate more on those.

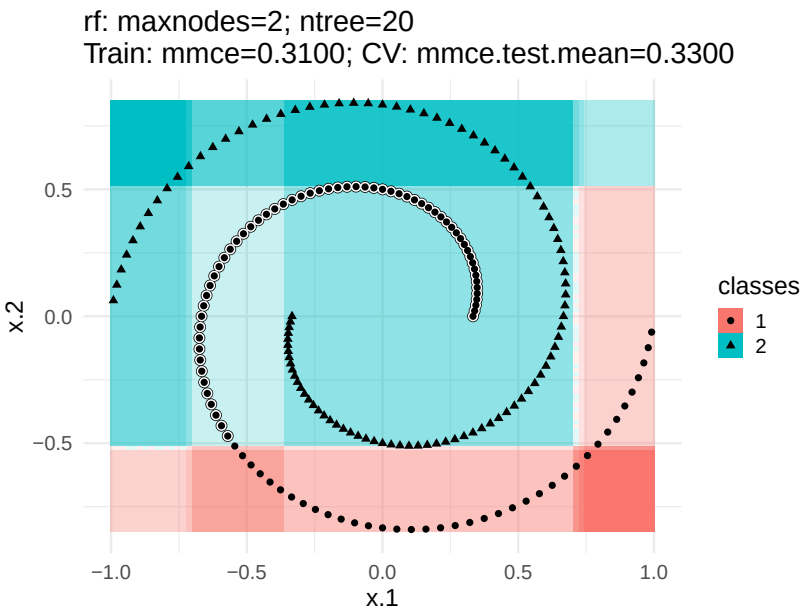
EXAMPLE: BAGGING VS BOOSTING

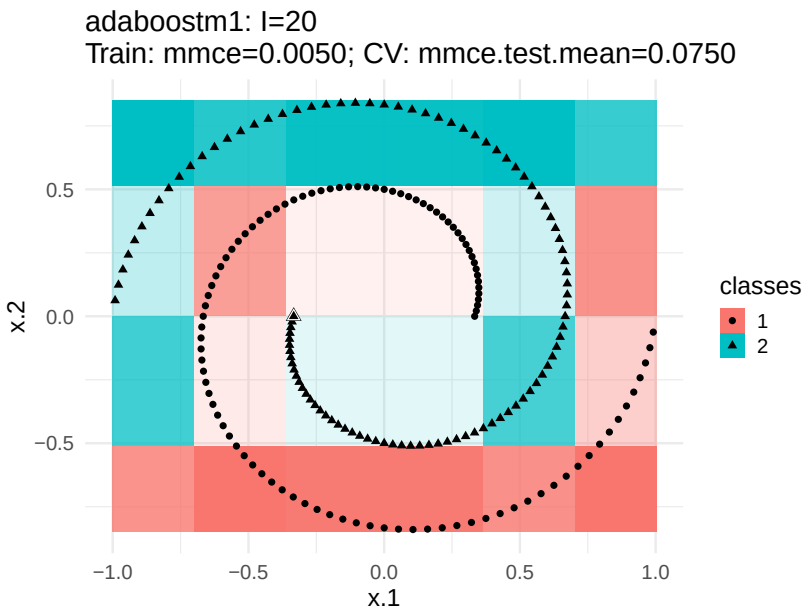
Random Forest versus AdaBoost (both with stumps) on Spirals data from mlbench ($n = 200$, sd$= 0$). Performance (mmce) is measured with 3×10 repeated CV.



EXAMPLE: BAGGING VS BOOSTING

- We see that, weak learners are not as powerful in combination with bagging compared with boosting.
- This is mainly caused by the fact that bagging only performs variance reduction and that tree stumps are very stable





OVERFITTING BEHAVIOR

A long-lasting discussion in the context of AdaBoost is its overfitting behavior.
The main instrument to avoid overfitting is the stopping iteration M:

- High values of M lead to complex solutions. Overfitting?
- Small values of M lead to simple solutions. Underfitting?

Although eventually it will overfit, AdaBoost in general shows a rather slow overfitting behavior.

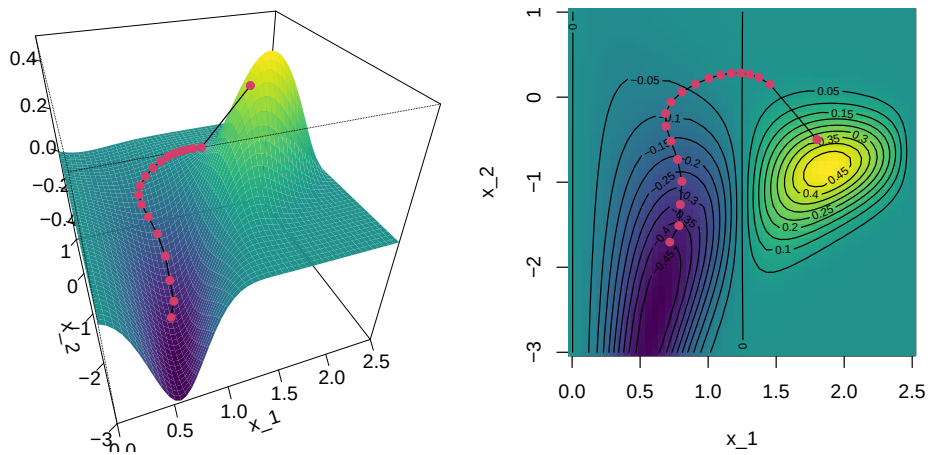
Gradient Boosting

GRADIENT DESCENT

Let $\mathcal{R}(\theta)$ be (just for the next few slides) an arbitrary, differentiable, unconstrained objective function, which we want to minimize. The gradient $\nabla\mathcal{R}(\theta)$ is the direction of the steepest ascent, $-\nabla\mathcal{R}(\theta)$ of **steepest descent**.

For an intermediate solution $\theta^{[k]}$ during minimization, we can iteratively improve by updating

$$\theta^{[k+1]} = \theta^{[k]} - \beta\nabla\mathcal{R}(\theta^{[k]}) \qquad \text{for } 0 < \beta \leq c\left(\theta^{[k]}\right)$$



FORWARD STAGEWISE ADDITIVE MODELING

Assume a regression problem for now (as this is simpler to explain); and assume a space of base learners $\mathcal{B}$.

We want to learn an additive model:

$$f(\mathbf{x}) = \sum_{m=1}^{M} \beta^{[m]} b(x, \theta^{[m]})$$

Hence, we minimize the empirical risk:

$$\mathcal{R}_{\text{emp}}(f) = \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right) = \sum_{i=1}^{n} L\left(y^{(i)}, \sum_{m=1}^{M} \beta^{[m]} b\left(\mathbf{x}^{(i)}, \theta^{[m]}\right)\right)$$

FORWARD STAGEWISE ADDITIVE MODELING

Because of the additive structure, it is difficult to jointly minimize $\mathcal{R}_{\text{emp}}(f)$ w.r.t. $\left(\left(\beta^{[1]}, \theta^{[1]}\right), \ldots, \left(\beta^{[M]}, \theta^{[M]}\right)\right)$, which is a very high-dimensional parameter space.

Moreover, considering trees as base learners is worse, as we would now have to grow M trees in parallel so they work optimally together as an ensemble.

Finally, stagewise additive modeling has nice properties, which we want to make use of, e.g. for regularization, early stopping, ...

FORWARD STAGEWISE ADDITIVE MODELING

Hence, we add additive components in a greedy fashion by sequentially minimizing the risk only w.r.t. the next additive component:

$$\min_{\beta,\theta} \sum_{i=1}^{n} L\left(y^{(i)}, \hat{f}^{[m-1]}\left(\mathbf{x}^{(i)}\right) + \beta b\left(\mathbf{x}^{(i)}, \theta\right)\right)$$

Doing this iteratively is called **forward stagewise additive modeling**

Algorithm 1 Forward Stagewise Additive Modeling.

1: Initialize $\hat{f}^{[0]}(x)$
2: **for** $m = 1 \to M$ **do**
3: $\quad (\hat{\beta}^{[m]}, \hat{\theta}^{[m]}) = \arg\min_{\beta,\theta} \sum_{i=1}^{n} L\left(y^{(i)}, \hat{f}^{[m-1]}\left(\mathbf{x}^{(i)}\right) + \beta b\left(\mathbf{x}^{(i)}, \theta\right)\right)$
4: $\quad$ Update $\hat{f}^{[m]}(x) \leftarrow \hat{f}^{[m-1]}(x) + \hat{\beta}^{[m]} b\left(x, \hat{\theta}^{[m]}\right)$
5: **end for**

GRADIENT BOOSTING

The algorithm we just introduced isn't really an algorithm, but rather an abstract principle. We need to find the new additive component $b\left(x, \theta^{[m]}\right)$ and its weight coefficient $\beta^{[m]}$ in each iteration m. This can be done by gradient descent, but in function space.

Thought experiment: Consider a completely non-parametric model f, where we can arbitrarily define its predictions on every point of the training data $\mathbf{x}^{(i)}$. So we basically specify f as a discrete, finite vector.

$$\left(f\left(x^{(1)}\right), \ldots, f\left(x^{(n)}\right)\right)^T$$

This implies n parameters $f\left(\mathbf{x}^{(i)}\right)$ (and the model would provide no generalization...).
Furthermore, we assume our loss function L to be differentiable.

GRADIENT BOOSTING

Now we want to minimize the risk of such a model with gradient descent (yes, this makes no sense, suspend all doubts for a few seconds).
So, we calculate the gradient at a point of the parameter space, that is the derivative with respect to each component of the parameter vector.

$$\frac{\partial \mathcal{R}_{\text{emp}}}{\partial f\left(\mathbf{x}^{(i)}\right)} = \frac{\partial \sum_j L(y^{(j)}, f(x^{(j)}))}{\partial f\left(\mathbf{x}^{(i)}\right)} = \frac{\partial L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)}{\partial f\left(\mathbf{x}^{(i)}\right)}$$

The gradient descent update for each vector component of f is:

$$f\left(\mathbf{x}^{(i)}\right) \leftarrow f\left(\mathbf{x}^{(i)}\right) - \beta \frac{\partial L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)}{\partial f\left(\mathbf{x}^{(i)}\right)}$$

This tells us how we could “nudge” our whole function f in the direction of the data to reduce its empirical risk.

GRADIENT BOOSTING

Combining this with the iterative additive procedure of "forward stagewise modelling", we are at the spot $f^{[m-1]}$ during minimization. At this point, we now calculate the direction of the negative gradient:

$$r^{[m](i)} = - \left[\frac{\partial L \left(y^{(i)}, f \left(\mathbf{x}^{(i)} \right) \right)}{\partial f(\mathbf{x}^{(i)})} \right]_{f = f^{[m-1]}}$$

The pseudo residuals $r^{[m](i)}$ match the usual residuals for the squared loss:

$$- \frac{\partial L(y, f(\mathbf{x}))}{\partial f(\mathbf{x})} = - \frac{\partial 0.5 (y - f(\mathbf{x}))^2}{\partial f(\mathbf{x})} = y - f(\mathbf{x})$$

GRADIENT BOOSTING

What is the point in doing all this? A model parameterized in this way is senseless, as it is just memorizing the instances of the training data...?

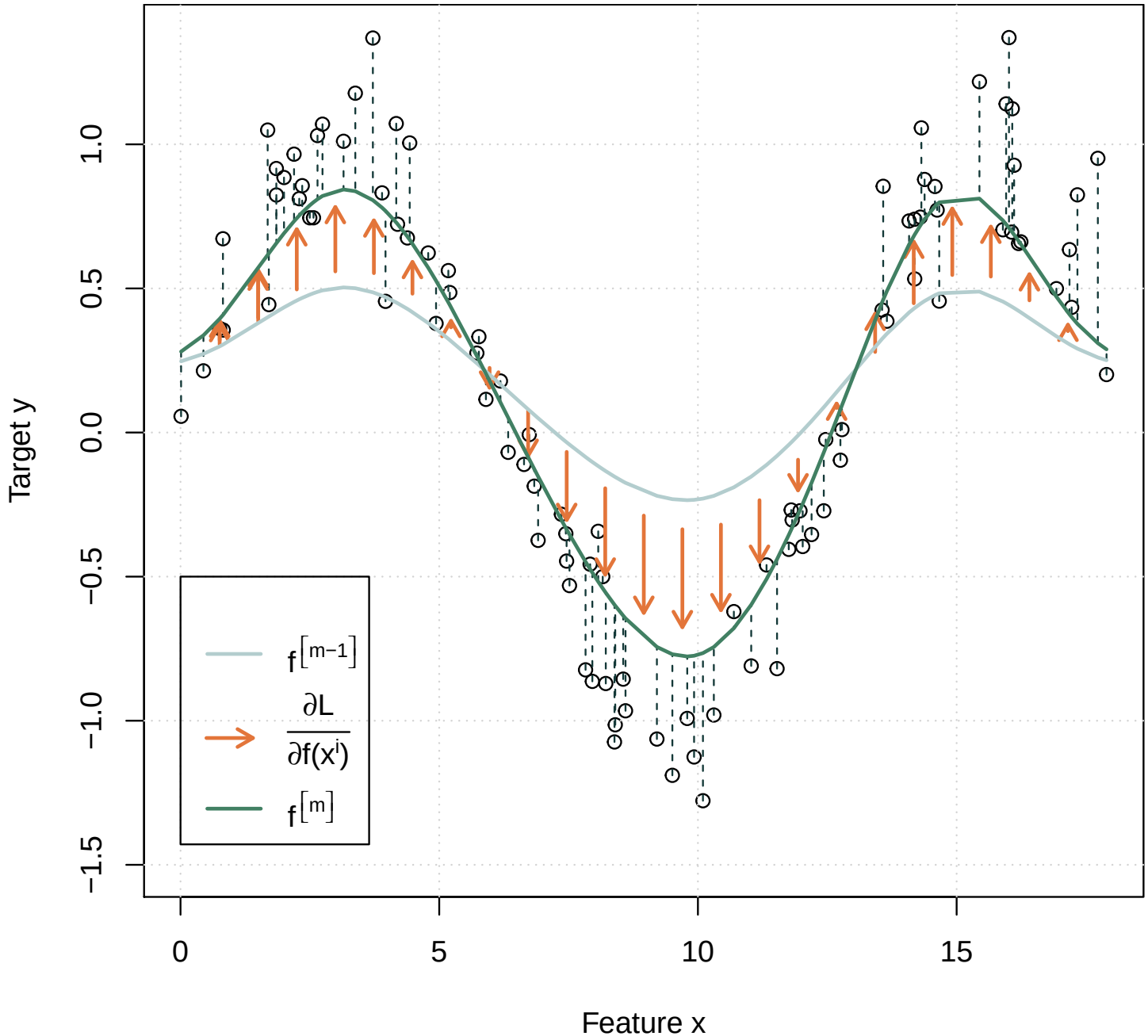
So, we restrict our additive components to $b\left(x, \theta^{[m]}\right) \in \mathcal{B}$.
The pseudo-residuals are calculated exactly as stated above, then we fit a regression model $b(x, \theta^{[m]})$ to them:

$$\hat{\theta}^{[m]} = \arg\min_{\theta} \sum_{i=1}^{n} (r^{[m](i)} - b(\mathbf{x}^{(i)}, \theta))^2$$

So, evaluated on the training data, our $b(x, \theta^{[m]})$ corresponds as closely as possible to the negative loss function gradient and generalizes to the whole space.

In a nutshell: One boosting iteration is exactly one approximated gradient step in function space, which minimizes the empirical risk as much as possible.

GRADIENT BOOSTING



GRADIENT BOOSTING ALGORITHM

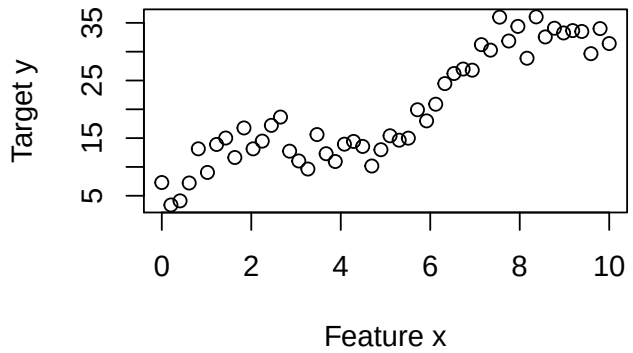
Algorithm 2 Gradient Boosting Algorithm.

1: Initialize $\hat{f}^{[0]}(x) = \arg\min_\theta \sum_{i=1}^{n} L(y^{(i)}, b(\mathbf{x}^{(i)}, \theta))$
2: **for** $m = 1 \to M$ **do**
3: For all i: $r^{[m](i)} = -\left[\frac{\partial L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)}{\partial f\left(\mathbf{x}^{(i)}\right)}\right]_{f=\hat{f}^{[m-1]}}$
4: Fit a regression base learner to the pseudo-residuals $r^{[m](i)}$:
5: $\hat{\theta}^{[m]} = \arg\min_\theta \sum_{i=1}^{n} (r^{[m](i)} - b(\mathbf{x}^{(i)}, \theta))^2$
6: Line search: $\hat{\beta}^{[m]} = \arg\min_\beta \sum_{i=1}^{n} L(y^{(i)}, f^{[m-1]}(x) + \beta b(x, \hat{\theta}^{[m]}))$
7: Update $\hat{f}^{[m]}(x) = \hat{f}^{[m-1]}(x) + \hat{\beta}^{[m]} b(x, \hat{\theta}^{[m]})$
8: **end for**
9: Output $\hat{f}(x) = \hat{f}^{[M]}(x)$

Note that we also initialize the model in a loss-optimal manner.

GRADIENT BOOSTING ILLUSTRATION - 1

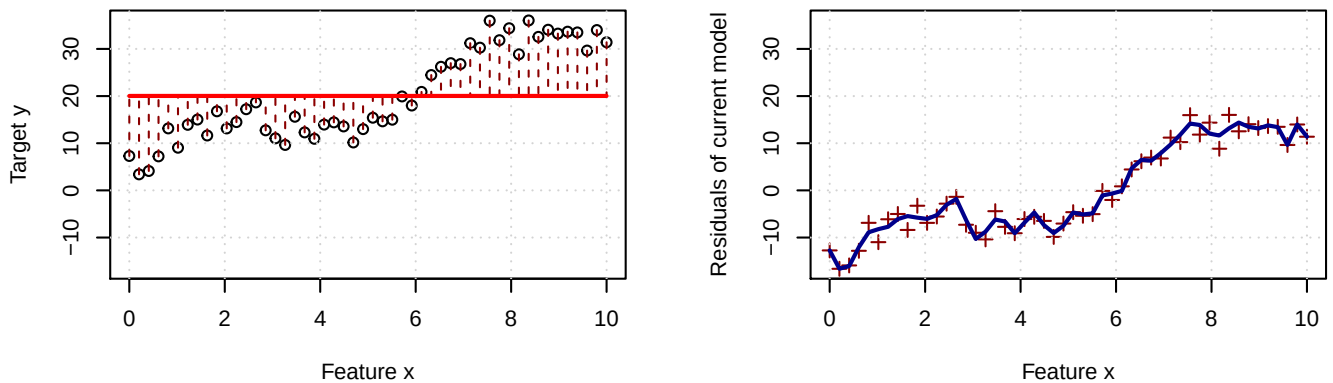
Assume one feature x and a target y.



1. We start with the simplest model, the optimal constant w.r.t. the L_2 loss (mean of the target variable).
2. To improve the model we calculate the pointwise residuals on the training data $y^{(i)} - f(\mathbf{x}^{(i)})$ and fit a GAM fitted on the residuals.
3. The GAM fitted on the residuals is then multiplied by a learning rate of 0.2 and added to the previous model.
4. This procedure is repeated multiple times.

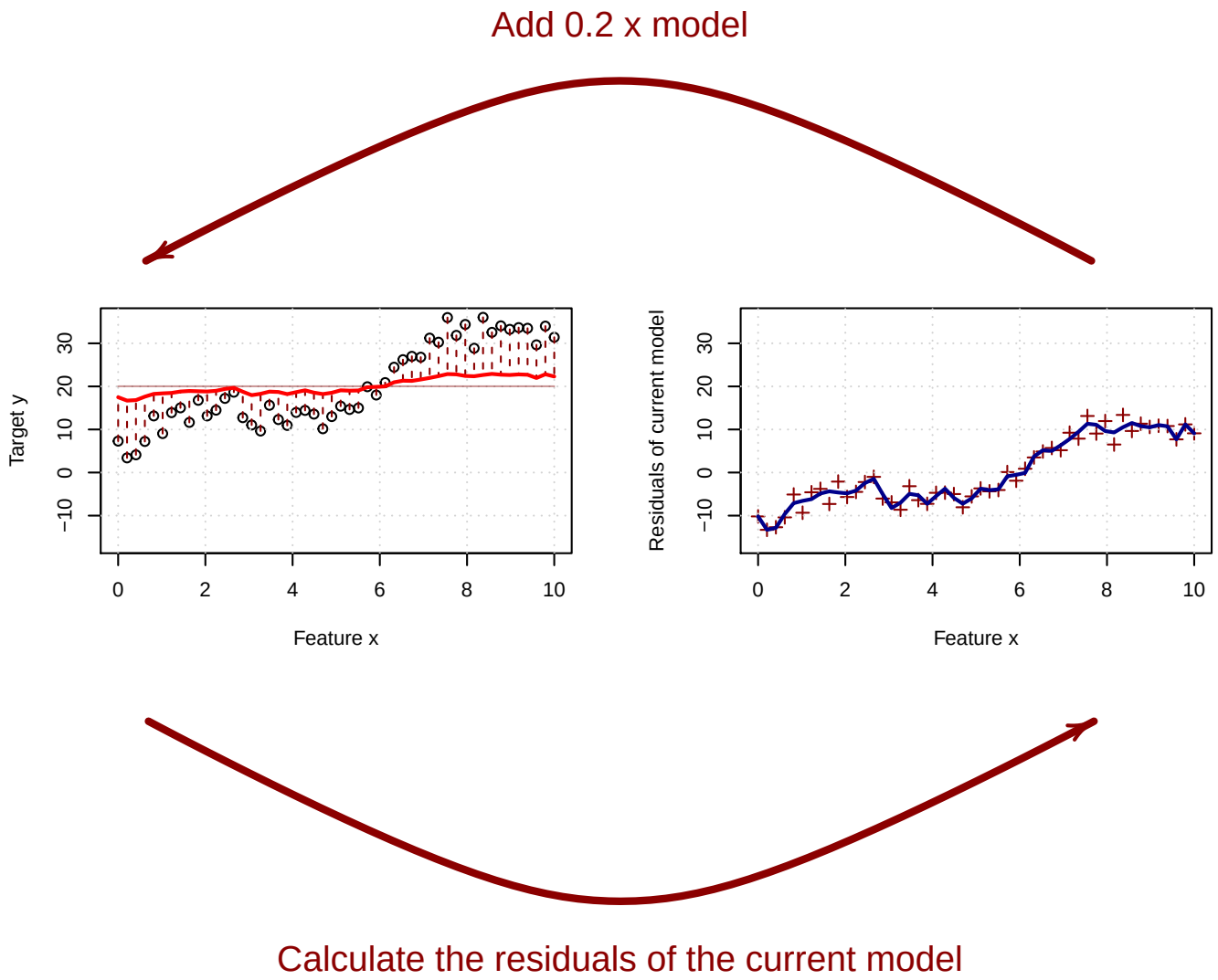
GRADIENT BOOSTING ILLUSTRATION - 1

Add 0.2 x model



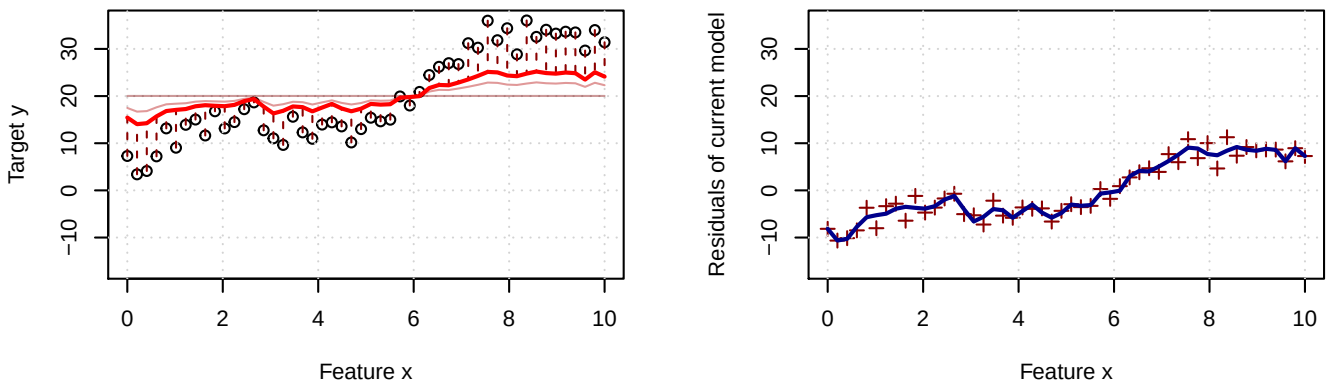
Calculate the residuals of the current model

GRADIENT BOOSTING ILLUSTRATION - 1



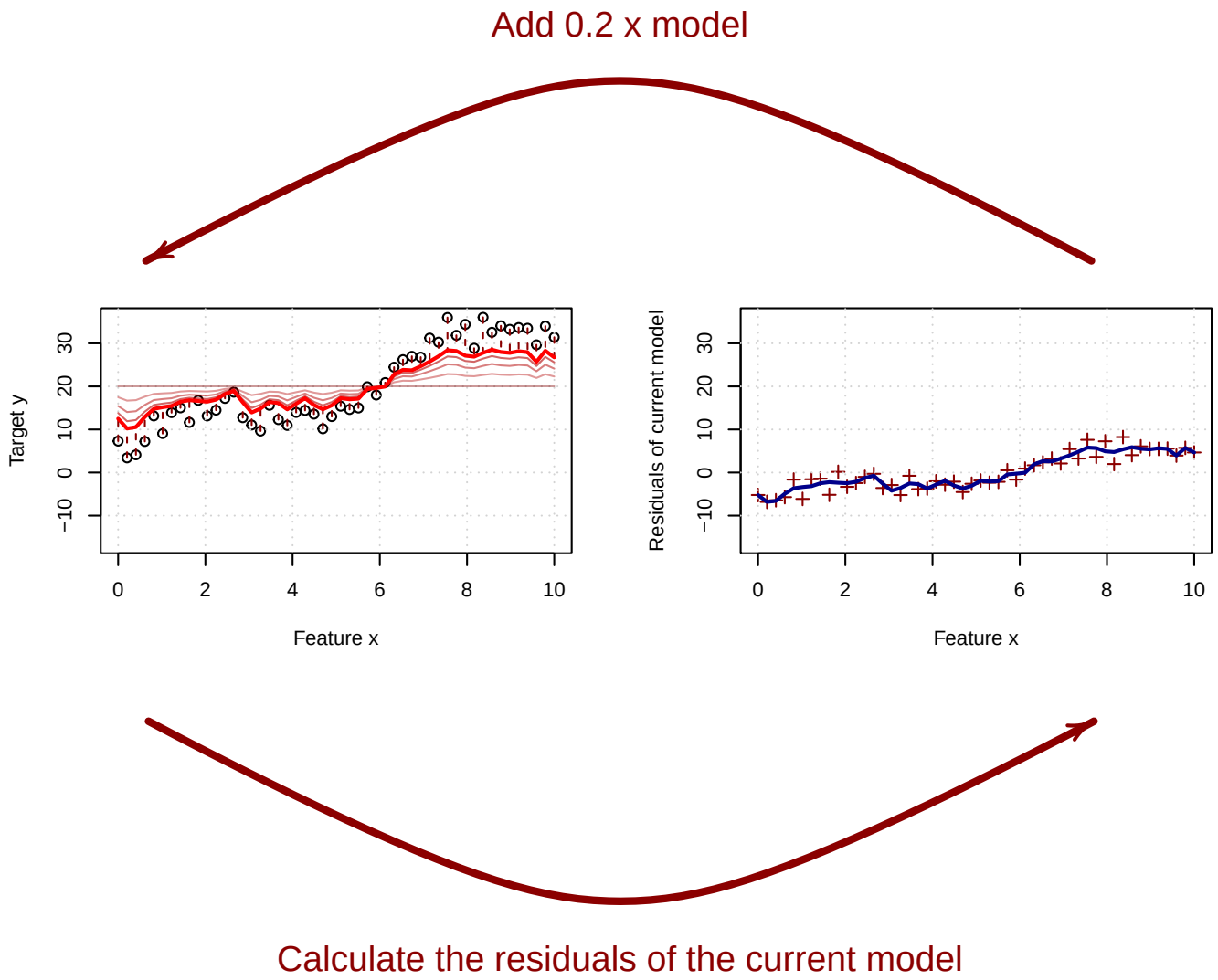
GRADIENT BOOSTING ILLUSTRATION - 1

Add 0.2 x model



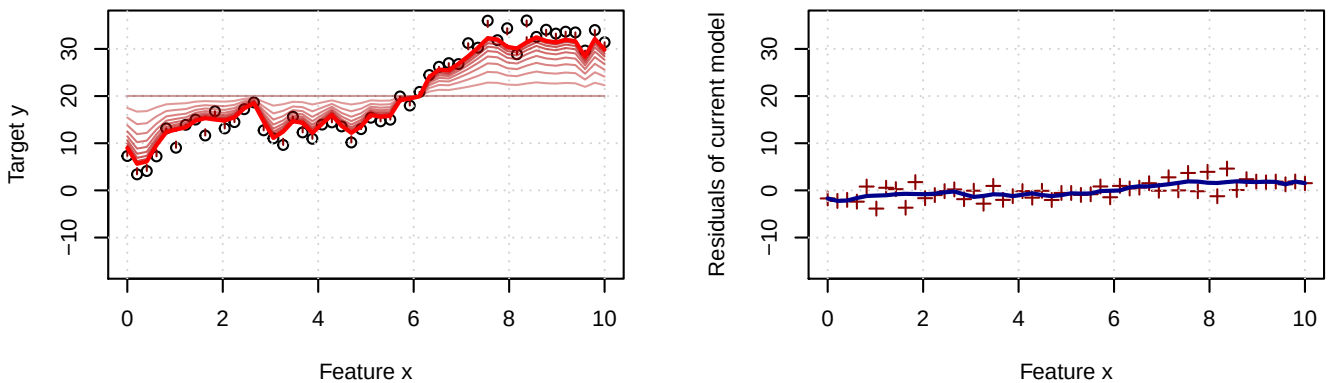
Calculate the residuals of the current model

GRADIENT BOOSTING ILLUSTRATION - 1



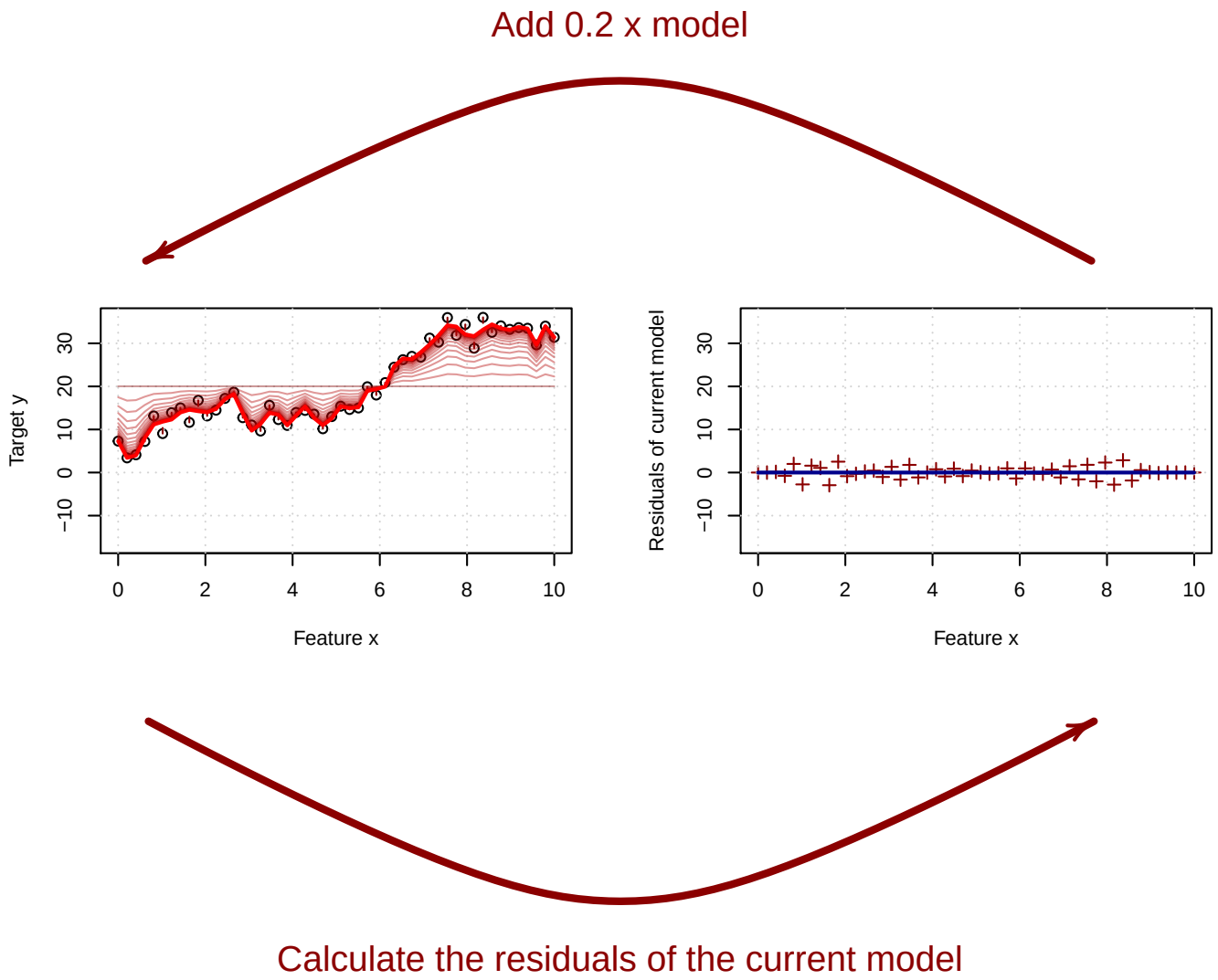
GRADIENT BOOSTING ILLUSTRATION - 1

Add 0.2 x model



Calculate the residuals of the current model

GRADIENT BOOSTING ILLUSTRATION - 1

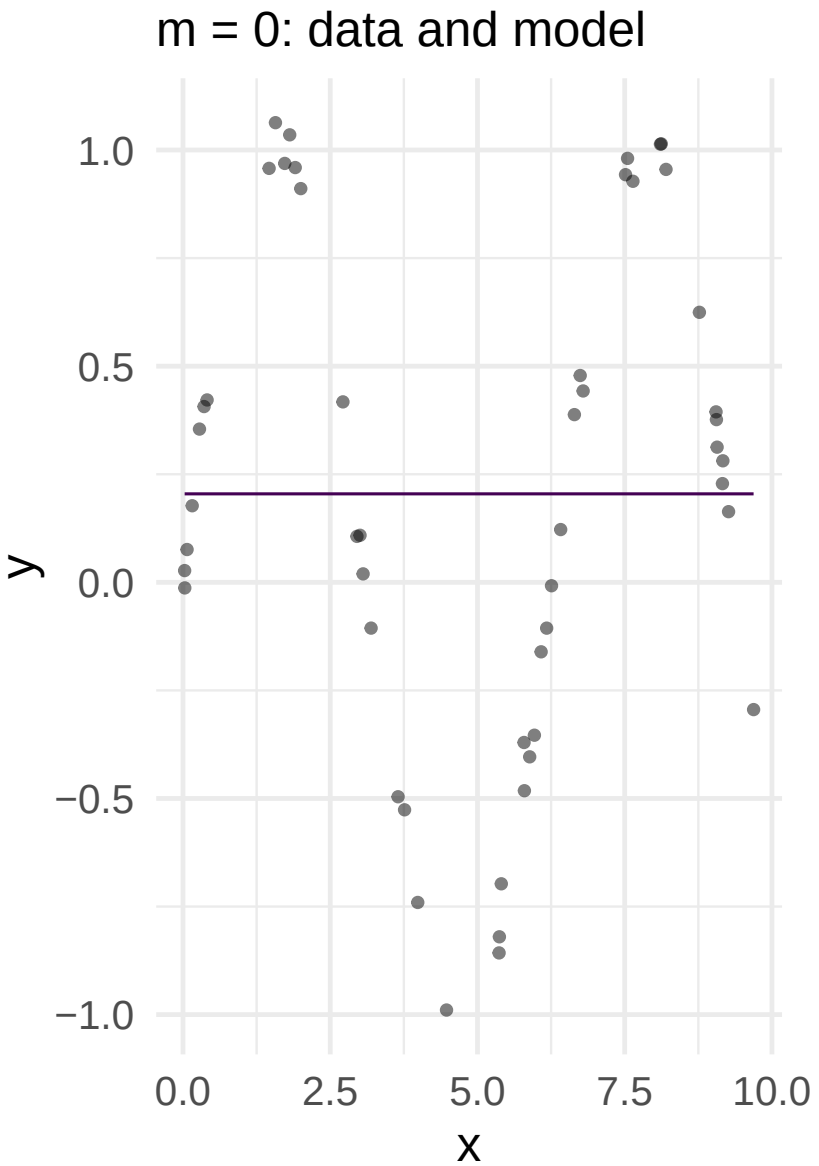


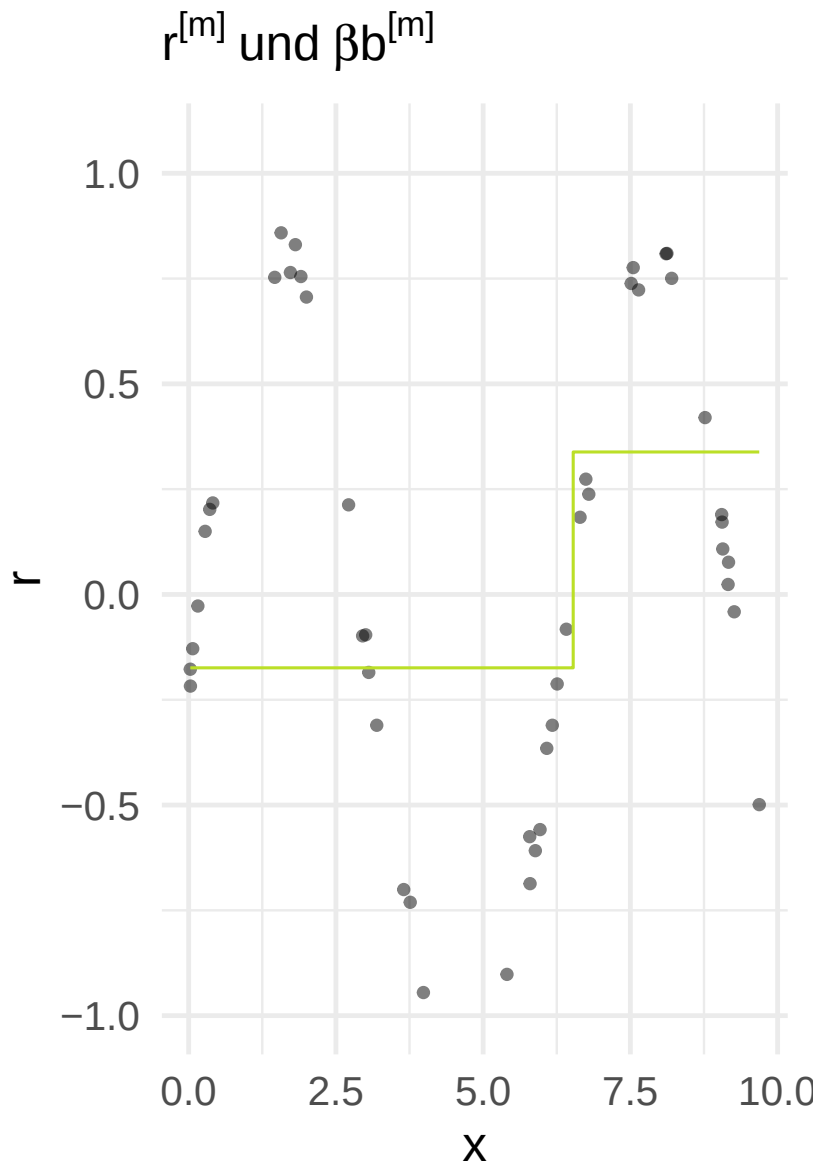
GRADIENT BOOSTING ILLUSTRATION - 2

We will consider a regression problem on the following slides:

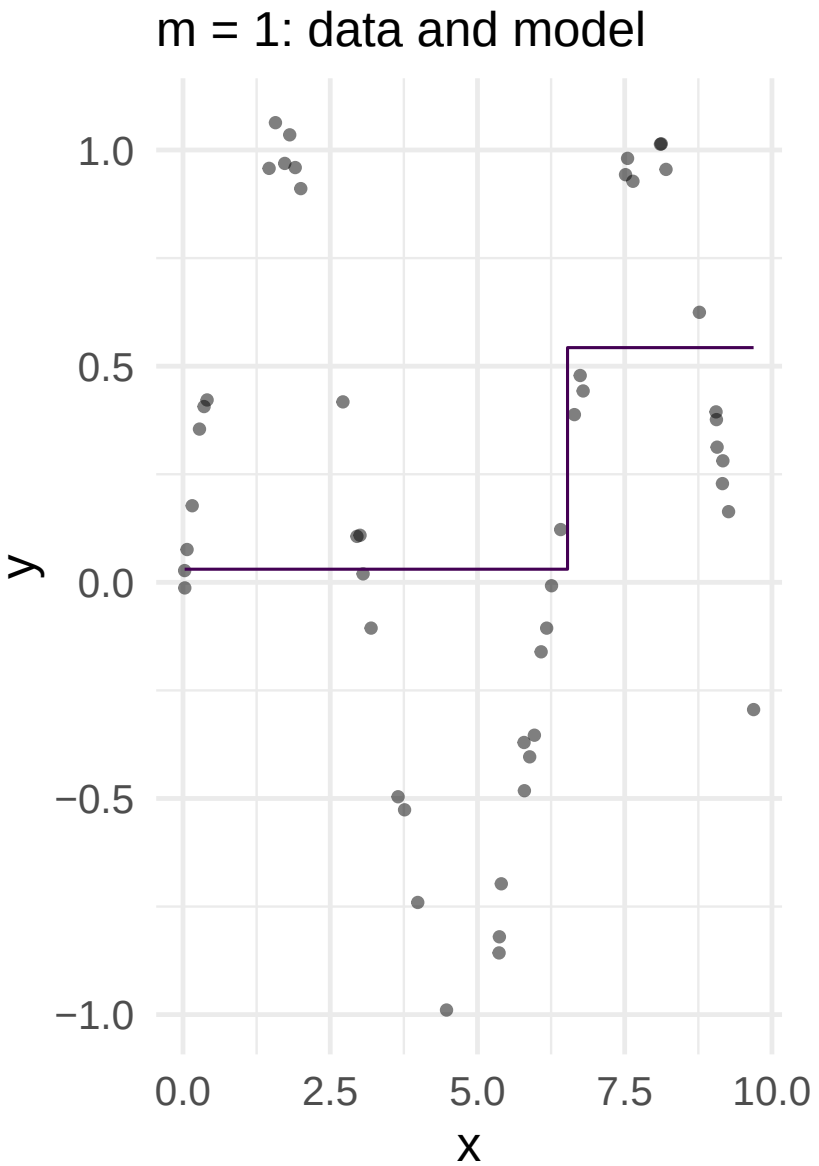
- We draw points from a sine-function with some additional noise.
- We use squared loss for L.
- Our base learner are tree stumps.
- The left plot shows the additive model learnt so far with the data, the right plot shows the residuals to which we fit the next base learner.
- The plot is generated by a self-implemented version of boosting, where the step size is obtained by line search.

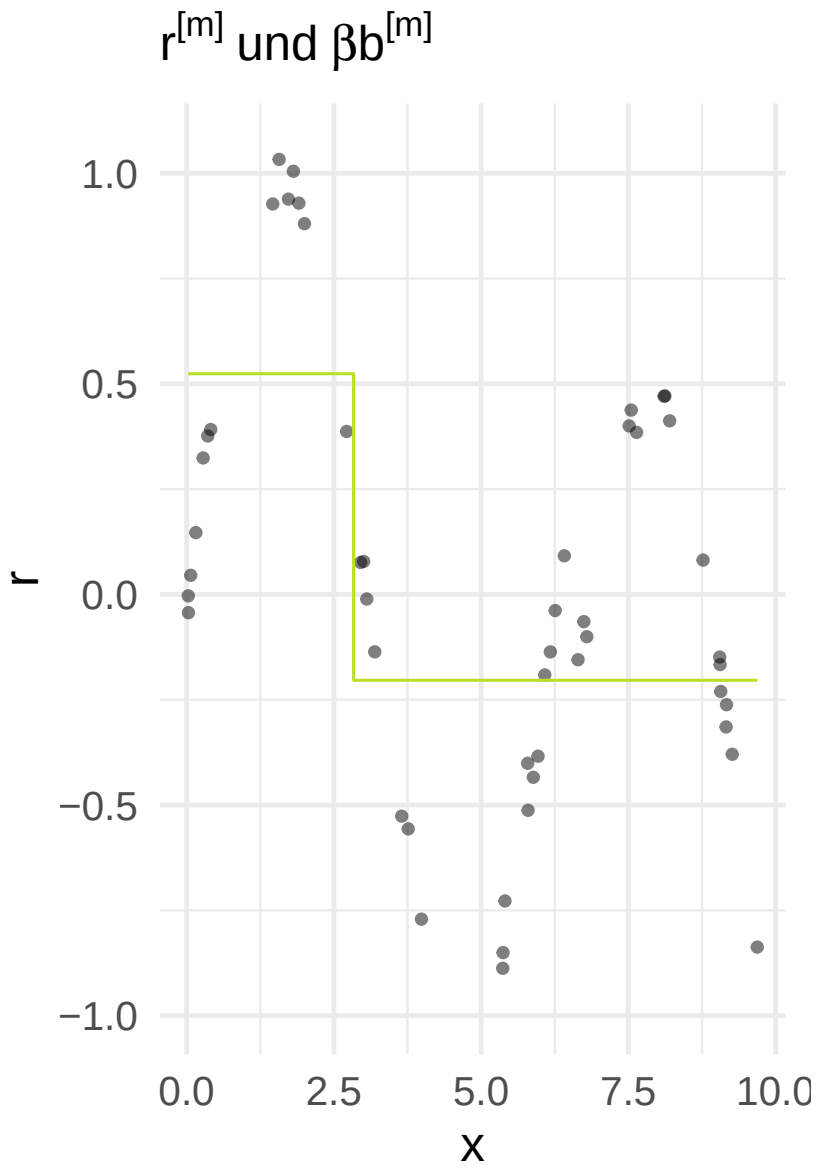
GRADIENT BOOSTING ILLUSTRATION - 2



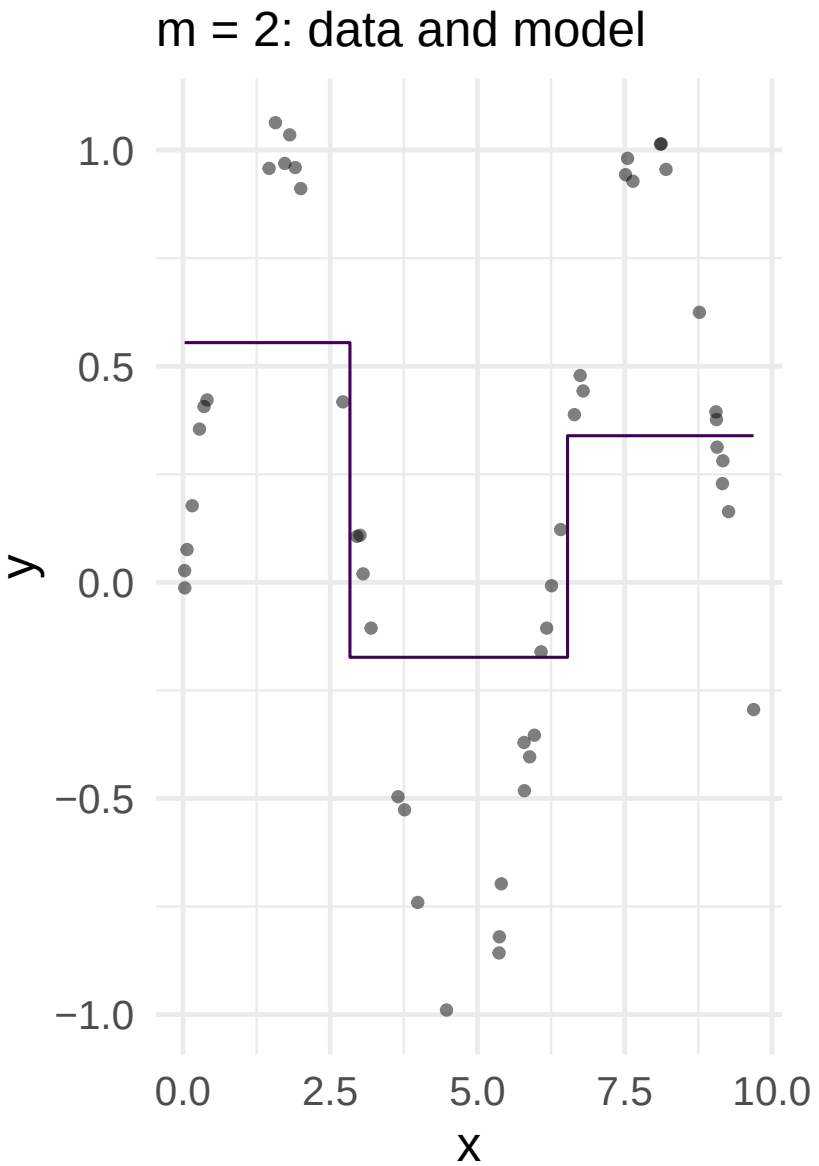


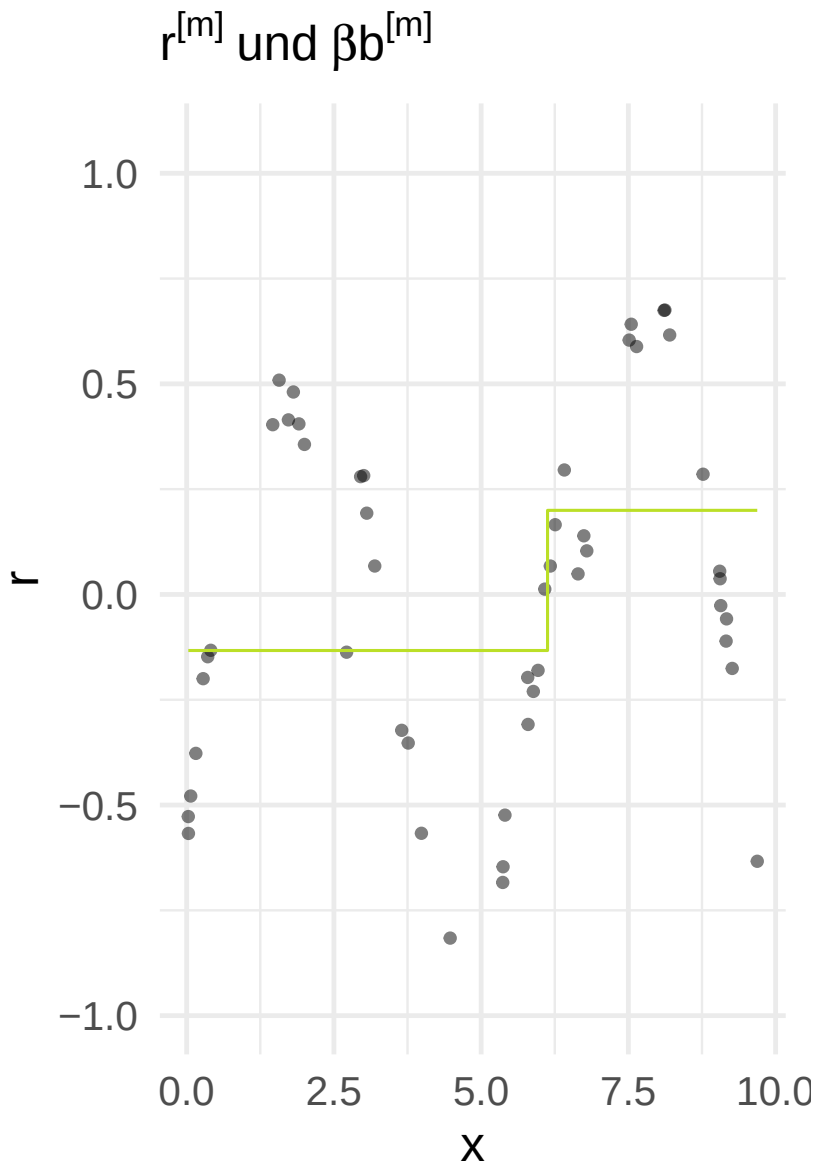
GRADIENT BOOSTING ILLUSTRATION - 2



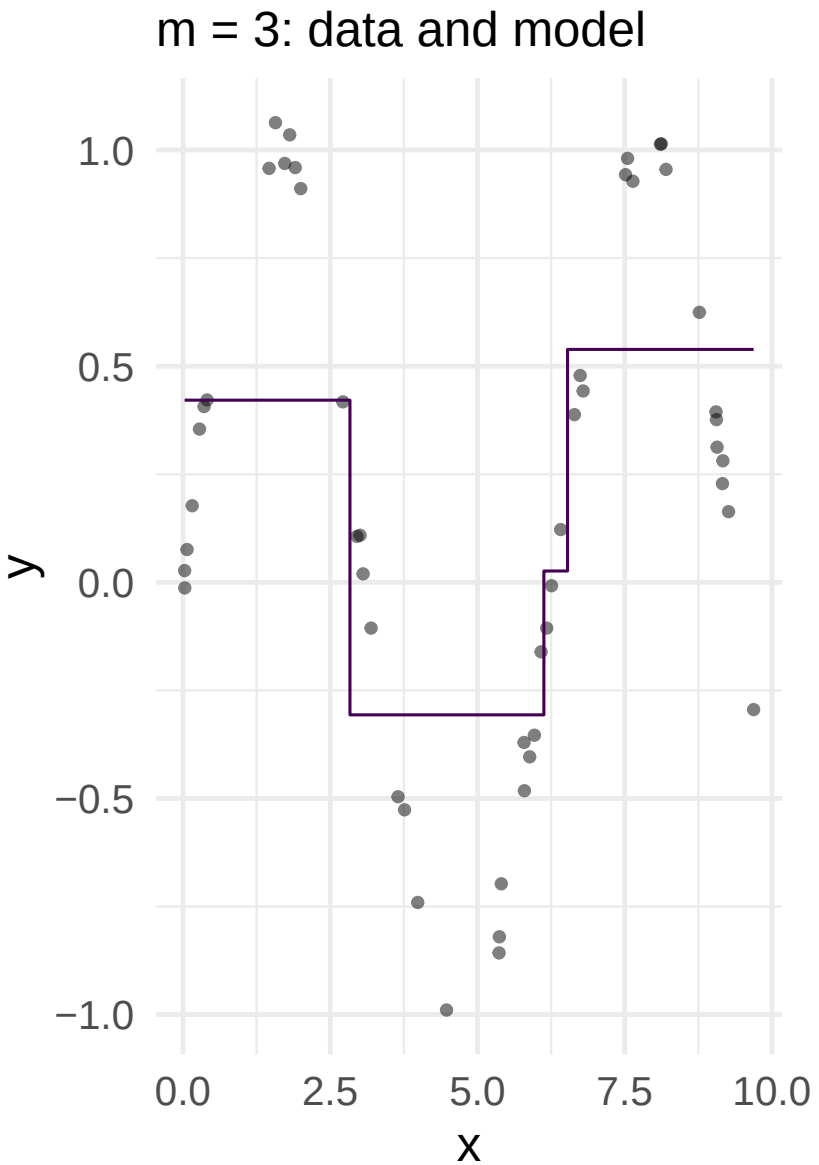


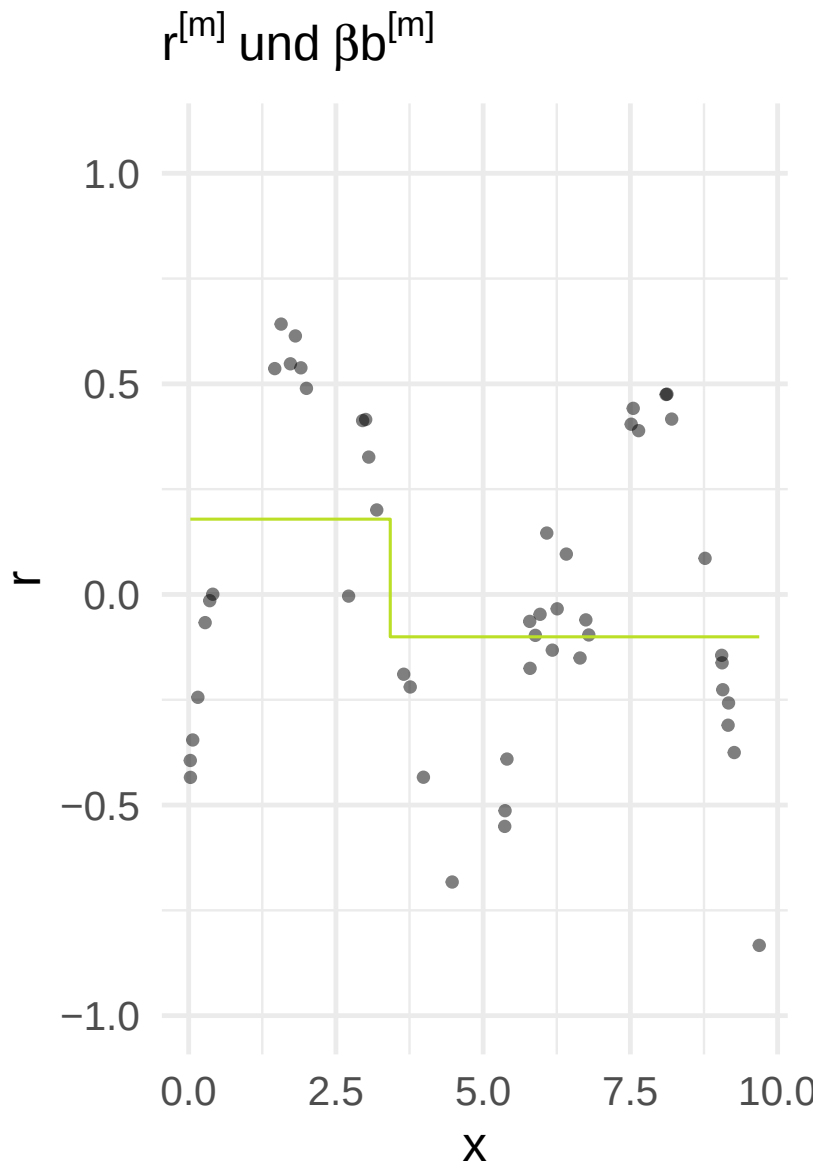
GRADIENT BOOSTING ILLUSTRATION - 2



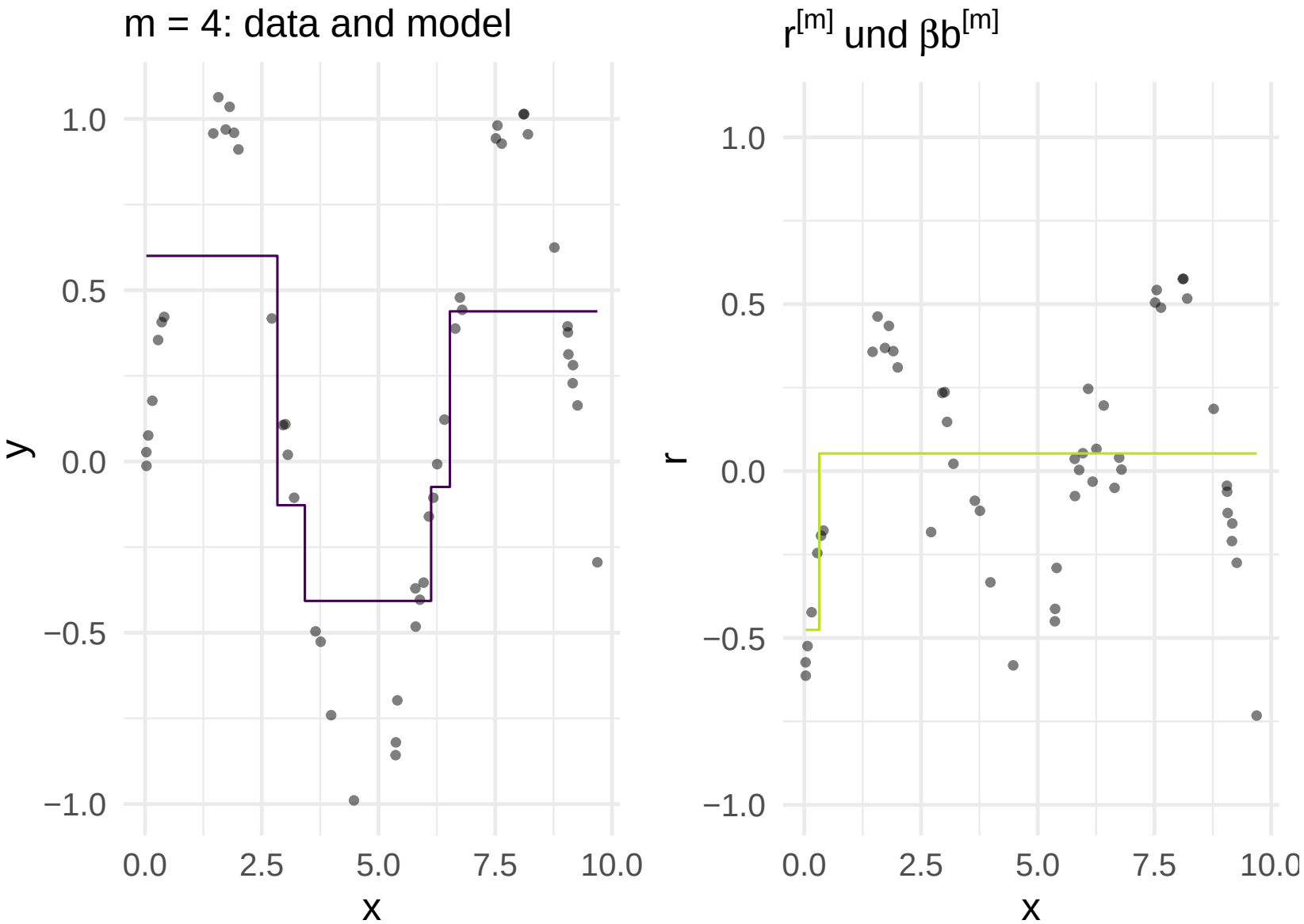


GRADIENT BOOSTING ILLUSTRATION - 2

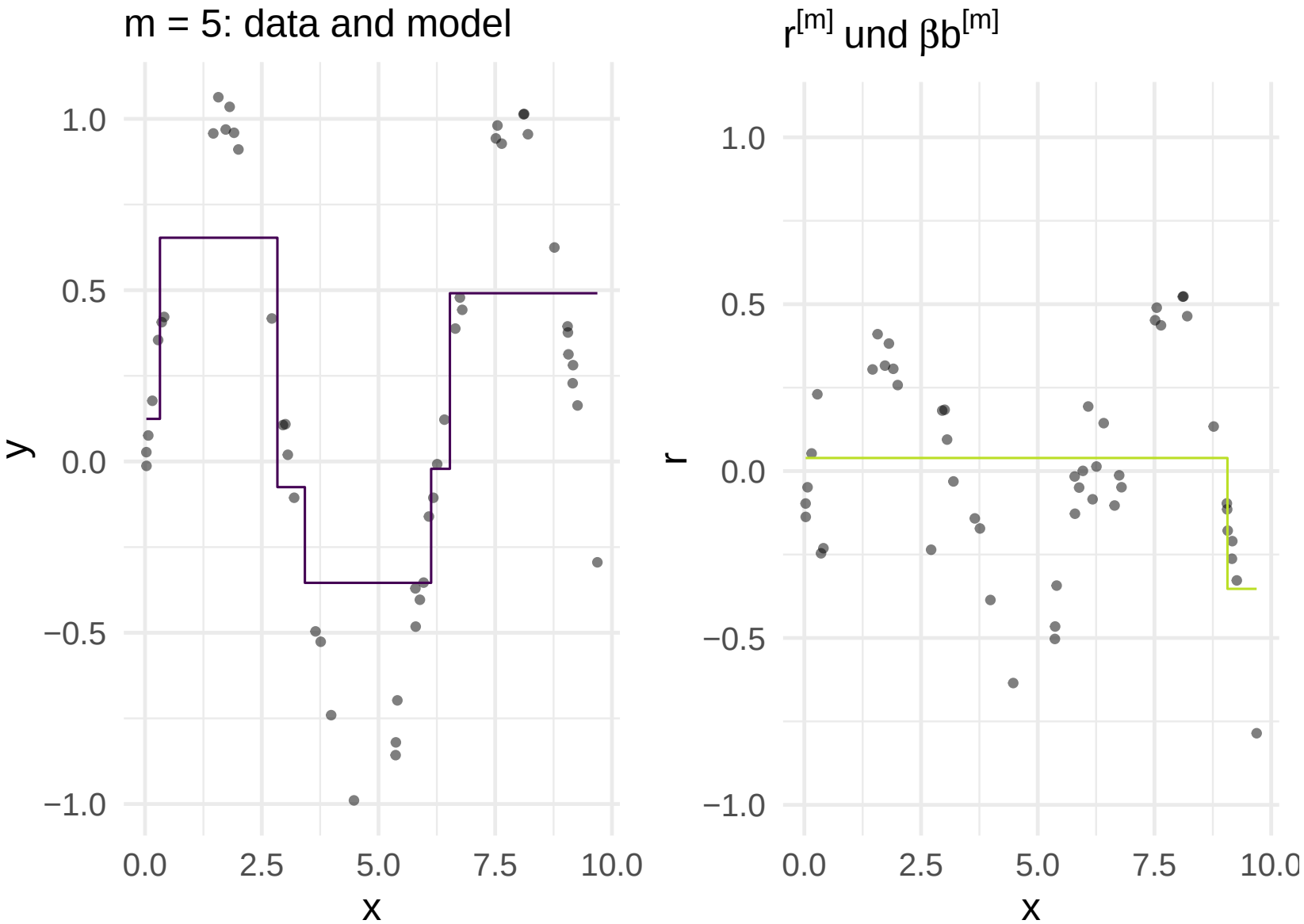




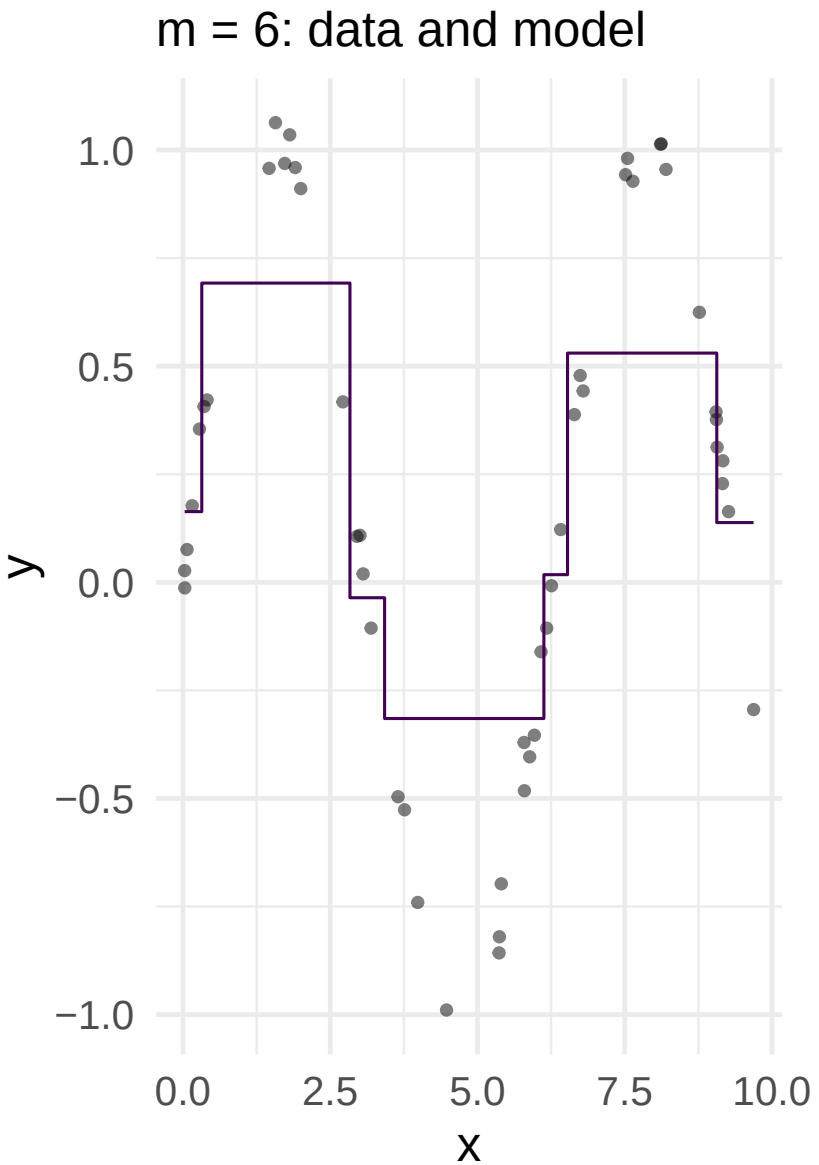
GRADIENT BOOSTING ILLUSTRATION - 2

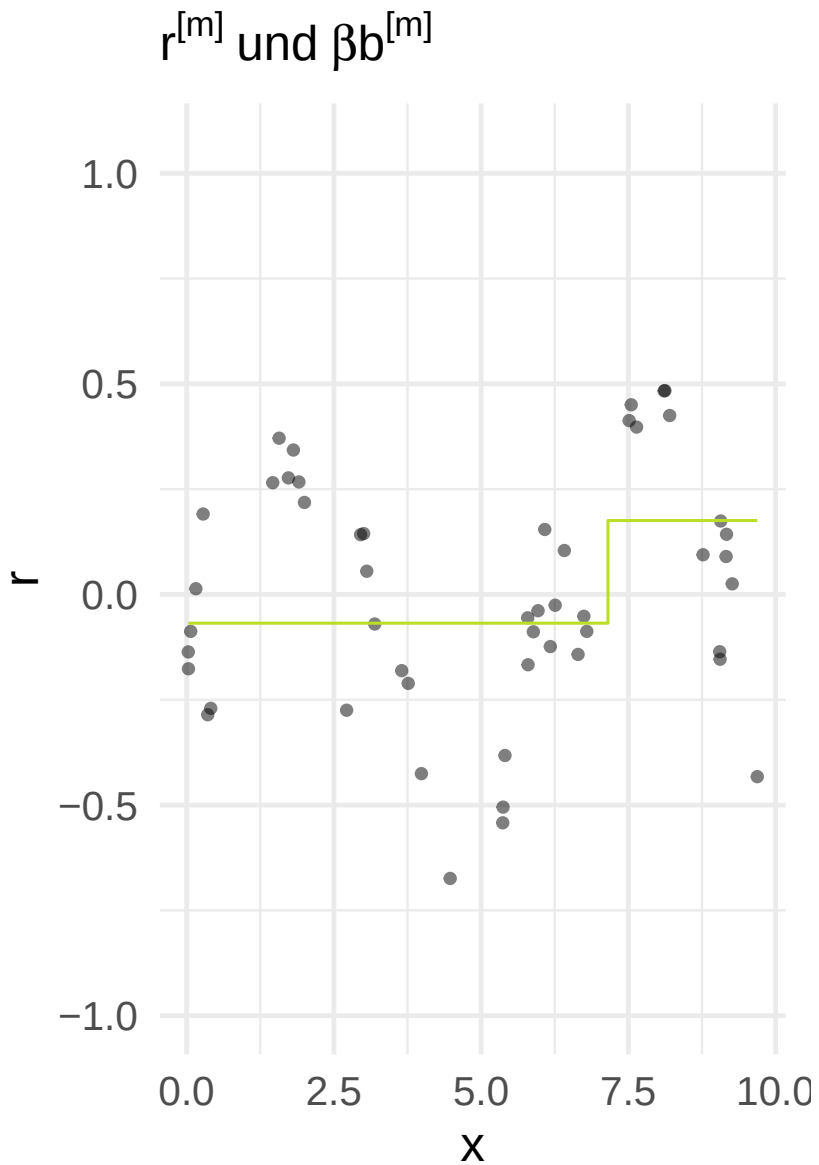


GRADIENT BOOSTING ILLUSTRATION - 2

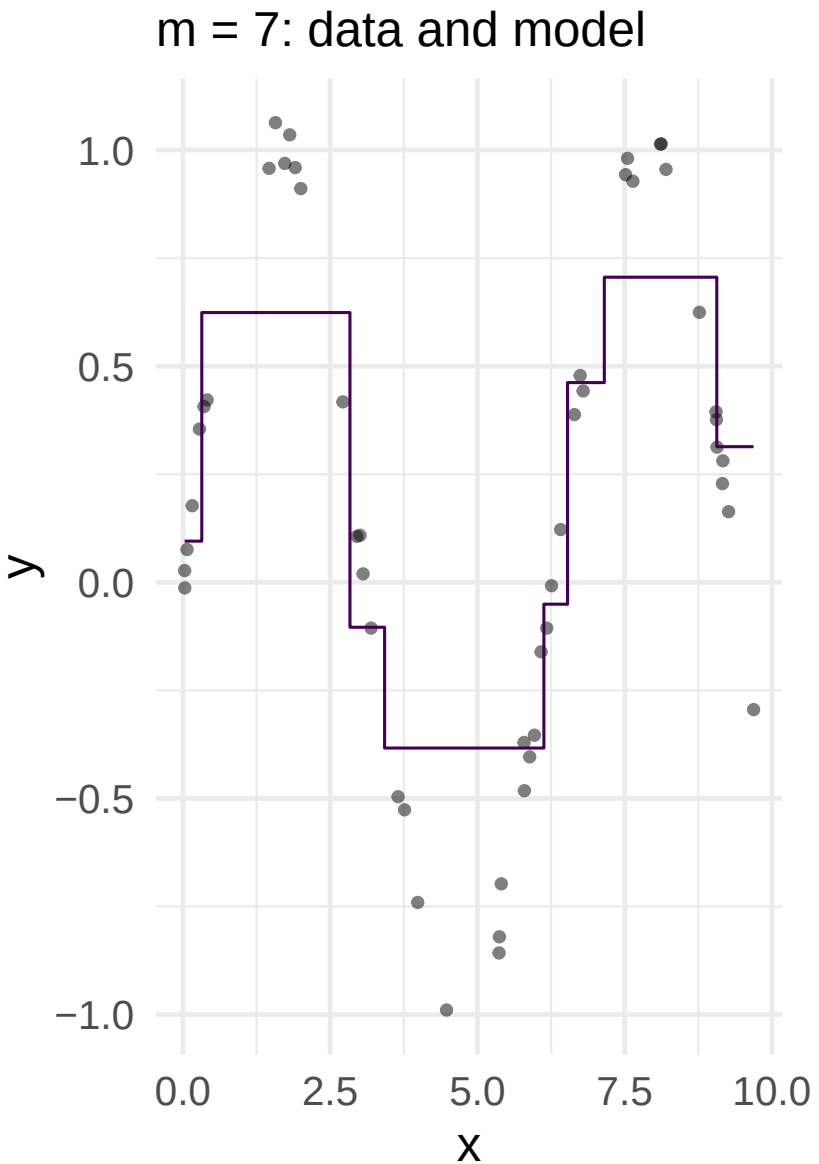


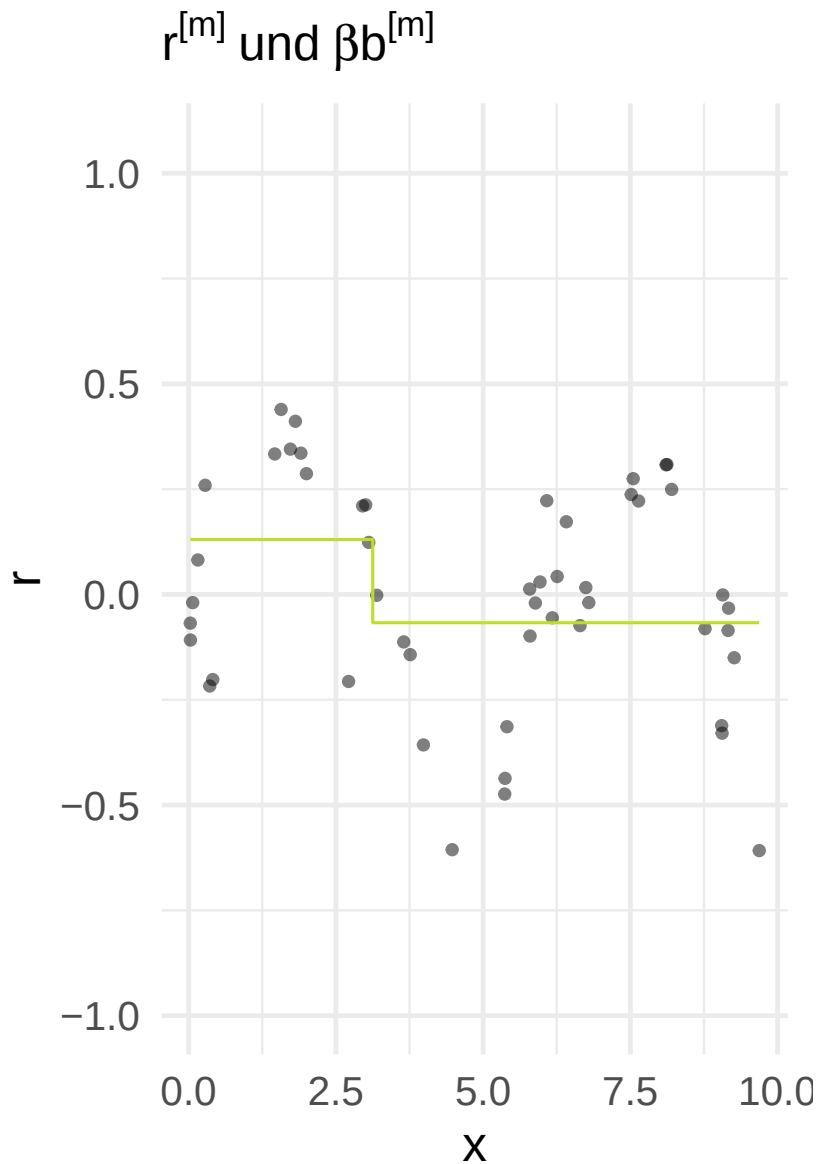
GRADIENT BOOSTING ILLUSTRATION - 2



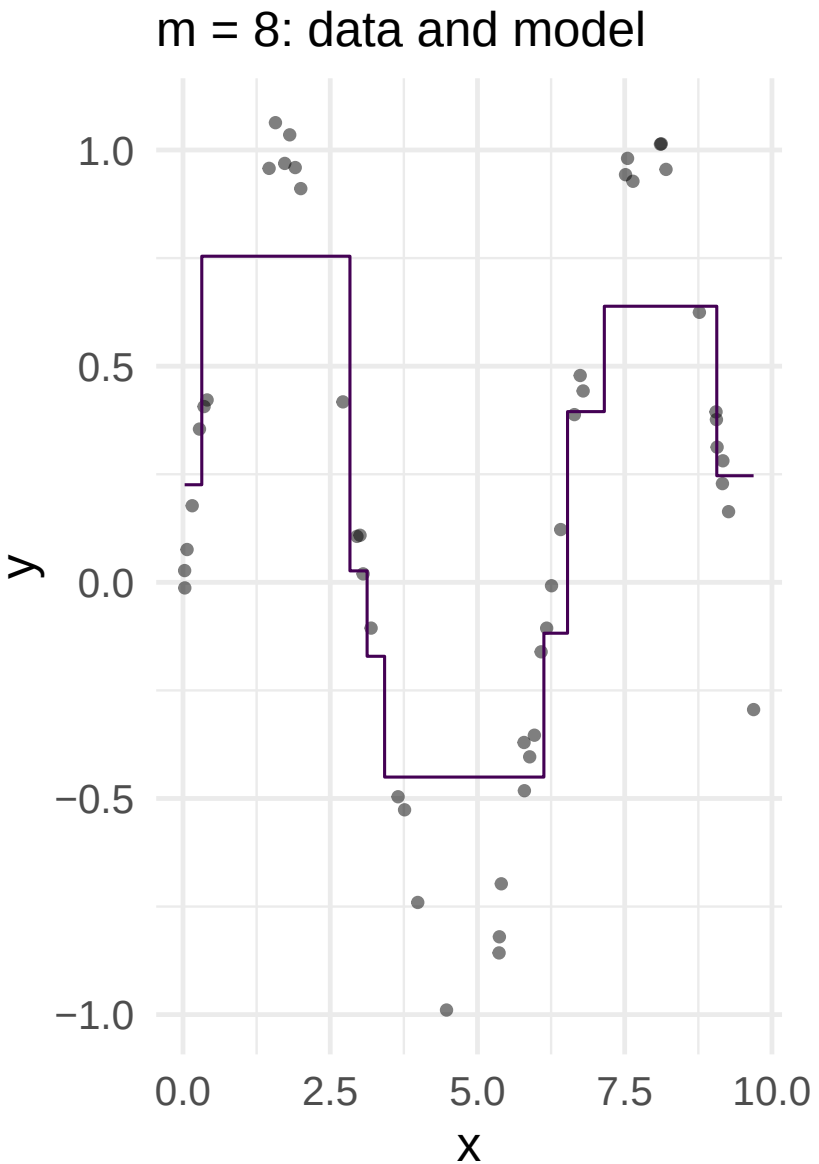


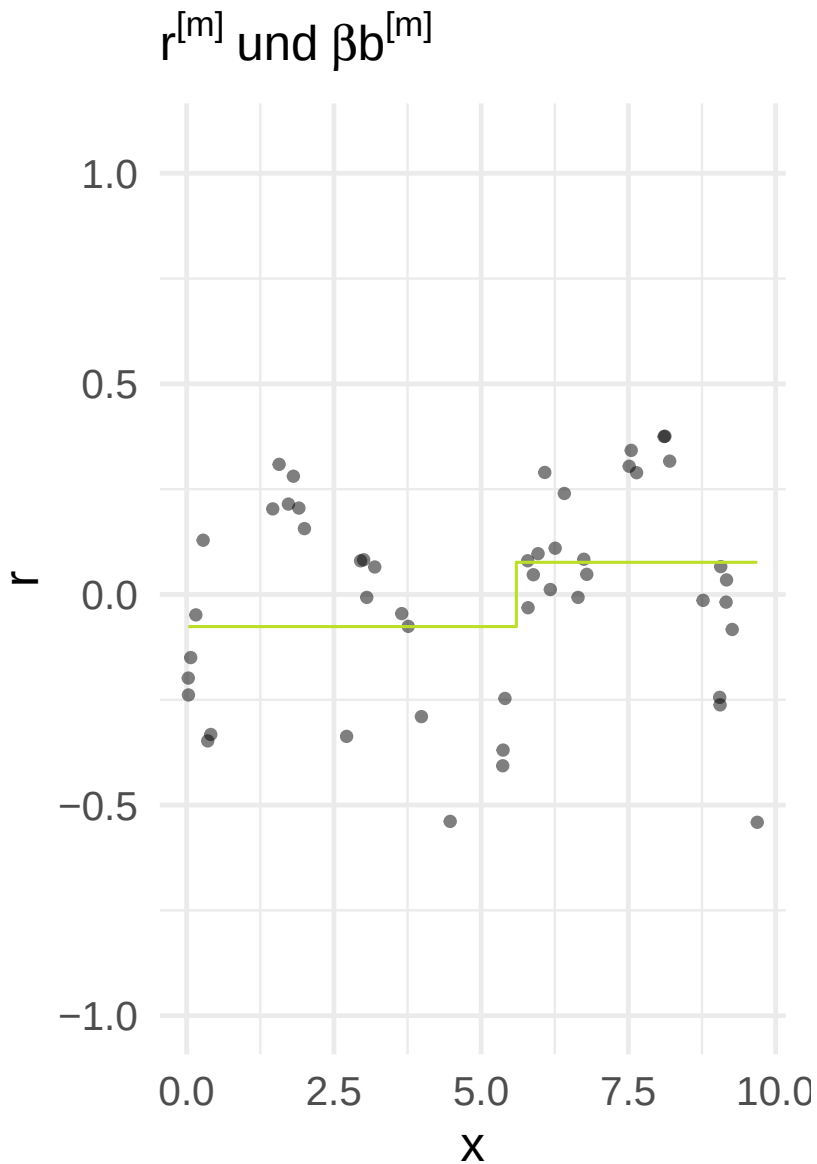
GRADIENT BOOSTING ILLUSTRATION - 2



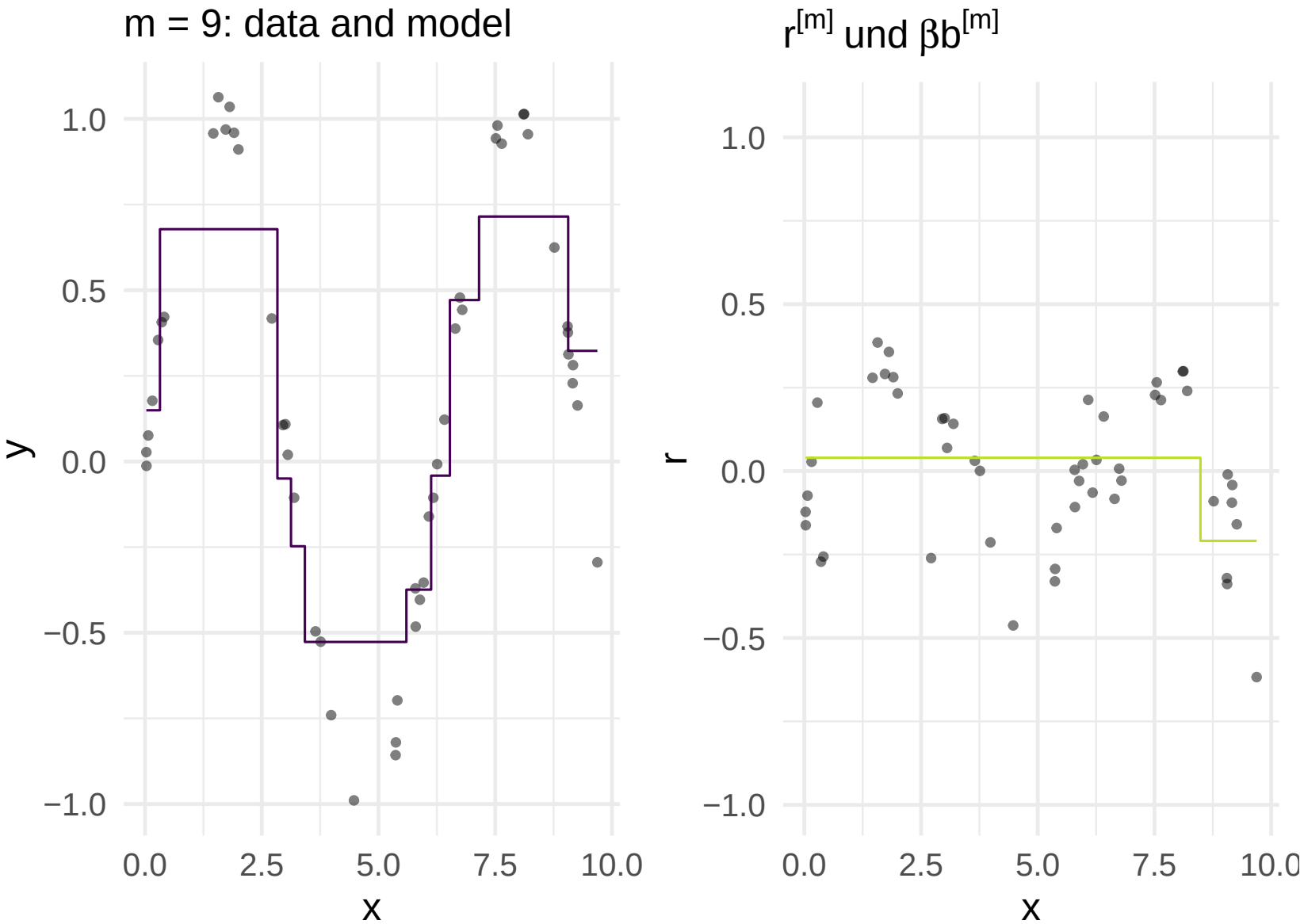


GRADIENT BOOSTING ILLUSTRATION - 2

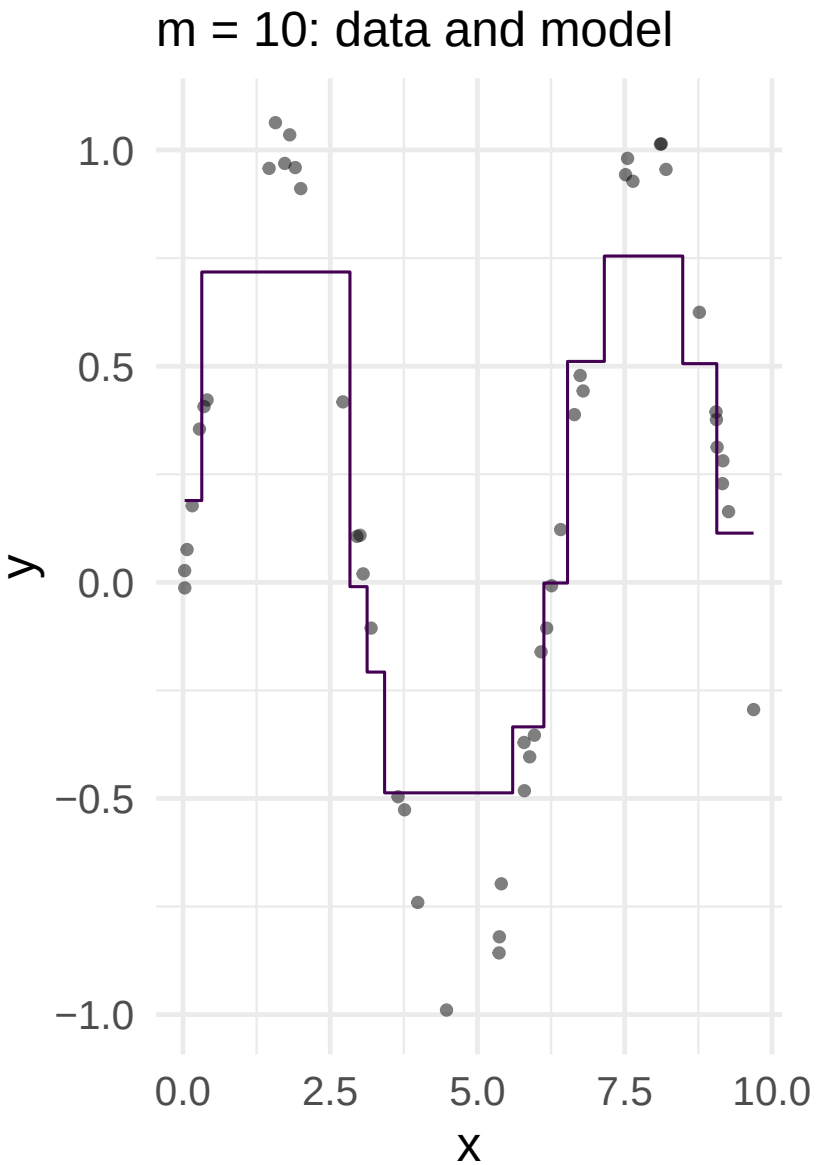


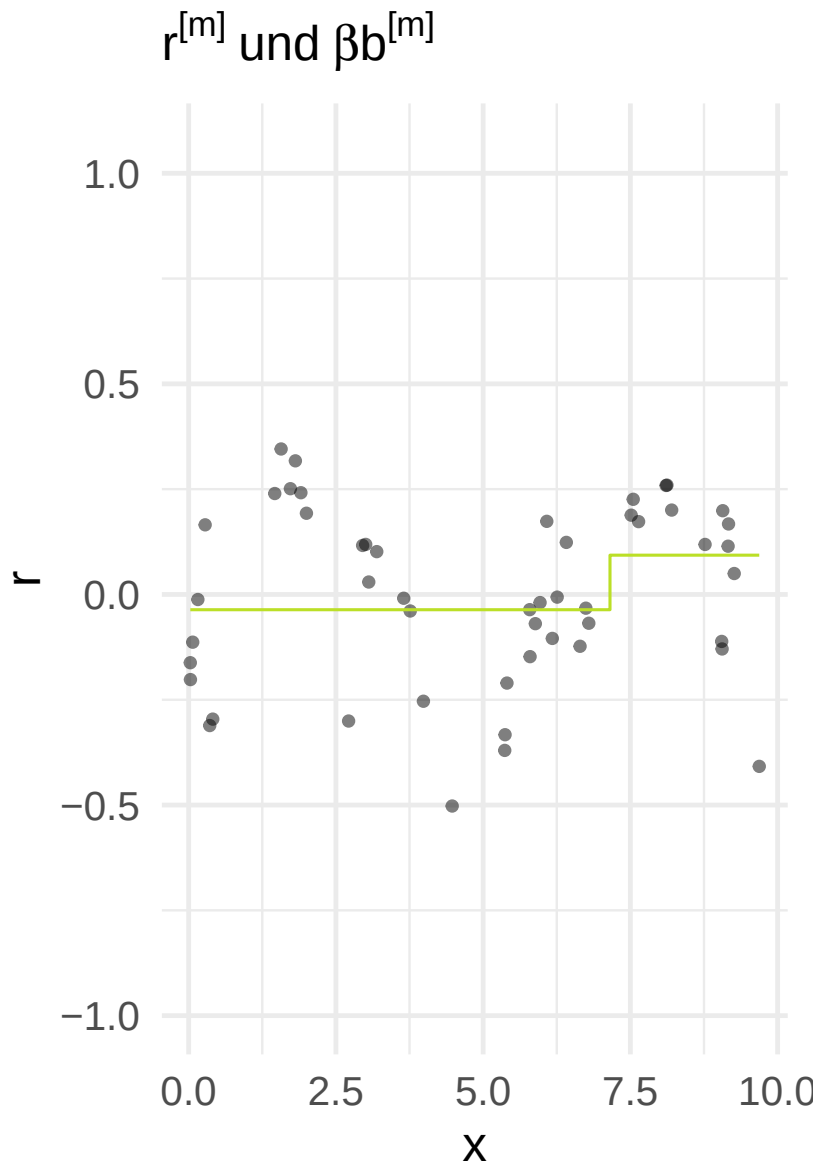


GRADIENT BOOSTING ILLUSTRATION - 2

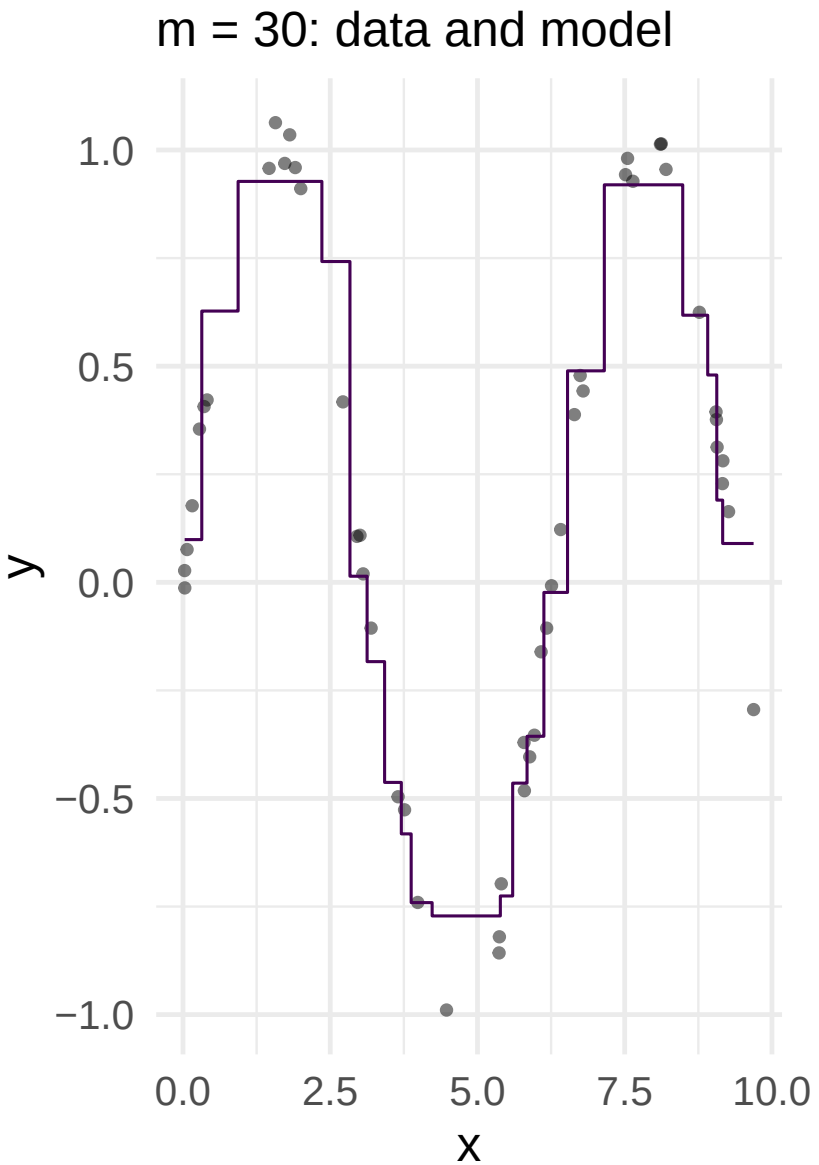


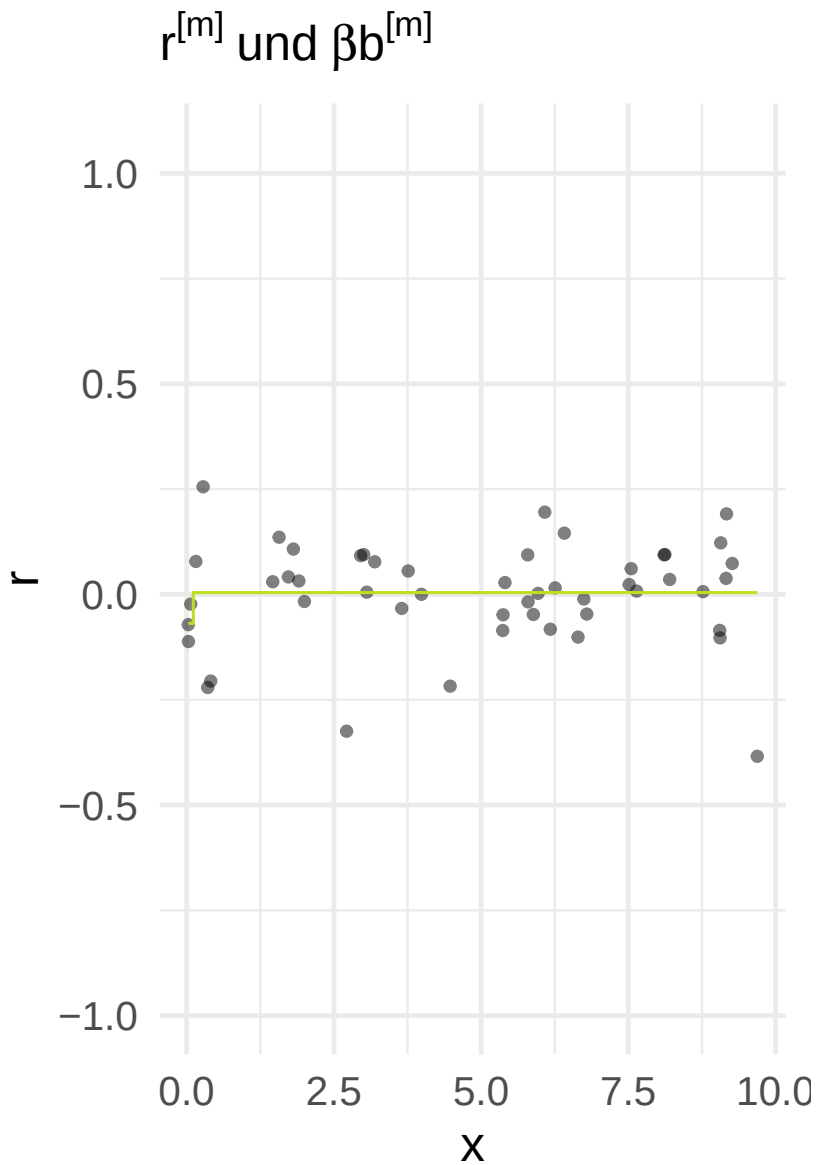
GRADIENT BOOSTING ILLUSTRATION - 2



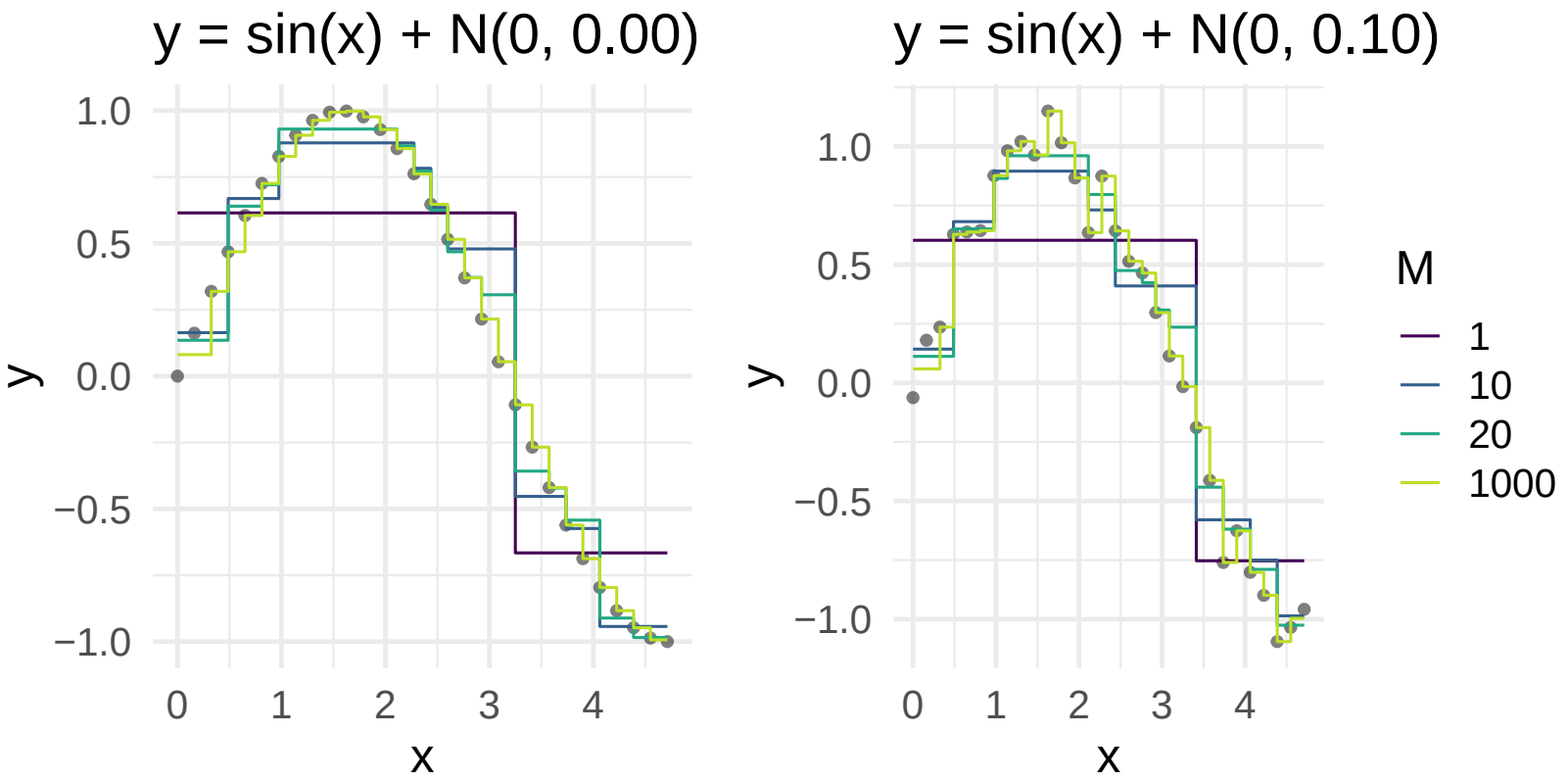


GRADIENT BOOSTING ILLUSTRATION - 2



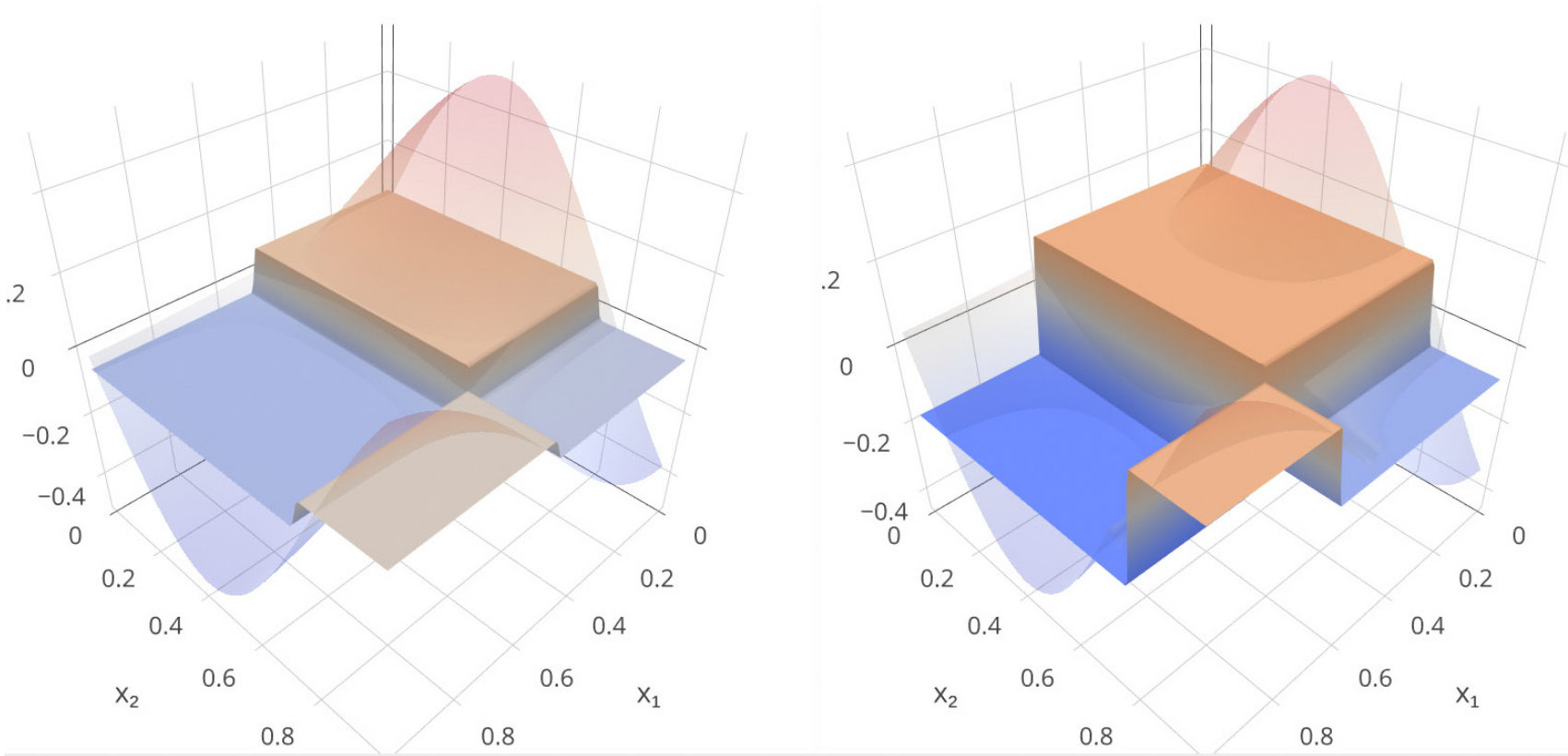


GRADIENT BOOSTING ILLUSTRATION - 2

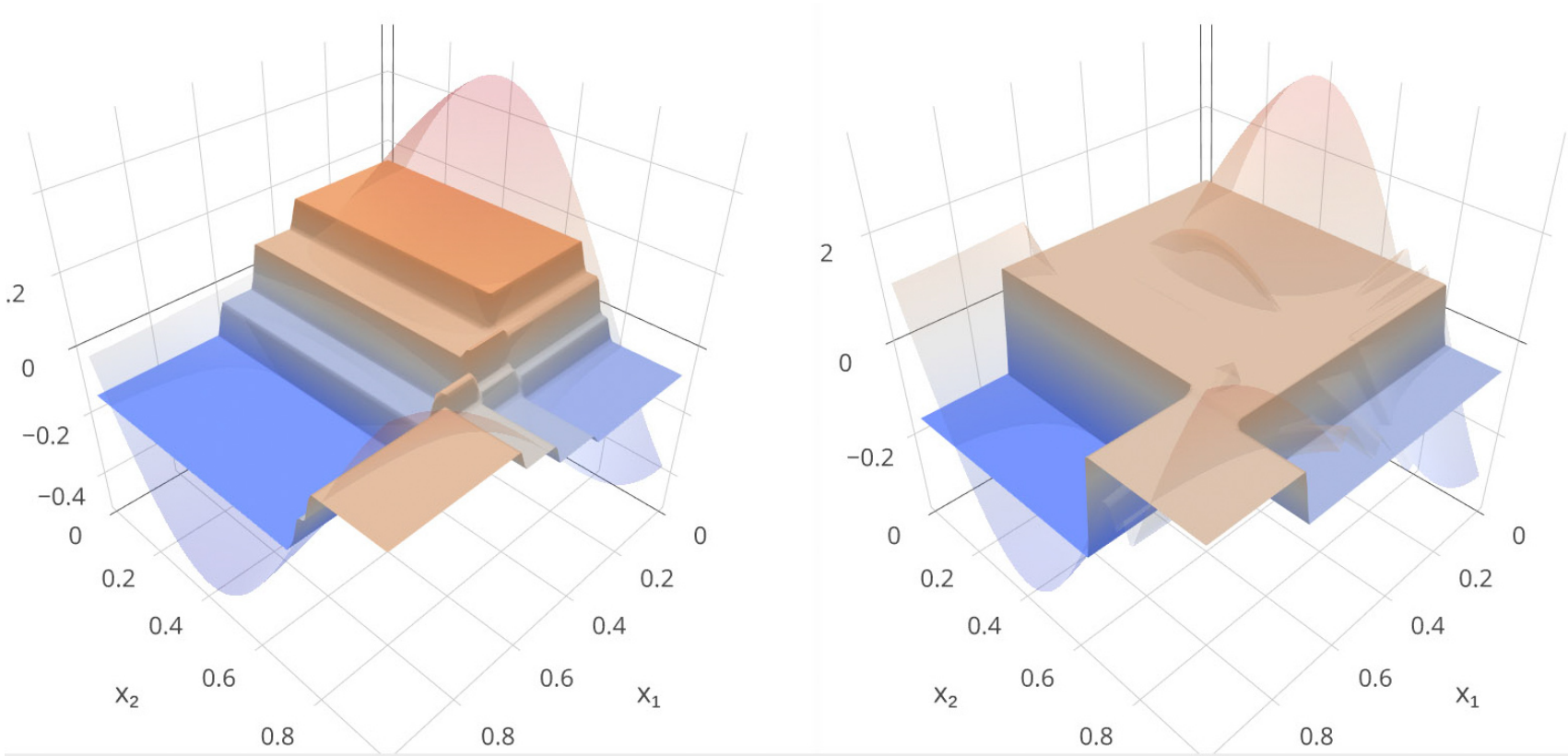


- Iterating this very simple base learner yields a rather nice approximation of a smooth model in the end.
- Severe overfitting apparent in the noisy case. We'll discuss and solve this problem later.

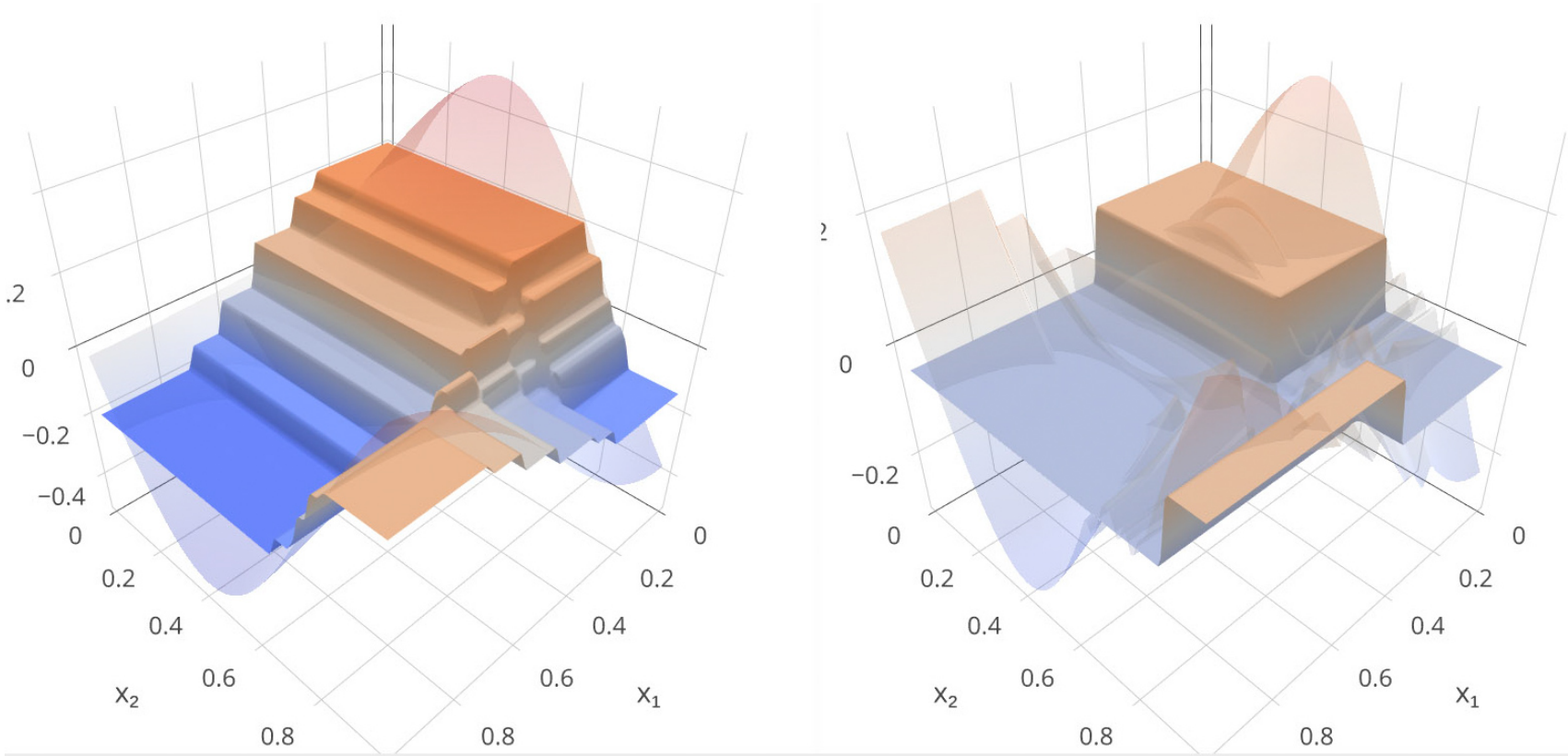
GRADIENT BOOSTING VISUALIZATION



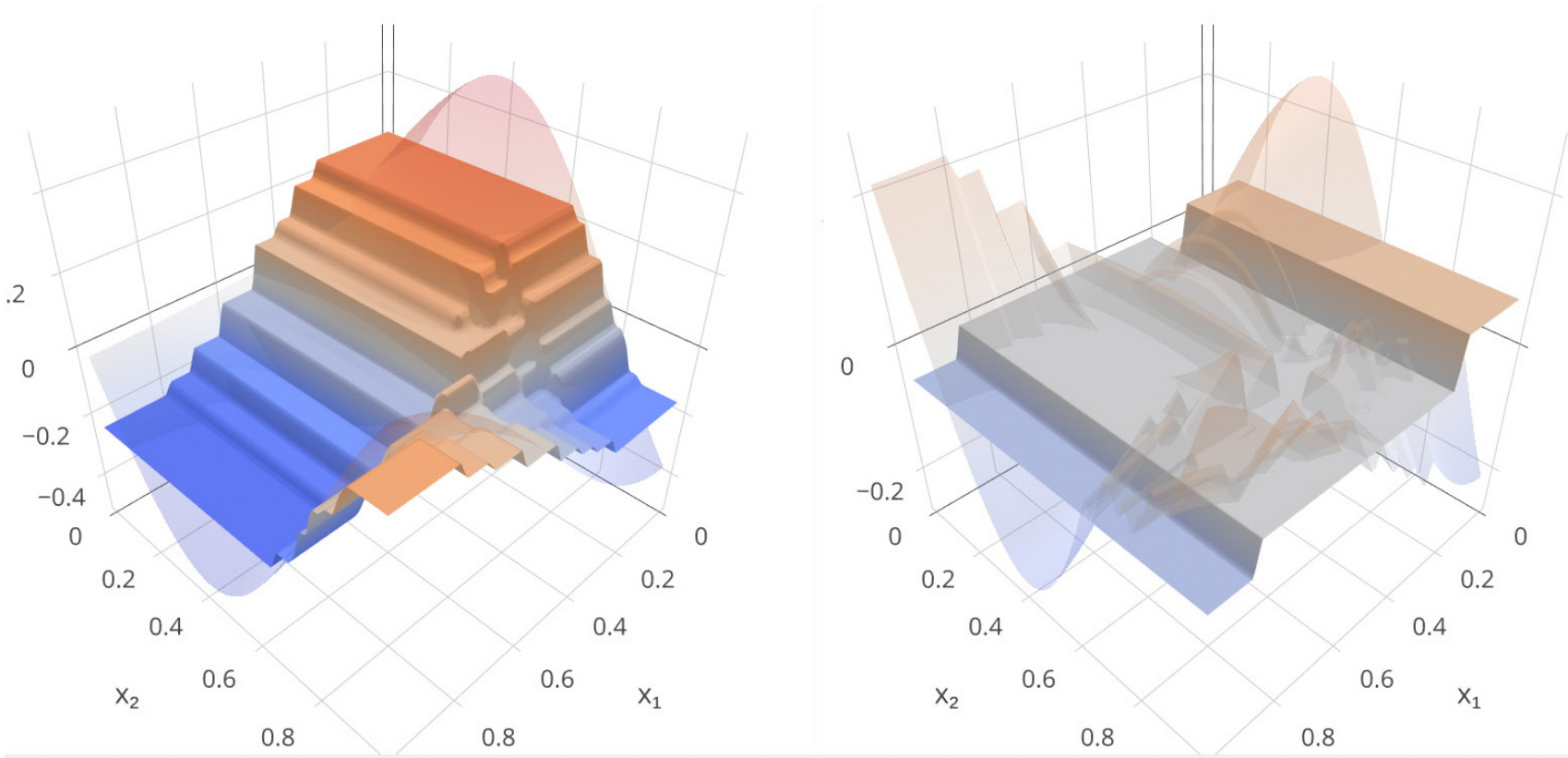
GRADIENT BOOSTING VISUALIZATION



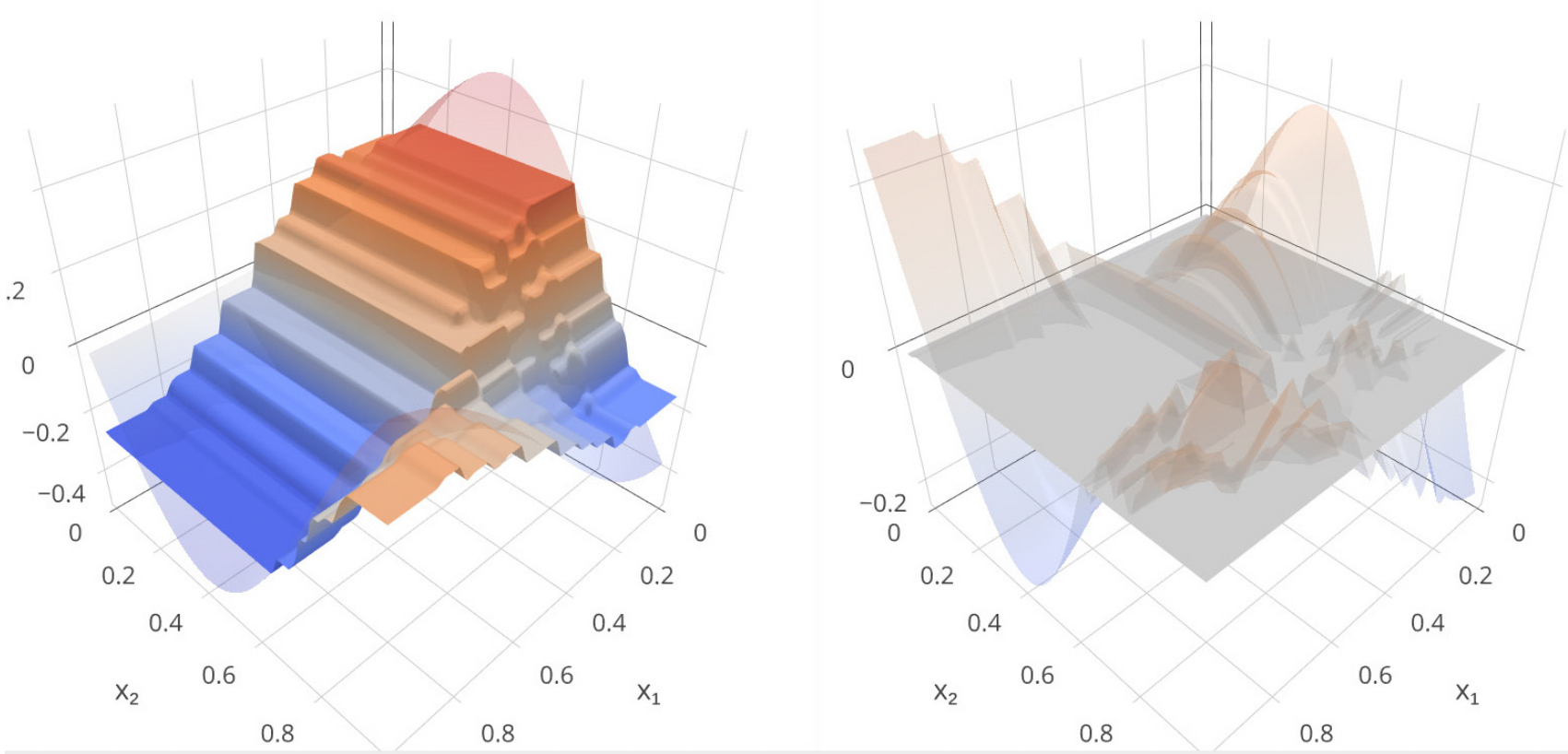
GRADIENT BOOSTING VISUALIZATION



GRADIENT BOOSTING VISUALIZATION



GRADIENT BOOSTING VISUALIZATION



GRADIENT BOOSTING AND TREES

Trees are mainly used as base learners for gradient boosting in ML. A great deal of research has been done on this combination so far, and it often provides the best results.

Reminder: Advantages of trees

- No problems with categorical features.
- No problems with outliers in feature values.
- No problems with missing values.
- No problems with monotone transformations of features.
- Trees (and stumps!) can be fitted quickly, even for large n.
- Trees have a simple built-in type of variable selection.

Gradient boosted trees retains all of them, and strongly improves the trees' predictive power. Furthermore, it is possible to adapt gradient boosting especially to tree learners.

GRADIENT BOOSTING AND TREES

One can write a tree as: $b(x) = \sum_{t=1}^{T} c_t \mathbb{I}(x \in R_t)$, where R_t are the terminal regions and c_t the corresponding means.

This special additive structure can be exploited by boosting:

$$
\begin{aligned}
f^{[m]}(x) &= f^{[m-1]}(x) + \beta^{[m]} b^{[m]}(x) \\
&= f^{[m-1]}(x) + \beta^{[m]} \sum_{t=1}^{T^{[m]}} c_t^{[m]} \mathbb{I}(x \in R_t^{[m]})
\end{aligned}
$$

Actually, we do not have to find $c_t^{[m]}$ and $\beta^{[m]}$ in two separate steps (fitting against pseudo-residuals, then line search). Also note that the $c_t^{[m]}$ will not really be loss-optimal as we used squared error loss to fit them against the pseudo residuals.

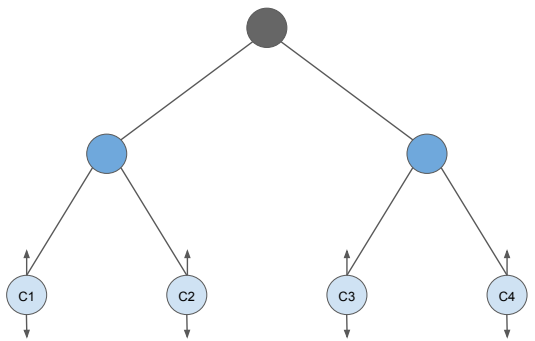
GRADIENT BOOSTING AND TREES

What we will do instead, is:

$$f^{[m]}(x) = f^{[m-1]}(x) + \sum_{j=1}^{T^{[m]}} \tilde{c}_t^{[m]} \mathbb{I}(x \in R_t^{[m]})$$

We now induce the structure of the tree with respect to squared error loss, but then determine / change all $\tilde{c}_t^{[m]}$ individually and directly L-optimally:

$$\tilde{c}_t^{[m]} = \arg\min_c \sum_{\mathbf{x}^{(i)} \in R_t^{[m]}} L(y^{(i)}, f^{[m-1]}(\mathbf{x}^{(i)}) + c)$$



Algorithm 3 Gradient Tree Boosting Algorithm.

Initialize $\hat{f}^{[0]}(x) = \arg\min_\theta \sum_{i=1}^{n} L(y^{(i)}, \theta)$
for $m = 1 \to M$ **do**
  For all i: $r^{[m](i)} = -\left[\frac{\partial L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)}{\partial f\left(\mathbf{x}^{(i)}\right)}\right]_{f=\hat{f}^{[m-1]}}$
  Fit regr. tree to the $r^{[m](i)}$ giving terminal regions $R_t^{[m]},\ t = 1, \ldots, T^{[m]}$
  for $t = 1 \to T^{[m]}$ **do**
    $\hat{c}_t^{[m]} = \arg\min_c \sum_{\mathbf{x}^{(i)} \in R_t^{[m]}} L(y^{(i)}, f^{[m-1]}(\mathbf{x}^{(i)}) + c)$
  end for
  $\hat{b}^{[m]}(x) = \sum_{t=1}^{T} \hat{c}_t^{[m]} \mathbb{I}(x \in R_t)$,
  Update $\hat{f}^{[m]}(x) = \hat{f}^{[m-1]}(x) + \hat{b}^{[m]}(x)$
end for
Output $\hat{f}(x) = \hat{f}^{[M]}(x)$

BINARY CLASSIFICATION

For $\mathcal{Y} = \{0, 1\}$, we simply have to select an appropriate loss function, so let's use binomial loss as in logistic regression:

$$L(y, f(\mathbf{x})) = -yf(\mathbf{x}) + ln(1 + \exp(f(\mathbf{x})))$$

Then,

$$\begin{aligned} \tilde{r} &= -\frac{\partial L(y, f(\mathbf{x}))}{\partial f(\mathbf{x})} \\ &= y - \frac{\exp(f(\mathbf{x}))}{1 + \exp(f(\mathbf{x}))} \\ &= y - \frac{1}{1 + \exp(-f(\mathbf{x}))} = y - s(f(\mathbf{x})). \end{aligned}$$

Here, $s(f(\mathbf{x}))$ is the logistic sigmoid function, applied to a scoring model. so effectively the pseudo residuals are $y - \pi(\mathbf{x})$.
Through $\pi(\mathbf{x}) = s(f(\mathbf{x}))$ we can also estimate posterior probabilities.

MULTI-CLASS PROBLEMS

We proceed as in softmax regression and model a categorical distribution, with multinomial loss. For $\mathcal{Y} = \{1, \ldots, g\}$, we create g discriminant functions $f_k(x)$, one for each class, each an additive model of trees.
The $\pi_k(x)$ we define through the softmax function:

$$\pi_k(\mathbf{x}) = s_k(f_1(x), \ldots, f_g(x)) = \exp(f_k(x)) / \sum_{j=1}^{g} \exp(f_j(x))$$

Multinomial loss L:

$$L(y, f_1(x), \ldots f_g(x)) = -\sum_{k=1}^{g} \mathbb{I}(y = k) \ln \pi_k(\mathbf{x})$$

And for the derivative the following holds:

$$-\frac{\partial L(y_k, f_1(y), \ldots, f_g(x))}{\partial f_k(x)} = \mathbb{I}(y = k) - \pi_k(\mathbf{x})$$

MULTI-CLASS PROBLEMS

Determining the tree structure by squared-error-loss works just like before in the 2 class problem.

In the estimation of the c though, all the models depend on each other because of the definition of L. Optimizing this is more difficult, so we will skip the details and just present the results.

The estimated class for x is of course exactly the k for which $\pi_k(\mathbf{x})$ is maximal.

MULTI-CLASS PROBLEMS

Algorithm 4 Gradient Boosting for K-class Classification.

Initialize $f_k^{[0]}(x) = 0,\ k = 1, \ldots, g$
for $m = 1 \to M$ **do**
  Set $\pi_k(\mathbf{x}) = \frac{\exp(f_k(x))}{\sum_j \exp(f_j(x))}, k = 1, \ldots, g$
  for $k = 1 \to g$ **do**
    For all i: Compute $r_k^{[m](i)} = \mathbb{I}(y^{(i)} = k) - \pi_k(\mathbf{x}^{(i)})$
    Fit regr. tree to the $r_k^{[m](i)}$ giving terminal regions $R_{tk}^{[m]}$
    Compute

$$\hat{c}_{tk}^{[m]} = \frac{g-1}{g} \frac{\sum_{\mathbf{x}^{(i)} \in R_{tk}^{[m]}} r_k^{[m](i)}}{\sum_{\mathbf{x}^{(i)} \in R_{tk}^{[m]}} \left| r_k^{[m](i)} \right| \left(1 - \left| r_k^{[m](i)} \right|\right)}$$

    Update $\hat{f}_k^{[m]}(x) = \hat{f}_k^{[m-1]}(x) + \sum_t \hat{c}_{tk}^{[m]} \mathbb{I}\left(x \in R_{tk}^{[m]}\right)$
  end for
end for
Output $\hat{f}_1^{[M]}, \ldots, \hat{f}_g^{[M]}$

MULTI-CLASS PROBLEMS

Derivation of the algorithm:

- from Friedman, J. H. - Greedy Function Approximation: A Gradient Boosting Machine (1999)
- In each iteration m we calculate the pseudo residuals

$$r_k^{[m](i)} = \mathbb{I}(y^{(i)} = k) - \pi_k^{[m-1]}(\mathbf{x}^{(i)}),$$

  where $\pi_k^{[m-1]}(\mathbf{x}^{(i)})$ is derived from $f^{[m-1]}(\mathbf{x})$
- Thus, g trees are induced at each iteration m to predict the corresponding current pseudo residuals for each class on the probability scale.
- Each of these trees has T terminal nodes with corresponding regions $R_{tk}^{[m]}$.

MULTI-CLASS PROBLEMS

- The model updates $\hat{c}_{tk}^{[m]}$ corresponding to these regions are the solution to

$$\hat{c}_{tk}^{[m]} = \arg\min_{c} \sum_{i=1}^{n} \sum_{k=1}^{g} L\left(y_k^{(i)}, f^{[m-1]}(\mathbf{x}^{(i)}) + \sum_{t=1}^{T} \hat{c}_{tk} \mathbb{I}\left(x_i \in R_t^{[m]}\right)\right)$$

  where L is the multinomial loss function
  $L(y, f_1(x), \ldots f_g(x)) = -\sum_{k=1}^{g} \mathbb{I}(y = k) \ln \pi_k(\mathbf{x})$ and
  $\pi_k(\mathbf{x}) = \frac{\exp(f_k(x))}{\sum_j \exp(f_j(x))}$ as before.

- This has no closed form solution and additionally, the regions corresponding to the different class tress overlap, so that the solution does not reduce to a separate calculation within each region of each tree.

MULTI-CLASS PROBLEMS

- Hence, we approximate the solution with a single Newton-Raphson step, using a diagonal approximation to the Hessian.
- This decomposes the problem into a separate calculation for each terminal node of each tree.
- The result is

$$\hat{c}_{tk}^{[m]} = \frac{g-1}{g} \frac{\sum_{\mathbf{x}^{(i)} \in R_{tk}^{[m]}} r_k^{[m](i)}}{\sum_{\mathbf{x}^{(i)} \in R_{tk}^{[m]}} \left| r_k^{[m](i)} \right| \left(1 - \left| r_k^{[m](i)} \right| \right)}$$

- The update is then done by

$$\hat{f}_k^{[m]}(x) = \hat{f}_k^{[m-1]}(x) + \sum_t \hat{c}_{tk}^{[m]} \mathbb{I}\left(x \in R_{tk}^{[m]} \right)$$

REGULARIZATION AND SHRINKAGE

If GB runs for a long number of iterations, it can overfit due to its aggressive loss minimization.

Options for regularization

- Limit the number of boosting iterations M ("early stopping"), i.e., limit the number of additive components.
- Limit the depth of the trees. This can also be interpreted as choosing the order of interaction.
- Shorten the step length $\beta^{[m]}$ in each iteration.

The latter is achieved by multiplying $\beta^{[m]}$ with a small $\nu \in (0, 1]$:

$$f^{[m]}(x) = f^{[m-1]}(x) + \nu\beta^{[m]} b(x, \theta^{[m]})$$

ν is called **shrinkage parameter** or **learning rate**.

REGULARIZATION AND SHRINKAGE

Obviously, the optimal values for M and ν strongly depend on each other: By increasing M one can use a smaller value for ν and vice versa.

In practice it is often recommended to choose ν quite small and choose M by cross-validation.

It's probably best to tune all three parameters jointly based on the training data via cross-validation or a related method.

STOCHASTIC GRADIENT BOOSTING

This is a minor modification to boosting to incorporate the advantages of bagging into the method. The idea was formulated quite early by Breiman.

Instead of fitting on all the data points, a random subsample is drawn in each iteration.

Especially for small training sets, this simple modification often leads to significant empirical improvements. How large the improvements are depends on data structure, size of the data set, base learner and size of the subsamples (so this is another tuning parameter).

EXAMPLE: SPAM DETECTION

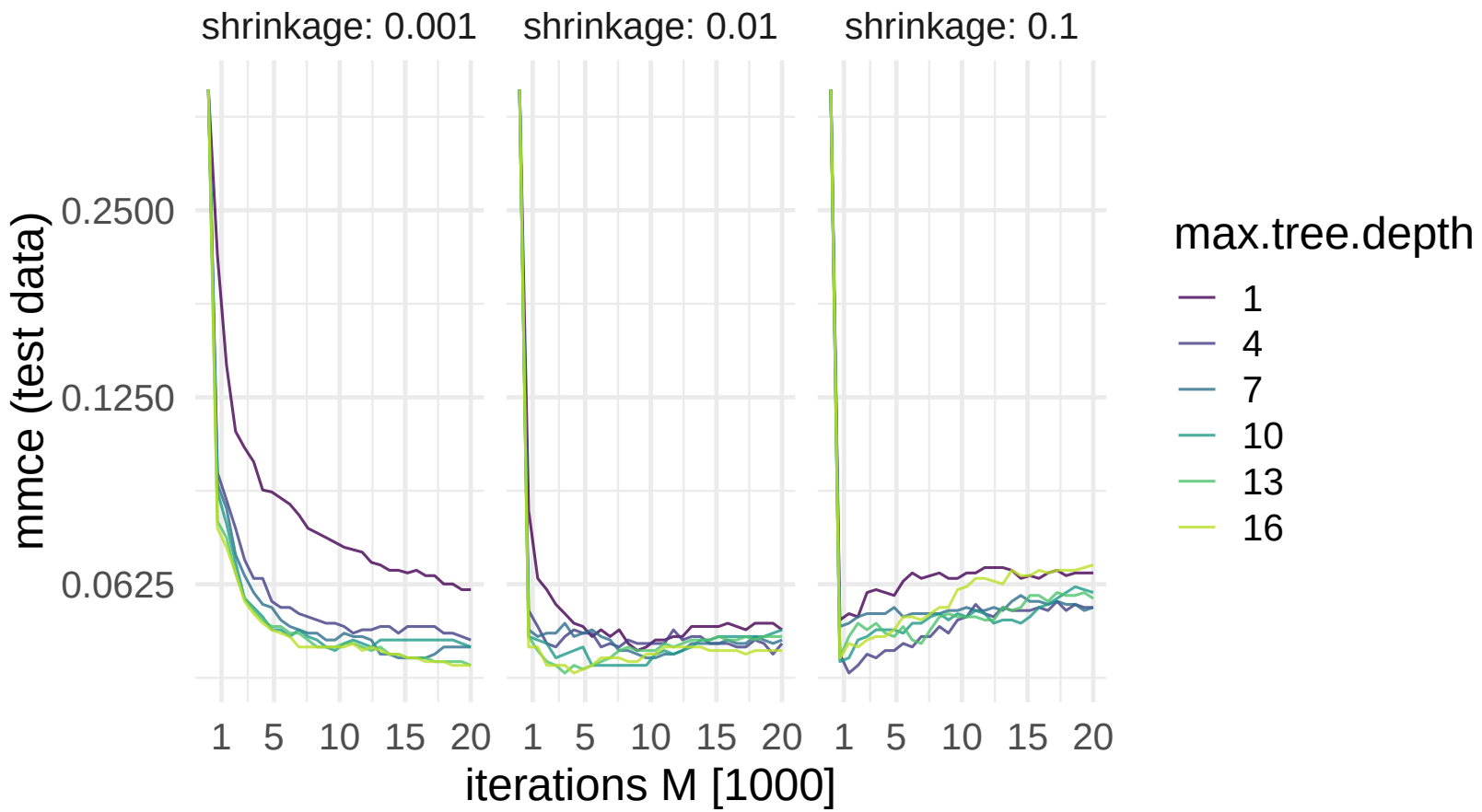
We fit a gradient boosting model for different parameter values

Parameter name	Values
distribution	Bernoulli (for classification)
shrinkage ν	$0.001, 0.01, 0.1$
number of trees M	$[0, \ldots, 20000]$
max. tree depth	$1, 4, 7, 10, 13, 16$

We observe the error on a separate test set to find the optimal parameters.

EXAMPLE: SPAM DETECTION

Misclassification rate for different hyperparameter settings (shrinkage and maximum tree depth) of gradient boosting:



ADDITIONAL INFORMATION

By choosing a suitable loss function it is also possible to model a large number of different problem domains

- Regression
- (Multi-class) Classification
- Count data
- Survival data
- Ordinal data
- Quantile regression
- Ranking problems
- ...

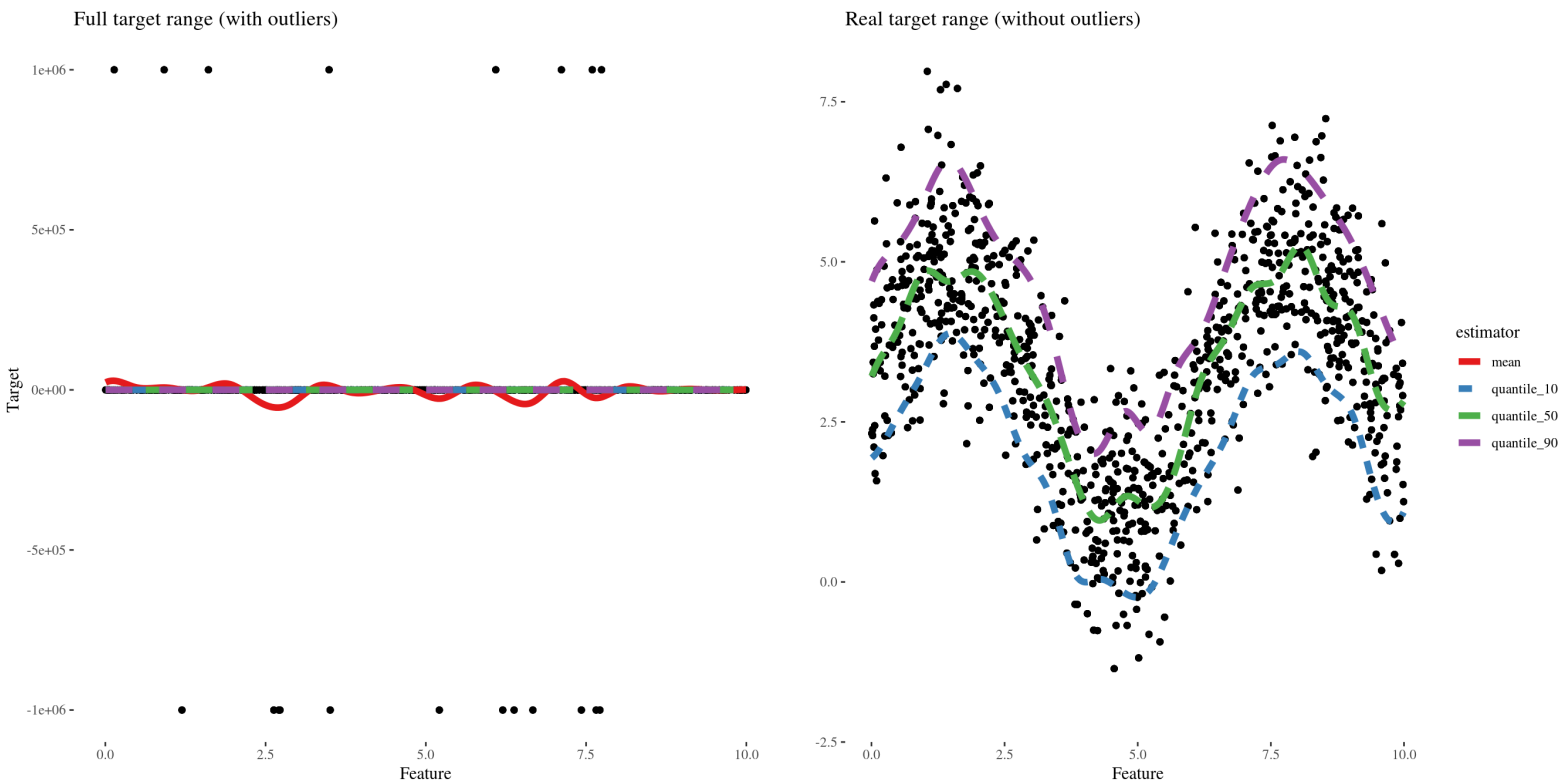
Different base learners increase flexibility (see componentwise gradient boosting). If we model only individual variables, the resulting regularized variable selection is closely related to L1 regularization.

ADDITIONAL INFORMATION

For example, using the pinball loss in boosting

$$L(y, f(\mathbf{x})) = \begin{cases} (1-\alpha)(f(\mathbf{x}) - y), & \text{if } y < f(\mathbf{x}) \\ \alpha(y - f(\mathbf{x})), & \text{if } y \geq f(\mathbf{x}) \end{cases}$$

models the α-quantiles:



ADDITIONAL INFORMATION

The AdaBoost fit has the structure of an additive model with "basis functions" $b^{[m]}(x)$.

It can be shown (see Hastie et al. 2009, chapter 10) that AdaBoost corresponds to minimizing the empirical risk in each iteration m using the *exponential* loss function:

$$L(y, \hat{f}^{[m]}(\mathbf{x})) = \exp\left(-y\hat{f}^{[m]}(\mathbf{x})\right)$$

$$\begin{aligned}\mathcal{R}_{\text{emp}}(\hat{f}^{[m]}) &= \sum_{i=1}^{n} L(y^{(i)}, \hat{f}^{[m]}(\mathbf{x}^{(i)})) \\ &= \sum_{i=1}^{n} L(y^{(i)}, \hat{f}^{[m-1]}(\mathbf{x}^{(i)}) + \beta b(\mathbf{x}^{(i)}))\end{aligned}$$

with minimizing over β and b and where $\hat{f}^{[m-1]}$ is the boosting fit in iteration $m - 1$.

TAKE HOME MESSAGE

Gradient boosting is a statistical reinterpretation of the older AdaBoost algorithm.

Base learners are added in a “greedy” fashion, so that they point in the direction of the negative gradient of the empirical risk.

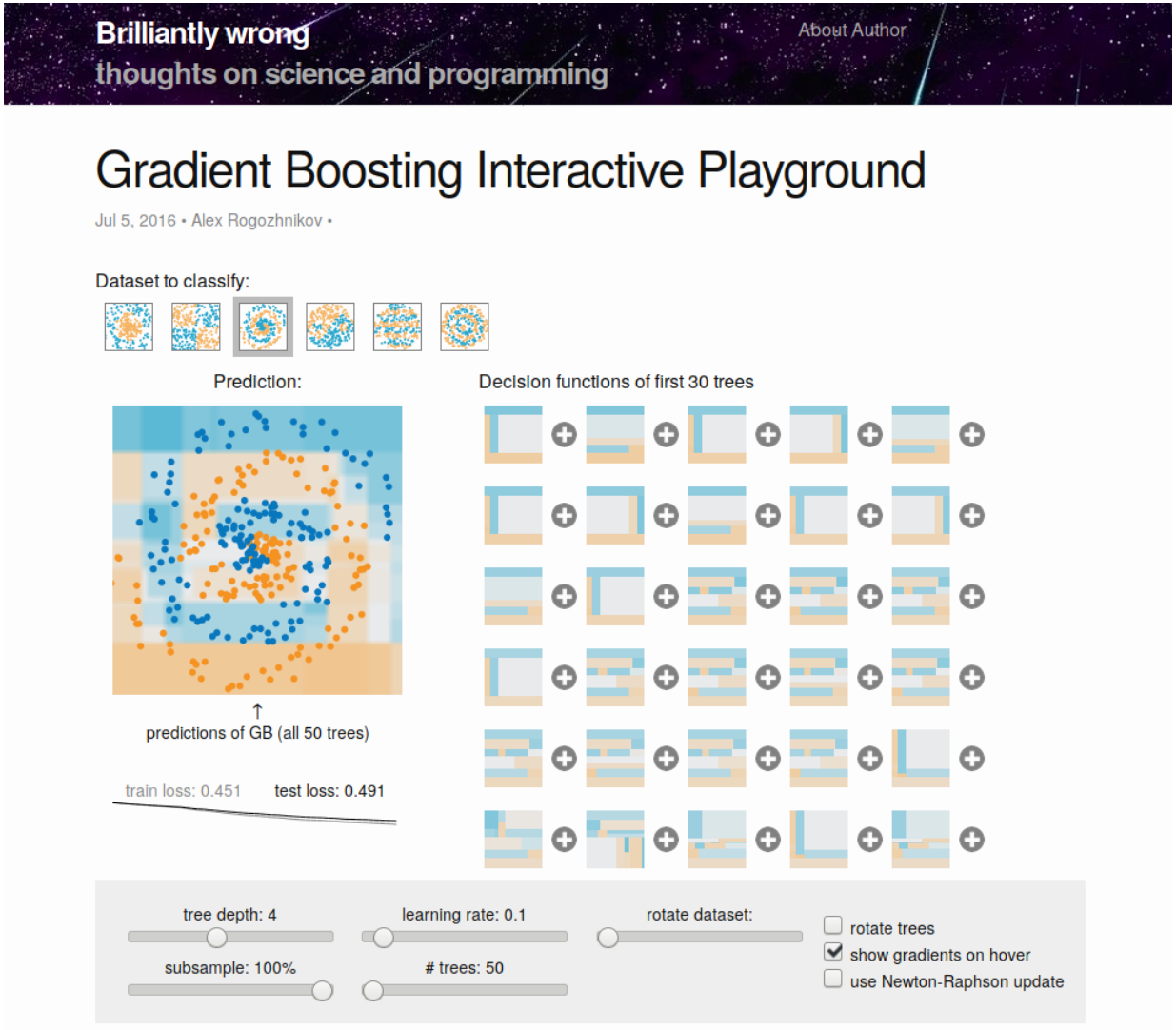
Regression base learners are fitted even for classification problems.

Often the base learners are (shallow) trees, but arbitrary base learners are possible.

The method can be adjusted flexibly by changing the loss function, as long as it’s differentiable.

Methods to evaluate variable importance and to do variable selection exist.

GRADIENT BOOSTING PLAYGROUND



▸ Open in browser.