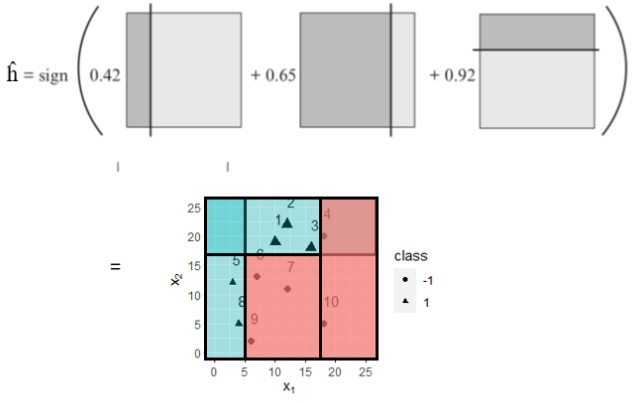
Introduction to Machine Learning

Introduction to Boosting / AdaBoost



Learning goals

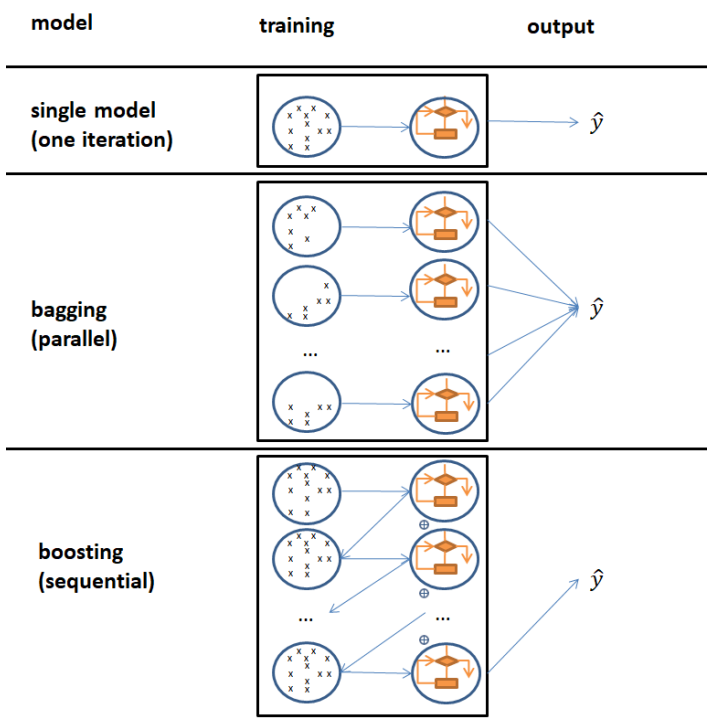
- Briefly cover the older AdaBoost and its general idea
- Understand difference between bagging and boosting

INTRODUCTION TO BOOSTING

- Boosting is one of the most powerful learning ideas since 1990.
- Originally designed for classification, (especially gradient) boosting handles regression (and many other supervised tasks) naturally nowadays.
- 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

- Homogeneous ensemble method (like bagging).
- **Idea:** Take a weak classifier and sequentially apply it to modified versions of the training data.
- Sequential not parallel model building, to minimize loss instead of variance reduction.



BOOSTING AS A THEORETICAL PROBLEM

Boosting was developed as the answer to a theoretical problem:

"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 on a balanced binary problem).
- 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).
- Any weak (base) learner can be iteratively boosted to become a strong learner (Schapire and Freund, 1990).
- Idea was refined into **AdaBoost** (Adaptive Boosting).

ADABOOST

- Assume binary classification with y encoded as $\{-1, +1\}$.
- We use binary classifiers as weak base learners (e.g., tree stumps) from a hypothesis space $\mathcal{B}$, denoted $b^{[m]}$ (or $b^{[m]}(\mathbf{x})$ or $b(\mathbf{x}, \boldsymbol{\theta}^{[m]})$), which are hard labelers and output from $\{-1, +1\}$.
- Decision score of the ensemble of size M is weighted average:

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

- The base learner is sequentially applied to weighted training observations. After each base learner fit, currently misclassified observations receive a higher weight for the next iteration, so we focus more on instances that are harder to classify.
- BLs with higher predictive accuracy receive higher weights $\beta^{[m]}$.

ADABOOST

Algorithm 1 AdaBoost

1: Initialize observation weights: $w^{[1](i)} = \frac{1}{n} \quad \forall i \in \{1, \ldots, n\}$
2: **for** $m = 1 \rightarrow M$ **do**
3: Fit classifier to training data with weights $w^{[m]}$ and get $\hat{b}^{[m]}$
4: Calculate weighted in-sample misclassification rate

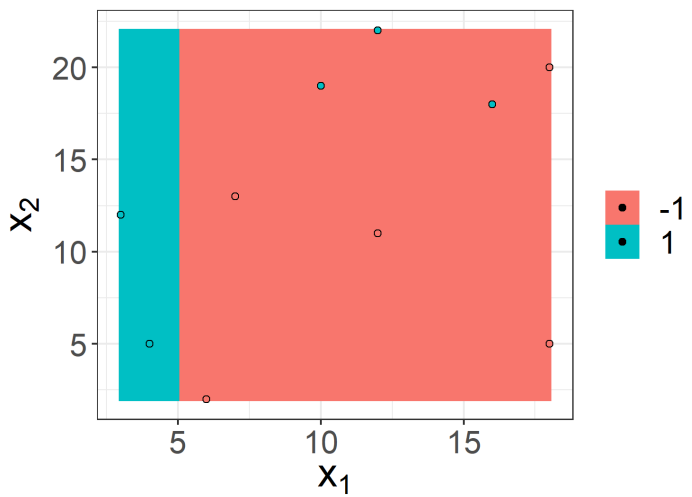
$$\text{err}^{[m]} = \sum_{i=1}^{n} w^{[m](i)} \cdot \mathbb{1}_{\{y^{(i)} \neq \hat{b}^{[m]}(\mathbf{x}^{(i)})\}}$$

5: Compute: $\hat{\beta}^{[m]} = \frac{1}{2} \log\left(\frac{1-\text{err}^{[m]}}{\text{err}^{[m]}}\right)$
6: Set: $w^{[m+1](i)} = w^{[m](i)} \cdot \exp\left(-\hat{\beta}^{[m]} \cdot y^{(i)} \cdot \hat{h}(\mathbf{x}^{(i)})\right)$
7: Normalize $w^{[m+1](i)}$ such that $\sum_{i=1}^{n} w^{[m+1](i)} = 1$
8: **end for**
9: Output: $\hat{f}(\mathbf{x}) = \sum_{m=1}^{M} \hat{\beta}^{[m]} \hat{b}^{[m]}(\mathbf{x})$

ADABOOST ILLUSTRATION

Example

- $n = 10$ observations and two features x_1 and x_2
- Tree stumps as base learners $b^{[m]}(\mathbf{x})$
- Balanced classification task with y encoded as $\{-1, +1\}$
- $M = 3$ iterations $\Rightarrow$ initial weights $w^{[1](i)} = \frac{1}{10} \quad \forall i \in 1, \ldots, 10.$



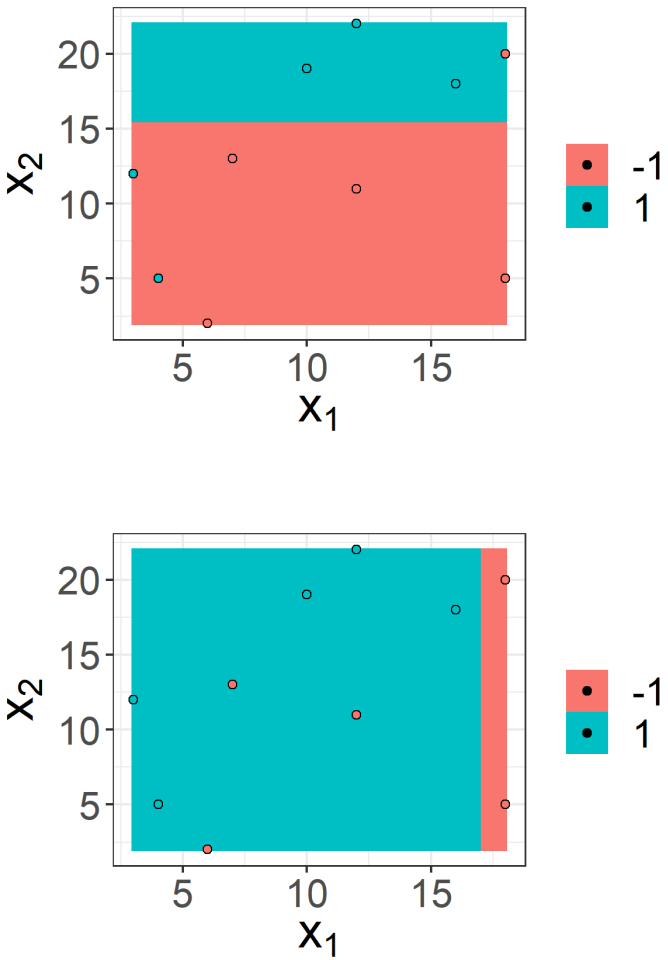
Iteration $m = 1$:

- $\text{err}^{[1]} = 0.3$
- $\hat{\beta}^{[1]} = \frac{1}{2} \log\left(\frac{1-0.3}{0.3}\right) \approx 0.42$

New observation weights:

- Prediction correct:
  $w^{[2](i)} = w^{[1](i)} \cdot \exp\left(-\hat{\beta}^{[1]} \cdot 1\right)$
  $\approx 0.065.$
- For 3 misclassified observations:
  $w^{[2](i)} = w^{[1](i)} \cdot \exp\left(-\hat{\beta}^{[1]} \cdot (-1)\right)$
  $\approx 0.15.$
- After normalization:
  - correctly classified: $w^{[2](i)} \approx 0.07$
  - misclassified: $w^{[2](i)} \approx 0.17$

ADABOOST ILLUSTRATION



Iteration $m = 2$:

- $\text{err}^{[2]} = 3 \cdot 0.07 = 0.21$
- $\hat{\beta}^{[2]} \approx 0.65$

New observation weights:

- E.g., for 3 misclassified observations: $w^{[3](i)} = w^{[2](i)} \cdot \exp\left(-\hat{\beta}^{[1]} \cdot (-1)\right) \approx 0.14.$
- After normalization: $w^{[3](i)} \approx 0.17$ (misclassified)

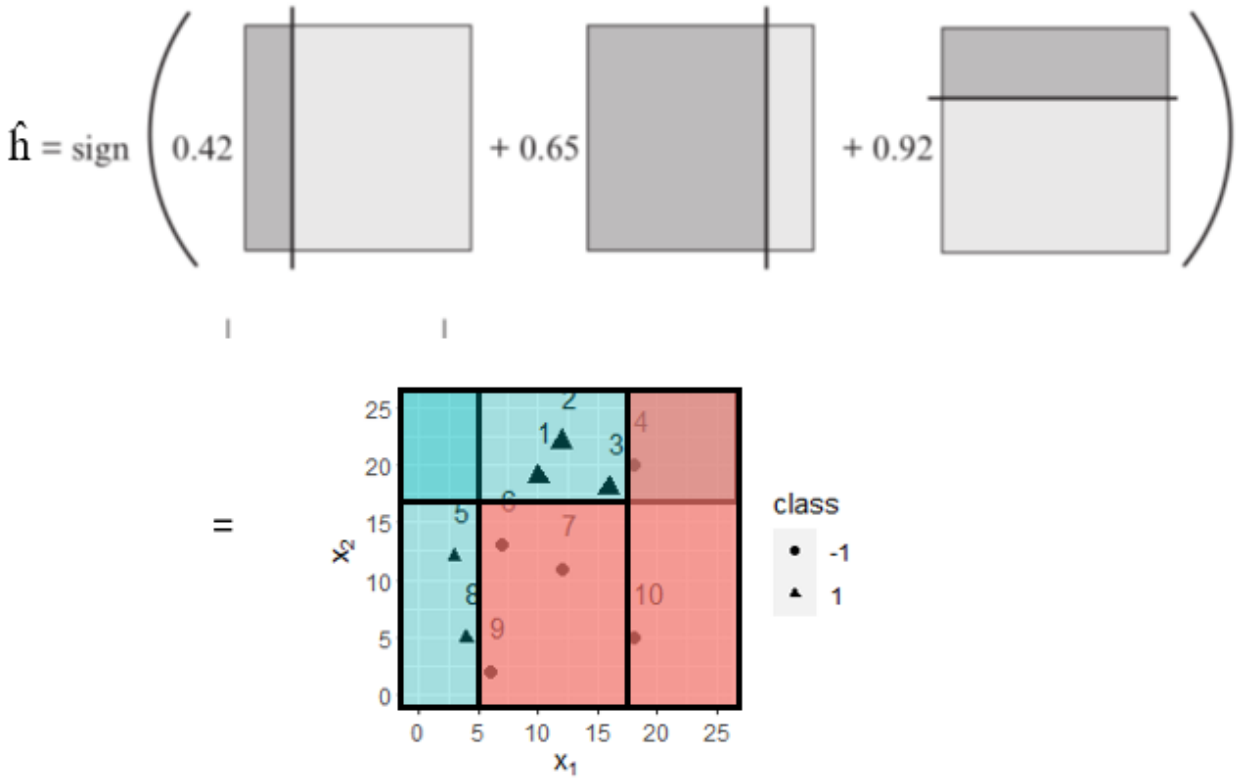
Iteration $m = 3$:

- $\text{err}^{[3]} = 3 \cdot 0.045 \approx 0.14$
- $\hat{\beta}^{[3]} \approx 0.92$

Note: the smaller the error rate of a base learner, the larger the weight, e.g., $\text{err}^{[3]} \approx 0.14 < \text{err}^{[1]} \approx 0.3$ and $\hat{\beta}^{[3]} \approx 0.92 > \hat{\beta}^{[1]} \approx 0.42$.

ADABOOST ILLUSTRATION

With $\hat{f}(\mathbf{x}) = \sum_{m=1}^{M} \hat{\beta}^{[m]} \hat{b}^{[m]}(\mathbf{x})$ and $h(\mathbf{x}) = \text{sign}(f(\mathbf{x})) \in \{-1, +1\}$, we get:



Hence, when all three base classifiers are combined, all samples are classified correctly.

BAGGING VS BOOSTING

Random forest

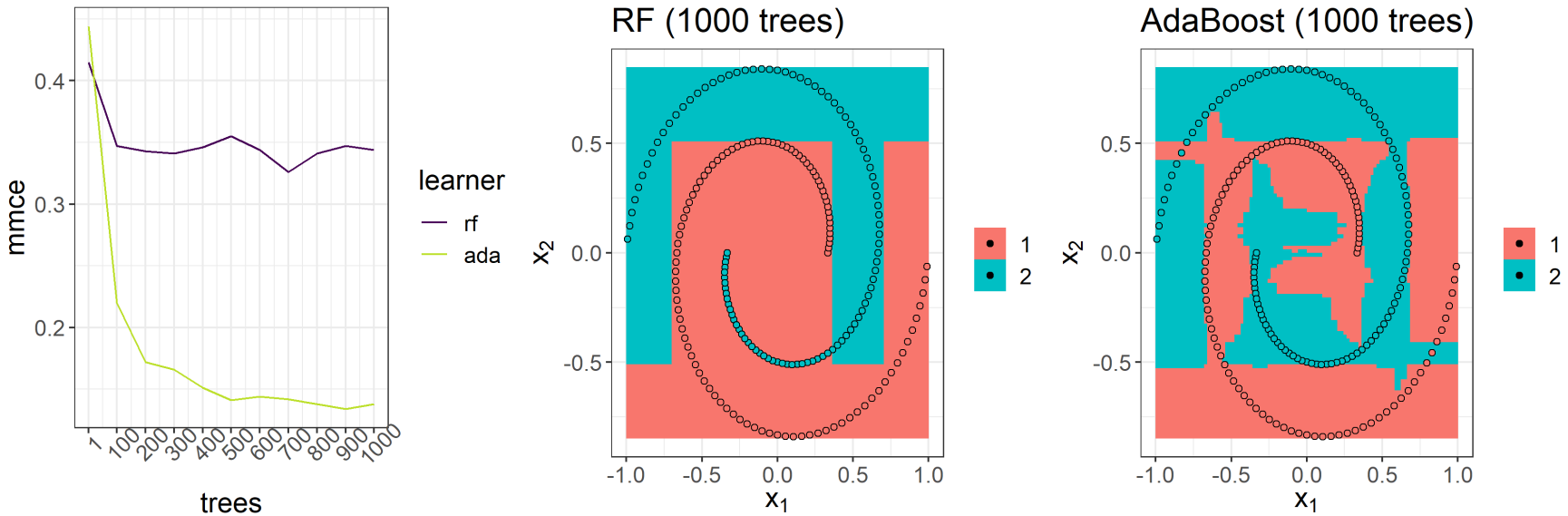
- Base learners are typically deeper decision trees (not only stumps!)
- Equal weights for BL
- BLs fitted independently
- Aim: variance reduction
- Tends **not** to overfit if ensemble size grows

AdaBoost

- BLs are weak learners, e.g., only stumps
- BL are weighted
- Sequential fitting of BLs
- Aim: loss reduction
- Tends to overfit with more iterations

BAGGING VS BOOSTING STUMPS

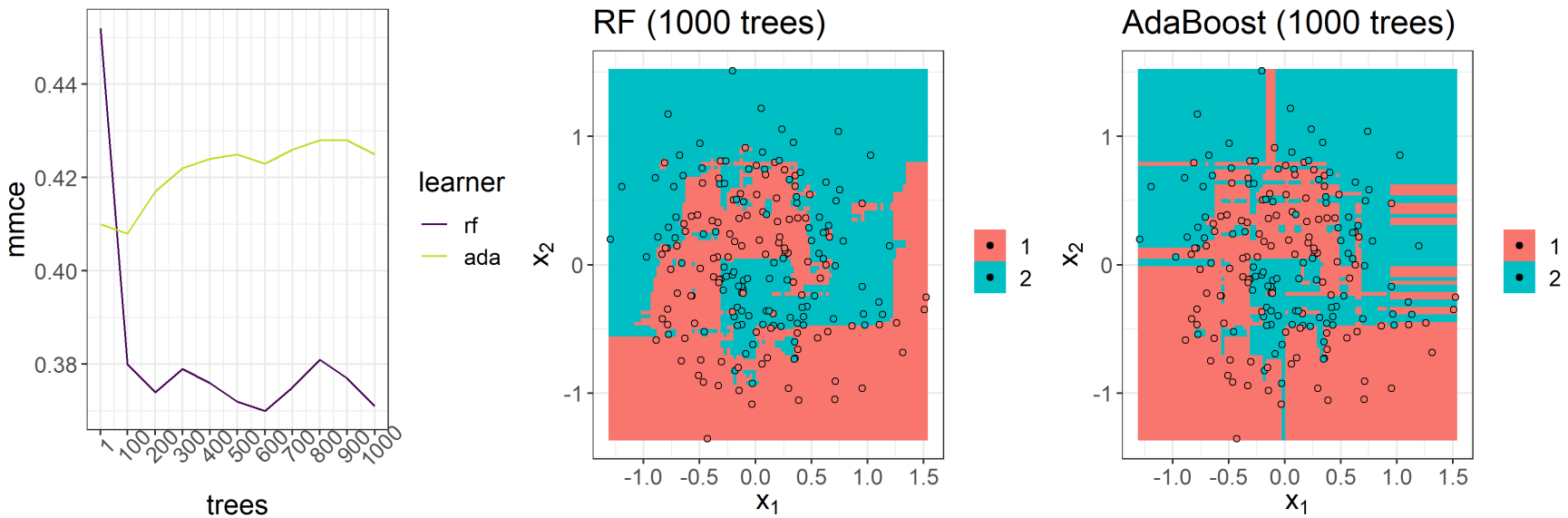
Random forest versus AdaBoost (both with stumps) on `spirals` data from `mlbench` ($n = 200$, $sd = 0$), with 5×5 repeated CV.



Weak learners do not work well with bagging as only variance, but no bias reduction happens.

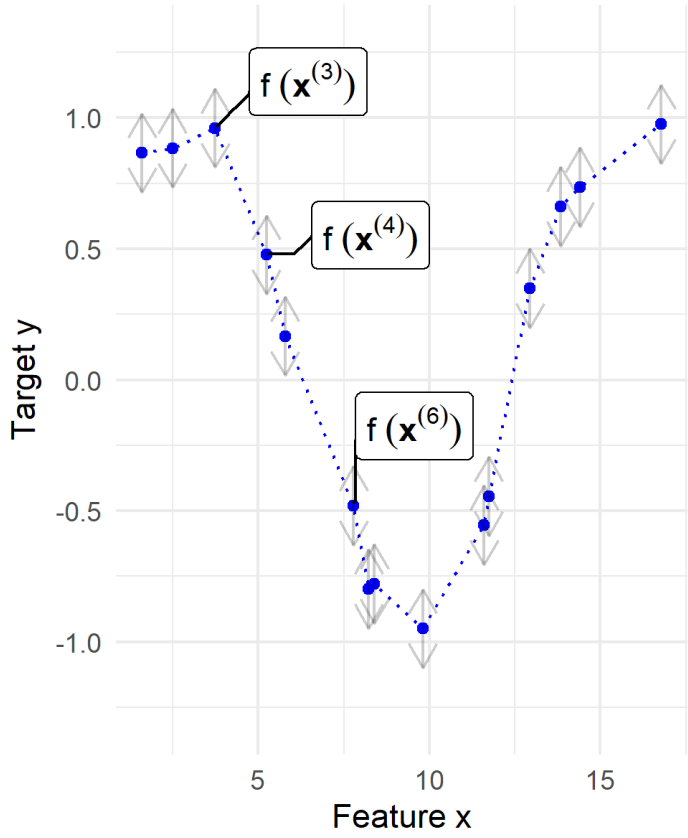
OVERFITTING BEHAVIOR

Historically, the overfitting behavior of AdaBoost was often discussed. Increasing standard deviation in `spirals` to $sd = 0.3$ and allowing for more flexibility in the base learners, AdaBoost overfits with increasing number of trees while the RF only saturates. The overfitting of AdaBoost here is quite typical as data is very noisy.



Einführung in das Statistische Lernen

Gradient Boosting



Learning goals

- Understand idea of forward stagewise modelling
- Understand fitting process of gradient boosting for regression problems

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:

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

Hence, we minimize the empirical risk:

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

FORWARD STAGEWISE ADDITIVE MODELING

Why is gradient boosting a good choice for this problem?

- Because of the additive structure it is difficult to jointly minimize $\mathcal{R}_{\text{emp}}(f)$ w.r.t. $\left(\left(\beta^{[1]}, \boldsymbol{\theta}^{[1]}\right), \ldots, \left(\beta^{[M]}, \boldsymbol{\theta}^{[M]}\right)\right)$, which is a very high-dimensional parameter space (though this is less of a problem nowadays, especially in the case of numeric parameter spaces).
- Considering trees as base learners is worse as we would have to grow M trees in parallel so they work optimally together as an ensemble.
- 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:

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

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

Algorithm 1 Forward Stagewise Additive Modeling.

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

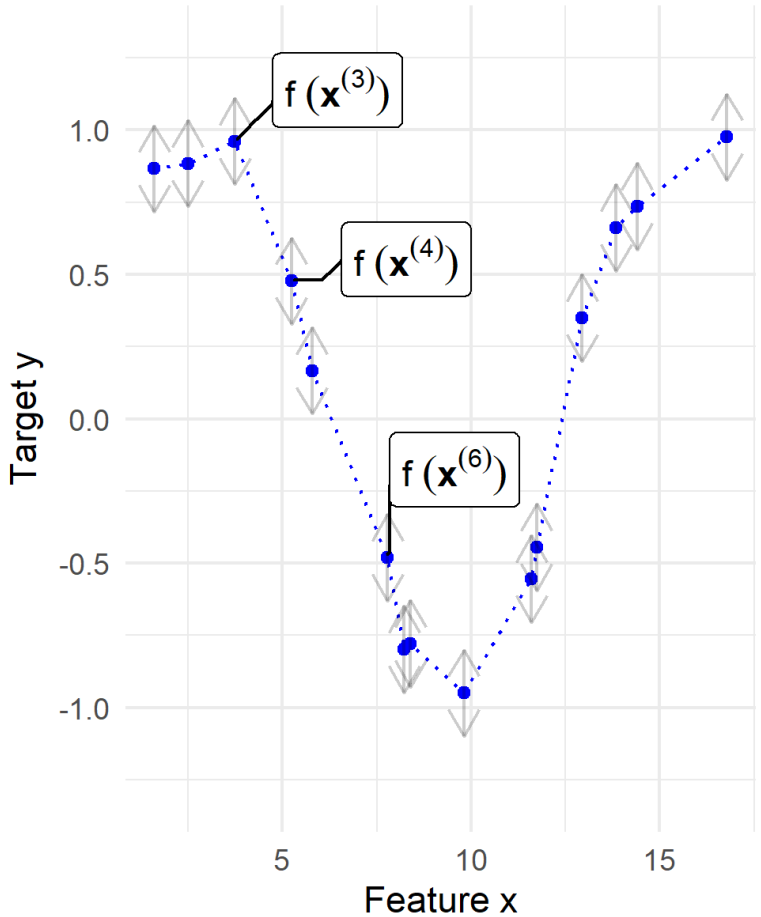
GRADIENT BOOSTING

This is not really an algorithm, but an abstract principle. To find $b\left(\mathbf{x}, \boldsymbol{\theta}^{[m]}\right)$ and $\beta^{[m]}$, we use gradient descent, but in function space!

Consider a model f whose predictions we can arbitrarily define for each training $\mathbf{x}^{(i)}$, i.e., f is a finite vector

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

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

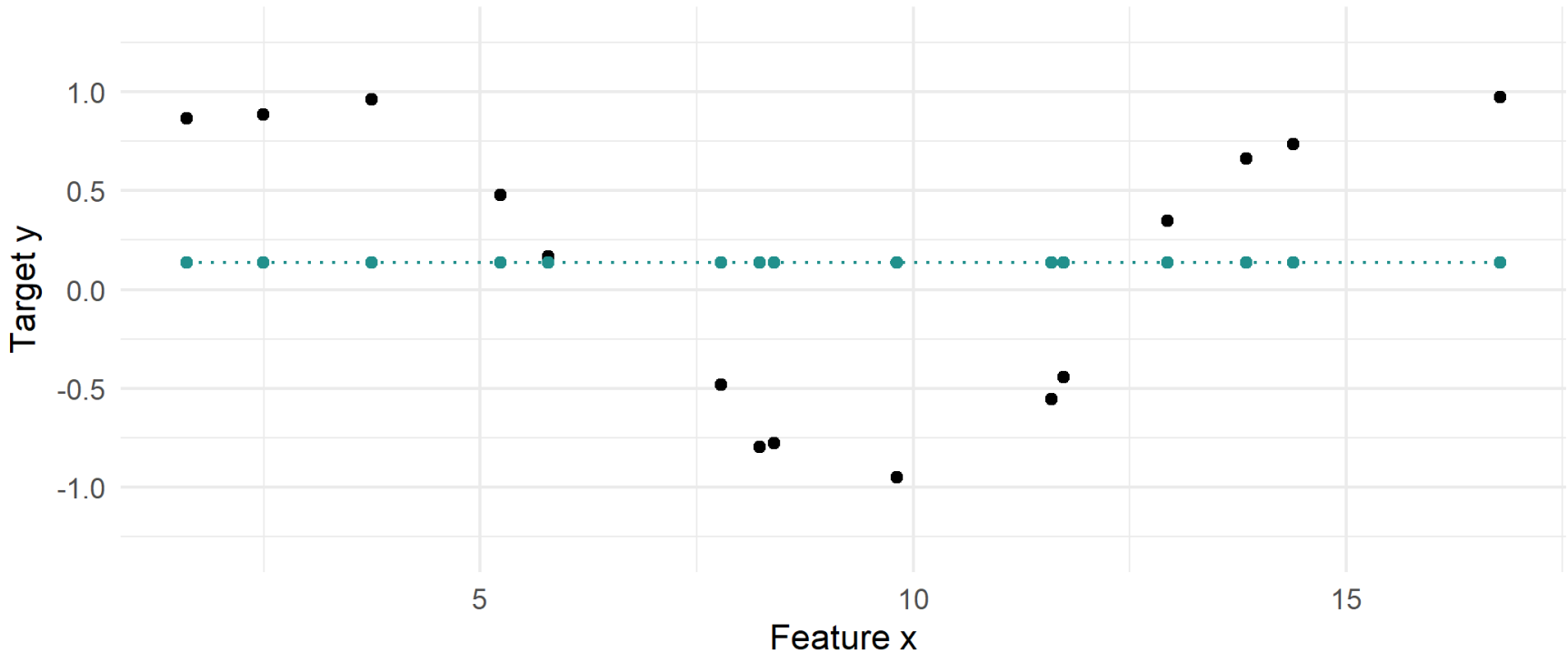


GRADIENT BOOSTING

Aim: Define a movement in function space so we can push our current function towards the data points.

Given: Regression problem with one feature x and target variable y.

Initialization: Set all parameters to the optimal constant value (e.g., the mean of y for $L2$).



PSEUDO RESIDUALS

How do we distort our f to move it towards the labels and reduce risk? Let's minimize risk with GD.

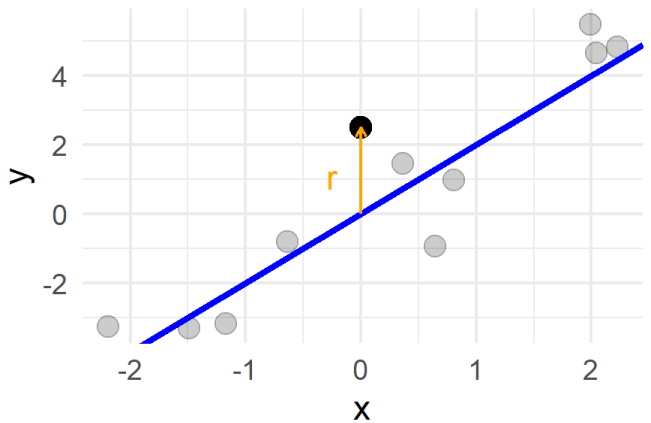
So, we calculate the (negative) gradient of the risk for each parameter, which (weirdly) here are the outputs $f\left(\mathbf{x}^{(i)}\right)$ (0 if $i \neq j$):

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

At each point, we would like to change the output via: $\tilde{r}(f) = -\frac{\partial L(y,f)}{\partial f}$.

L2 Example: The PRs match the usual residuals:

$$\tilde{r}(f) = -\frac{\partial 0.5(y-f)^2}{\partial f} = y - f$$



BOOSTING AS GRADIENT DESCENT

Combining this with "forward stagewise modeling", we are at $f^{[m-1]}$ during minimization. Here, we calculate the direction of the negative gradient or vector of PRs:

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

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

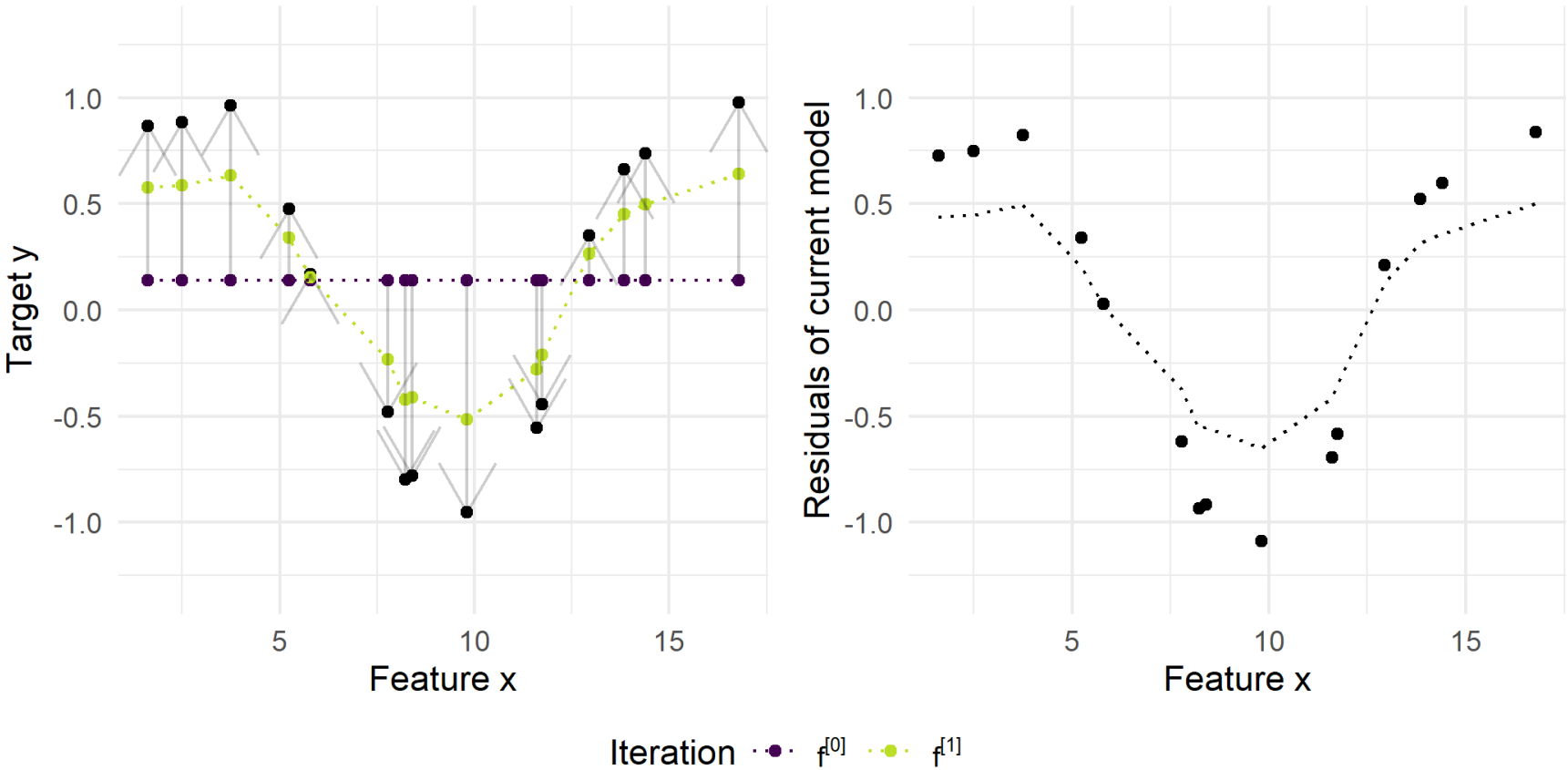
$$f^{[m]}(\mathbf{x}^{(i)}) = f^{[m-1]}(\mathbf{x}^{(i)}) + \beta \tilde{r}^{[m](i)}.$$

Like this, we should "nudge" f in the direction best risk reduction.

GRADIENT BOOSTING

Iteration 1:

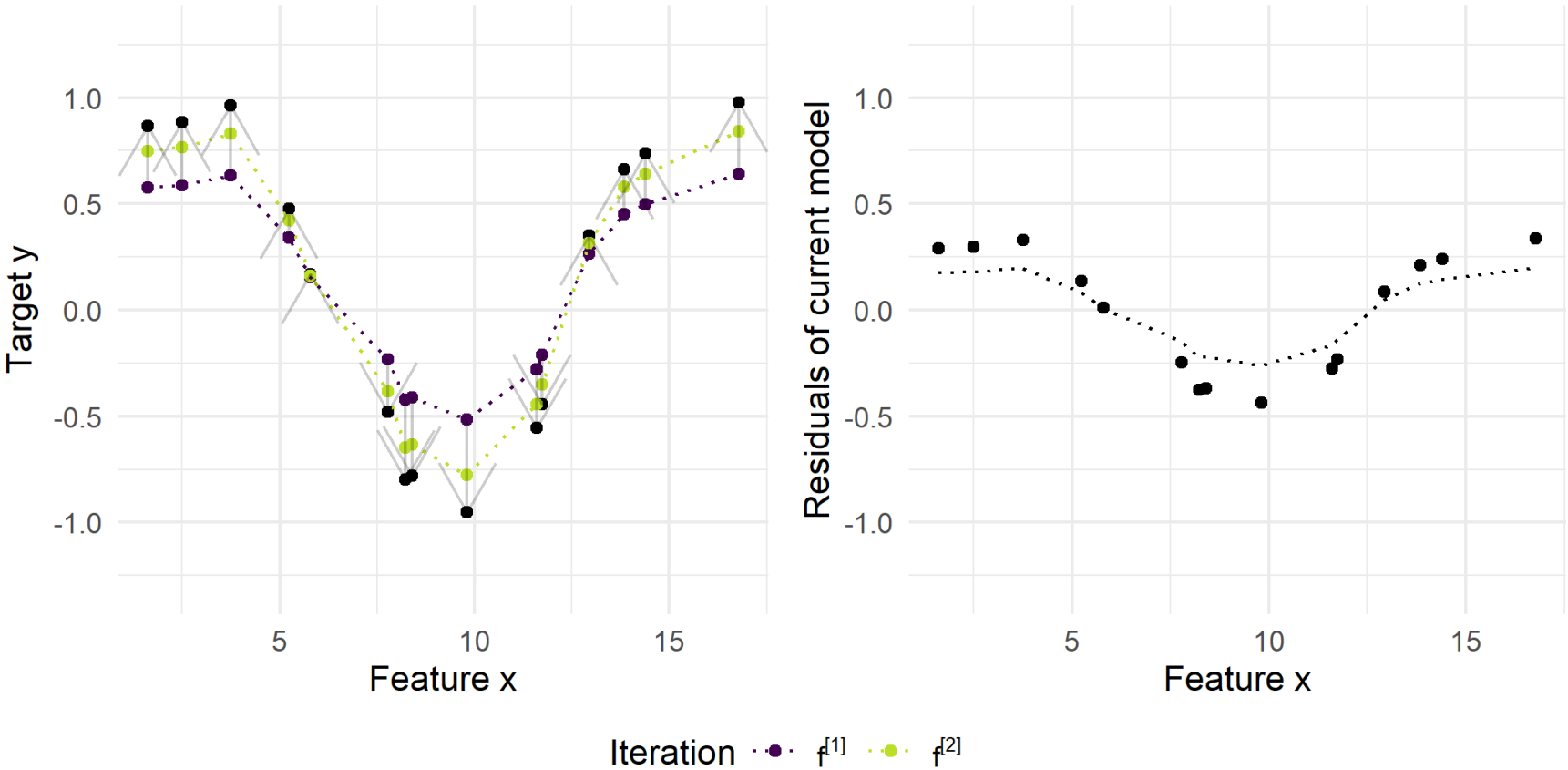
Let's move our function $f\left(\mathbf{x}^{(i)}\right)$ a fraction towards the pseudo-residuals with a learning rate of $\beta = 0.6$.



GRADIENT BOOSTING

Iteration 2:

Let's move our function $f\left(\mathbf{x}^{(i)}\right)$ a fraction towards the pseudo-residuals with a learning rate of $\beta = 0.6$.



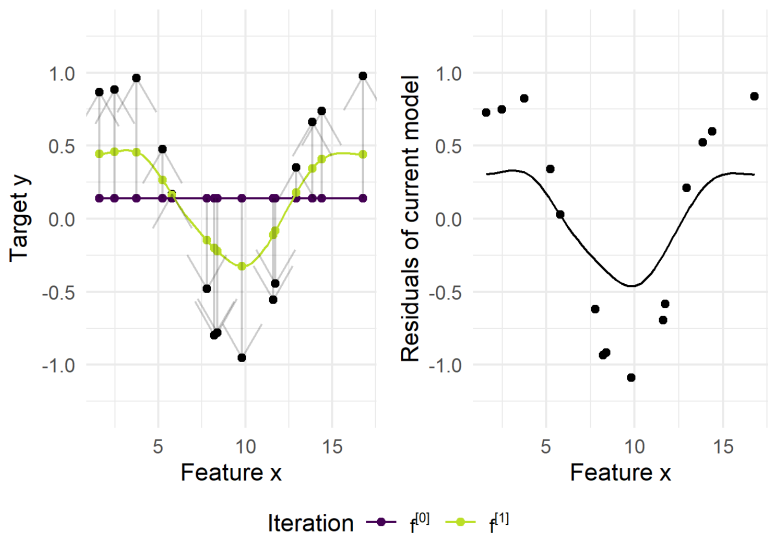
GRADIENT BOOSTING

As said, such a model parameterization is pointless.

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

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

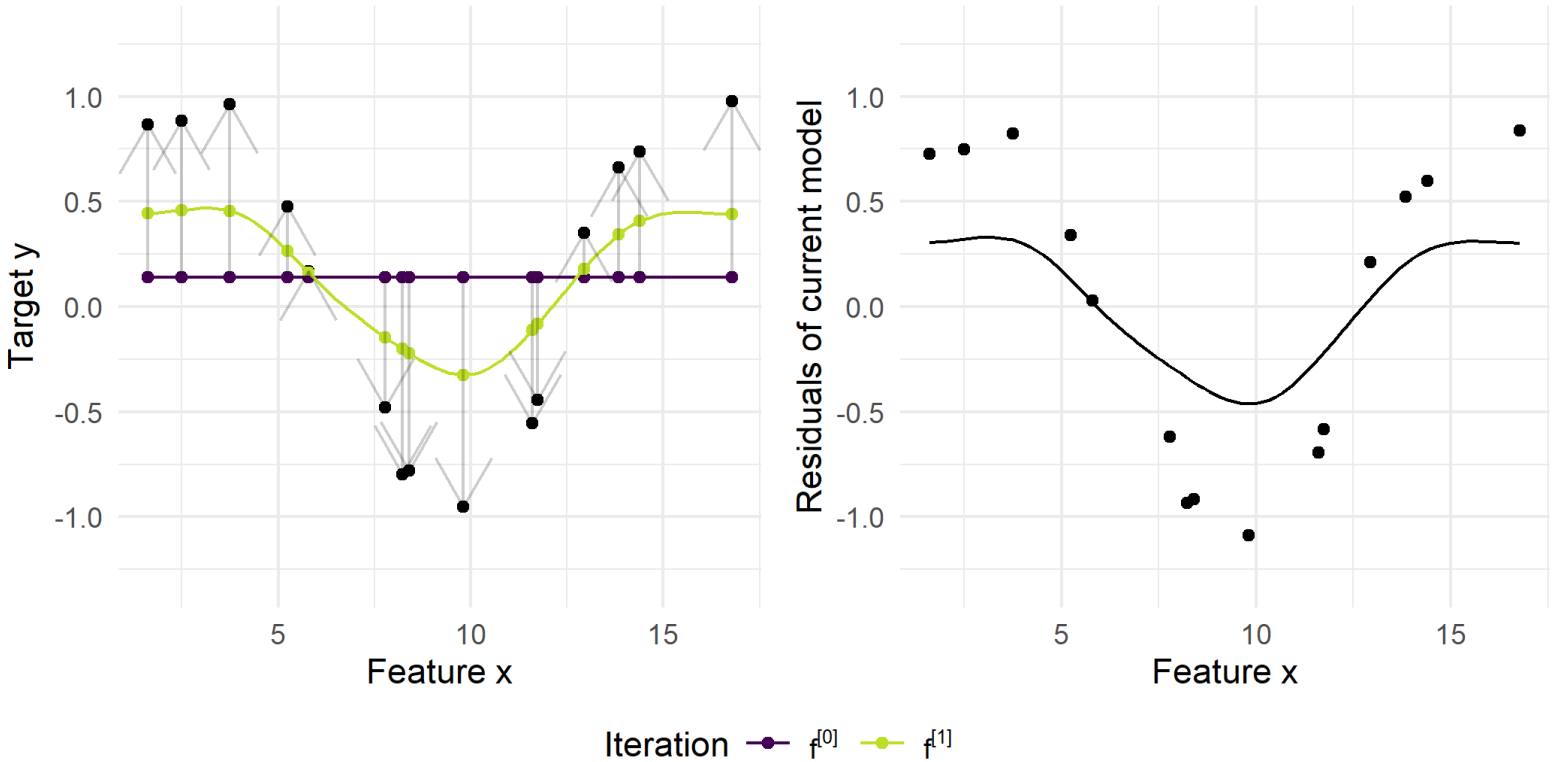
So, evaluated on the training data, $b(\mathbf{x}, \boldsymbol{\theta}^{[m]})$ corresponds as closely as possible to the negative risk gradient and generalizes over $\mathcal{X}$.



GRADIENT BOOSTING

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

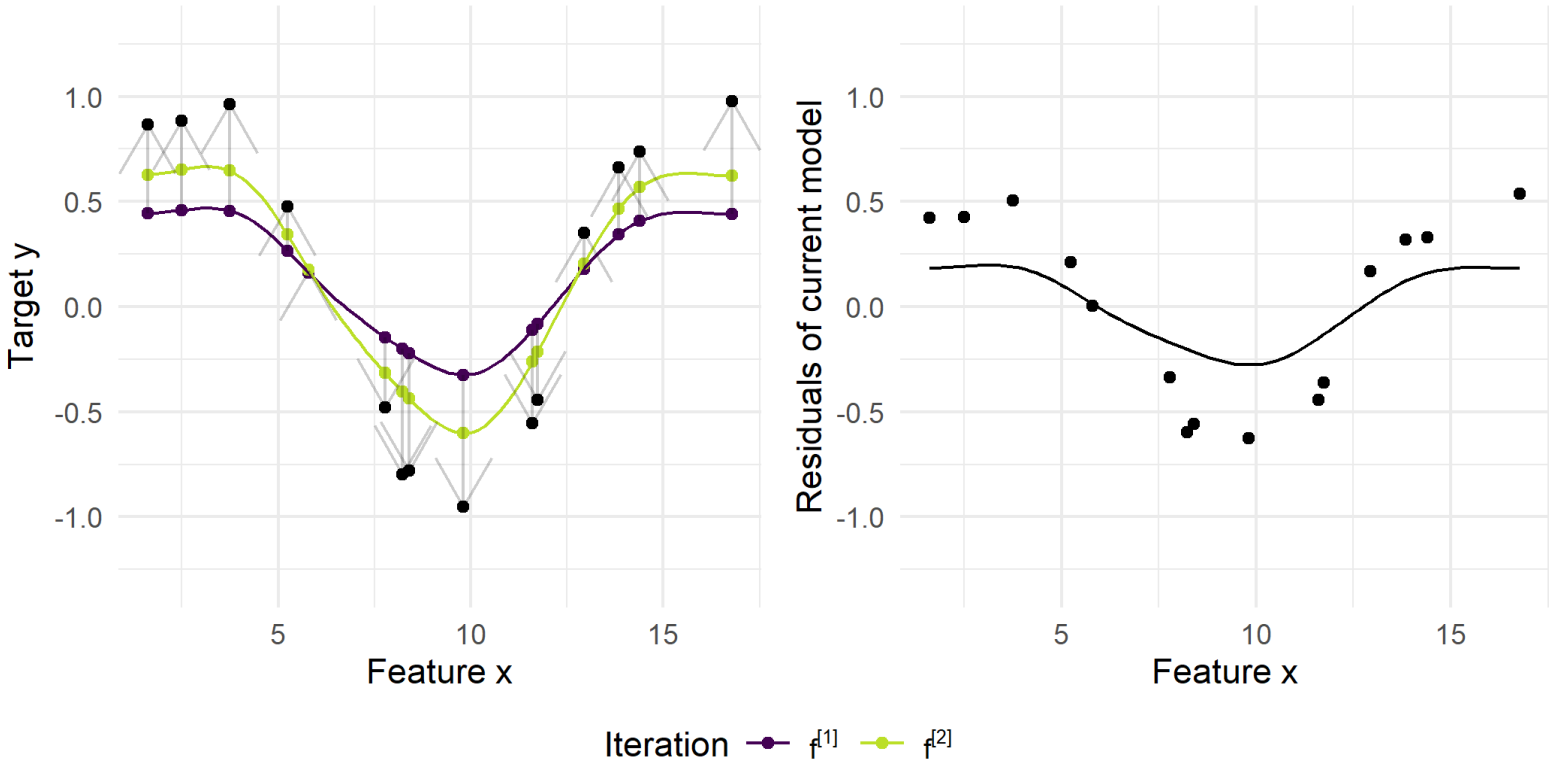
Iteration 1:



GRADIENT BOOSTING

Instead of moving the function values for each observation by a fraction closer to the observed data, we fit a regression base learner to the pseudo-residuals (right plot).

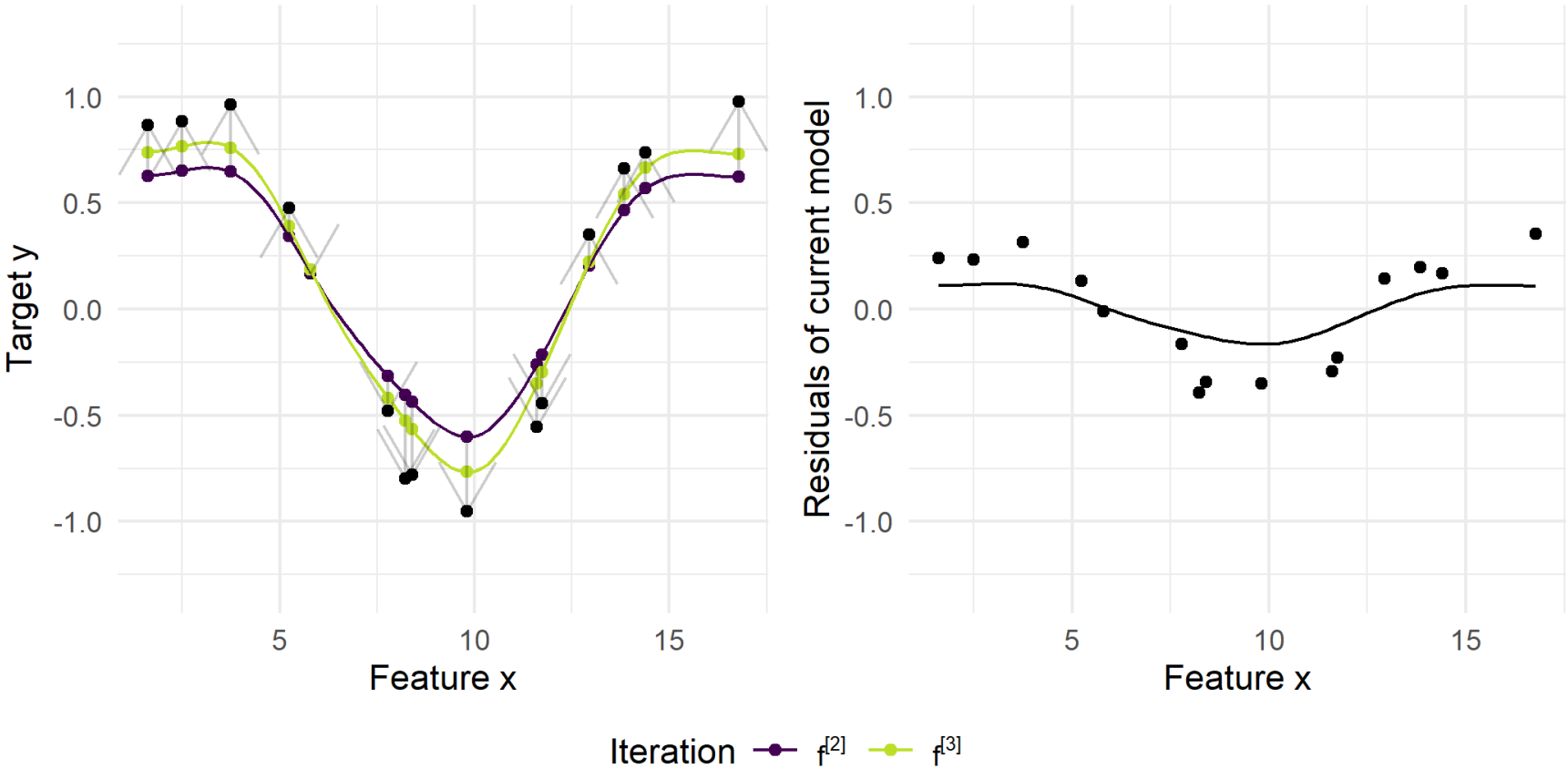
Iteration 2:



GRADIENT BOOSTING

This BL is added to the of the ensemble, weighted by a learning rate (here: $\beta = 0.4$). Then we iterate.

Iteration 3:



GRADIENT BOOSTING ALGORITHM

Algorithm 2 Gradient Boosting Algorithm.

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

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

LINE SEARCH

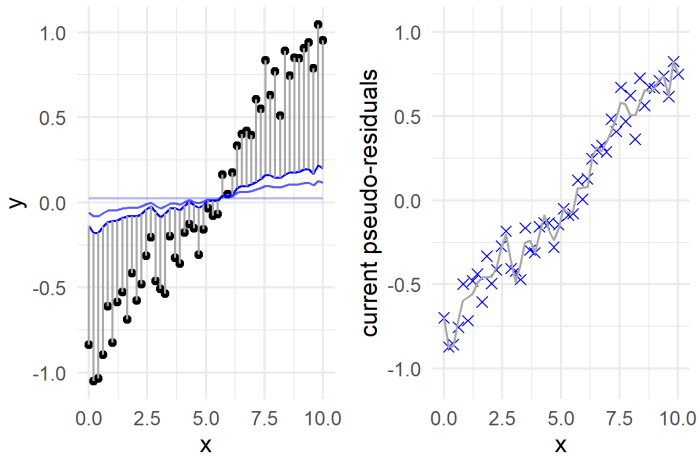
The learning rate, as always, influences how we converge. Although a small constant LR is commonly used, we can also run line search,

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

Alternatively, an (inexact) backtracking line search can be used to find the $\beta^{[m]}$ that minimizes the above equation.

Einführung in das Statistische Lernen

Gradient Boosting - Illustration



Learning goals

- Understand impact of different loss functions and
- Understand impact of different base learners for regression

GRADIENT BOOSTING ILLUSTRATION - GAM

We now compare different loss functions and base learners. We start with a GAM as base learner and compare the *L2* loss with the *L*1 loss.

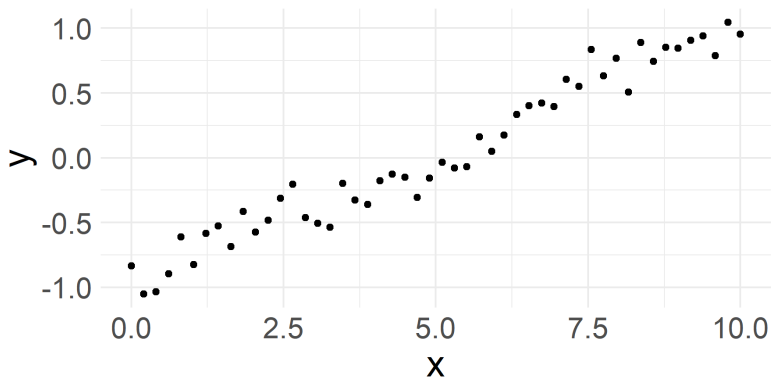
Reminder: Pseudo-residuals

- *L2*: $\tilde{r}(f) = r(f) = y - f(\mathbf{x})$
- *L*1: $\tilde{r}(f) = sign(y - f(\mathbf{x}))$

We consider a regression task with a single feature x and target y, with the following true underlying relationship:

$$y^{(i)} = -1 + 0.2 \cdot x^{(i)} + 0.1 \cdot sin(x^{(i)}) + \epsilon^{(i)}$$

$$\text{with } n = 50 \text{ and } \epsilon^{(i)} \sim \mathcal{N}(0, 0.1) \quad \forall i \in \{1, \ldots, n\}$$

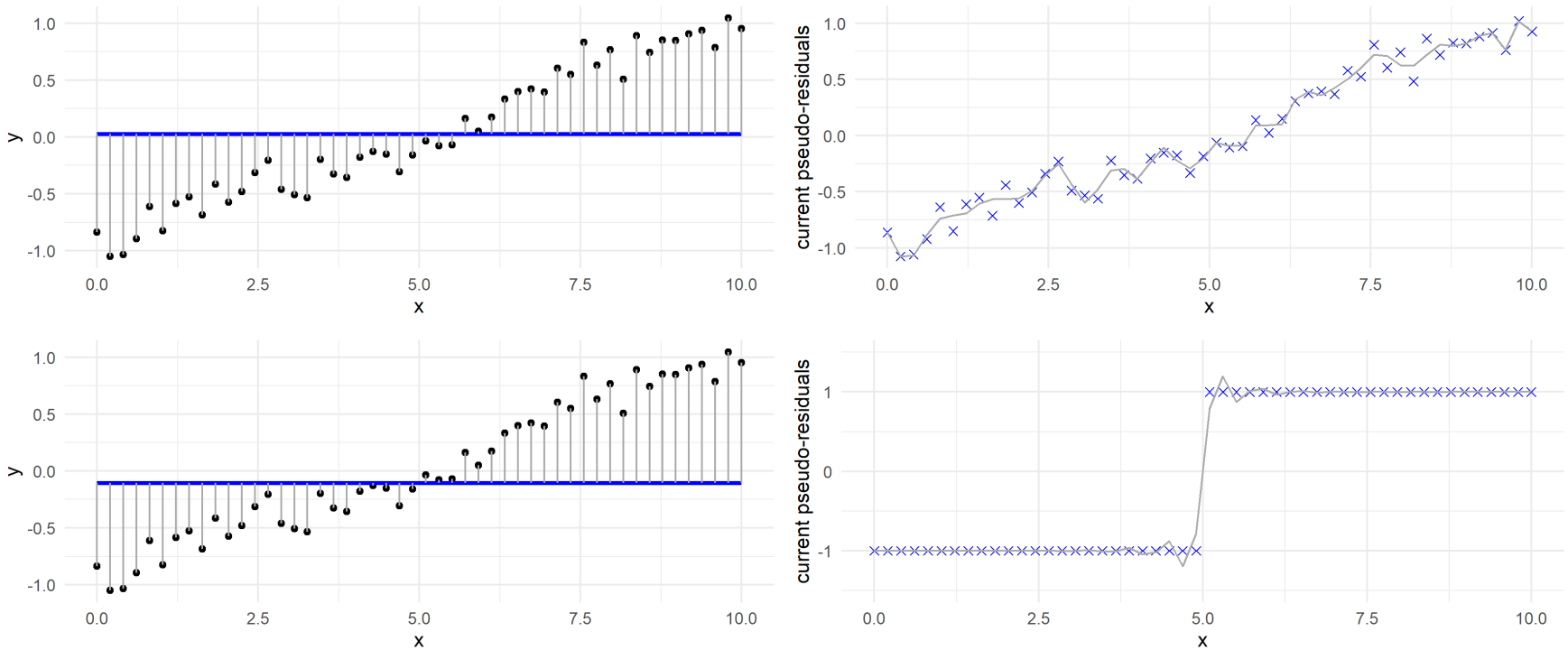


GRADIENT BOOSTING ILLUSTRATION - GAM

1. We start with the simplest model, the optimal constant – mean of the target variable in the case of *L2* loss and median in the case of $L1$ loss.
2. We improve the model by calculating the pointwise pseudo-residuals on the training data, and fit a GAM on the residuals.
    1. The GAM base learners model the conditional mean via cubic *B*-splines with 40 knots.
    2. In each step, the GAM fitted on the current pseudo-residuals is multiplied by a constant learning rate of 0.2 and added to the previous model.
    3. This procedure is repeated multiple times.

GAM WITH *L2* VS *L*1 LOSS

Top: *L2* loss, bottom: *L*1 loss

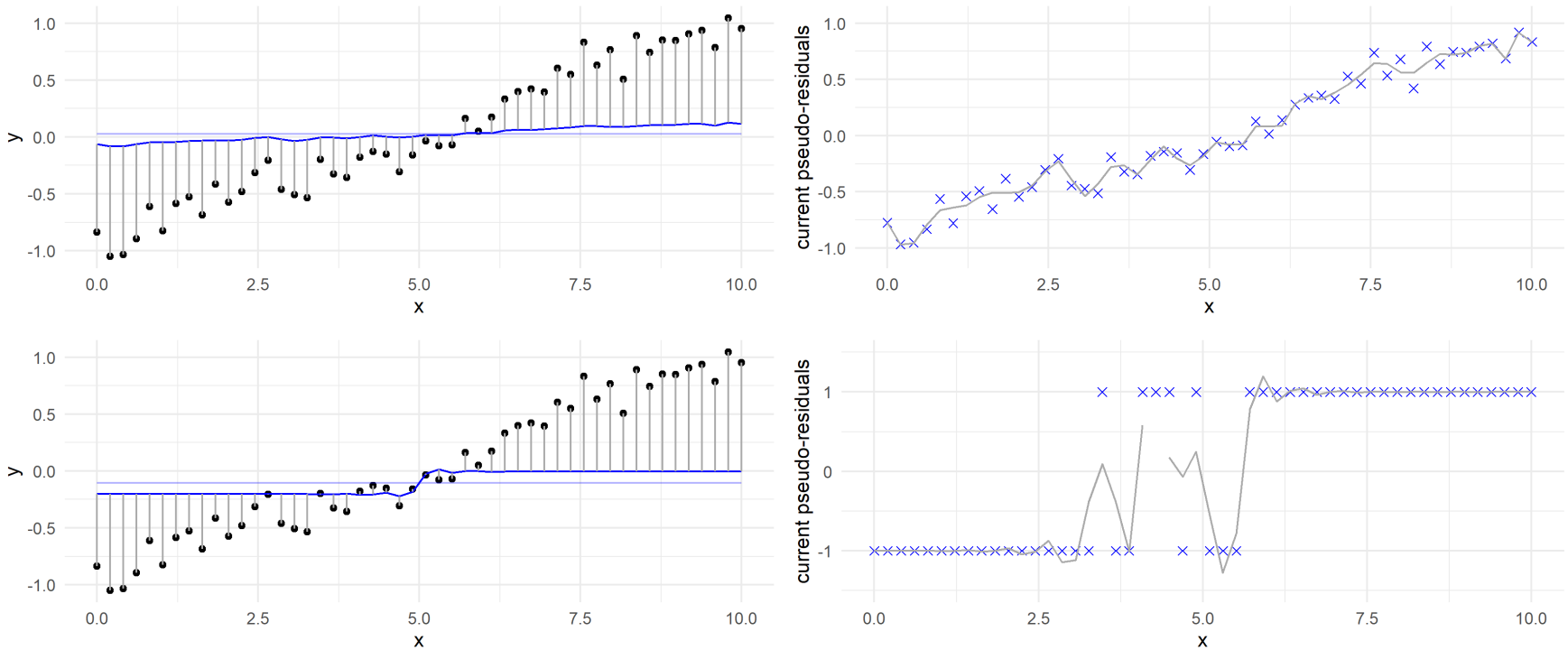


Iteration 1

The nature of the pseudo-residuals affects gradual model fit: as *L*1 only considers residuals' sign, the corresponding base learners are less affected by very large or small values compared to *L2* and hence lead to more moderate changes.

GAM WITH *L2* VS *L1* LOSS

Top: *L2* loss, bottom: *L1* loss

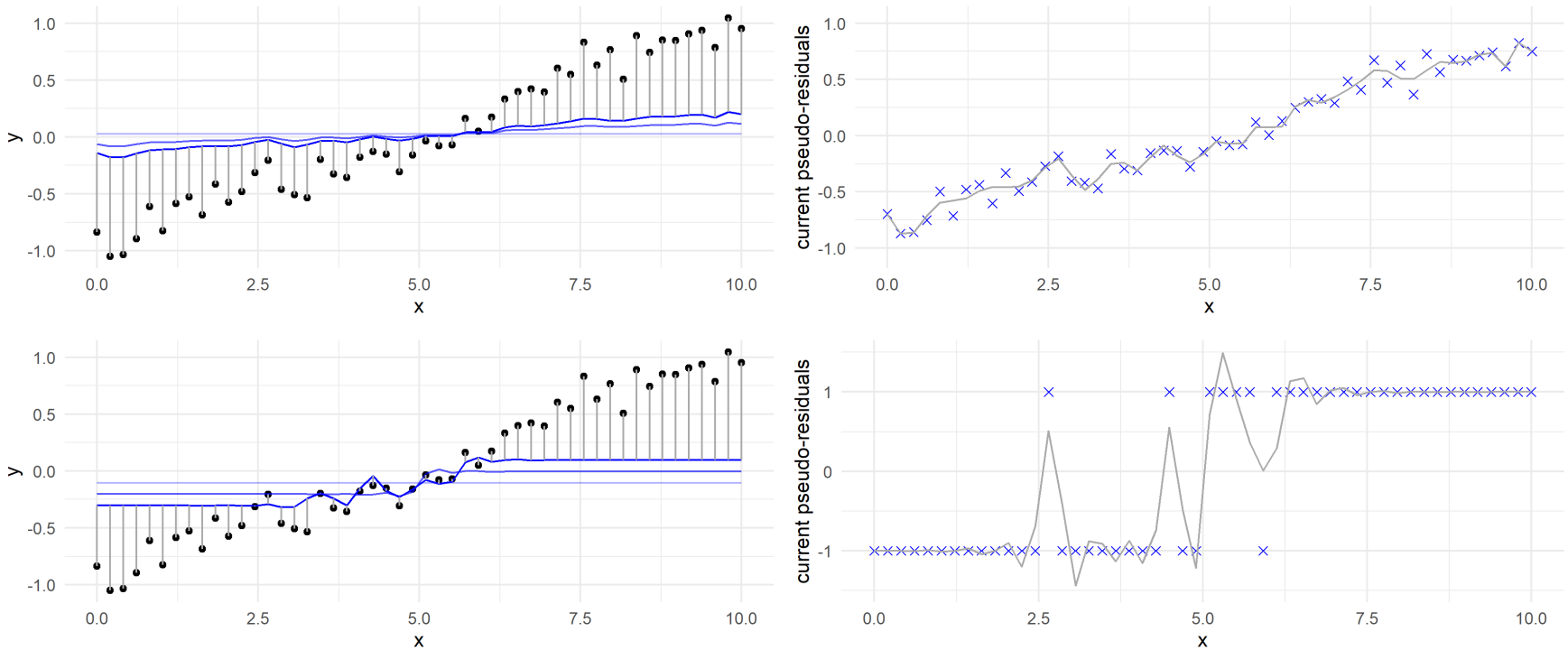


Iteration 2

The nature of the pseudo-residuals affects gradual model fit: as *L1* only considers residuals' sign, the corresponding base learners are less affected by very large or small values compared to *L2* and hence lead to more moderate changes.

GAM WITH *L2* VS *L1* LOSS

Top: *L2* loss, bottom: *L1* loss

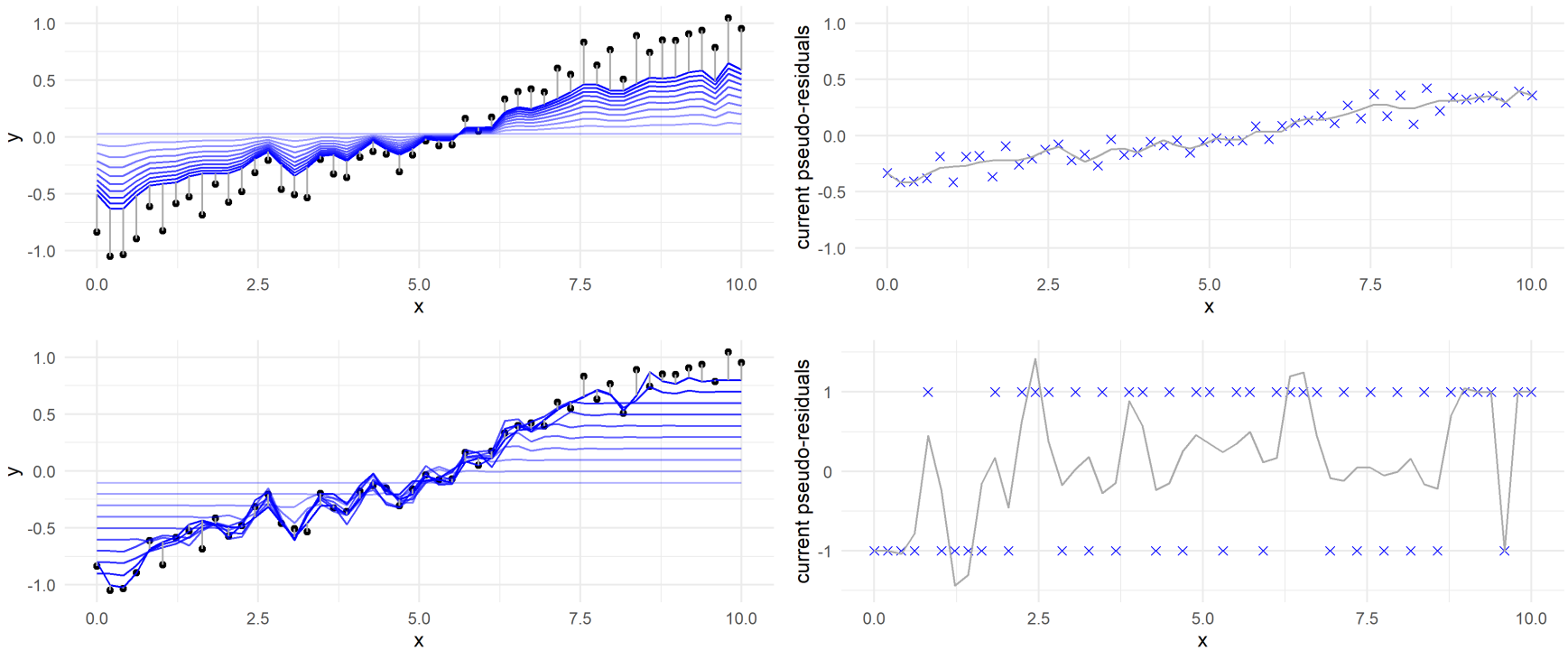


Iteration 3

The nature of the pseudo-residuals affects gradual model fit: as *L1* only considers residuals' sign, the corresponding base learners are less affected by very large or small values compared to *L2* and hence lead to more moderate changes.

GAM WITH *L2* VS *L1* LOSS

Top: *L2* loss, bottom: *L1* loss

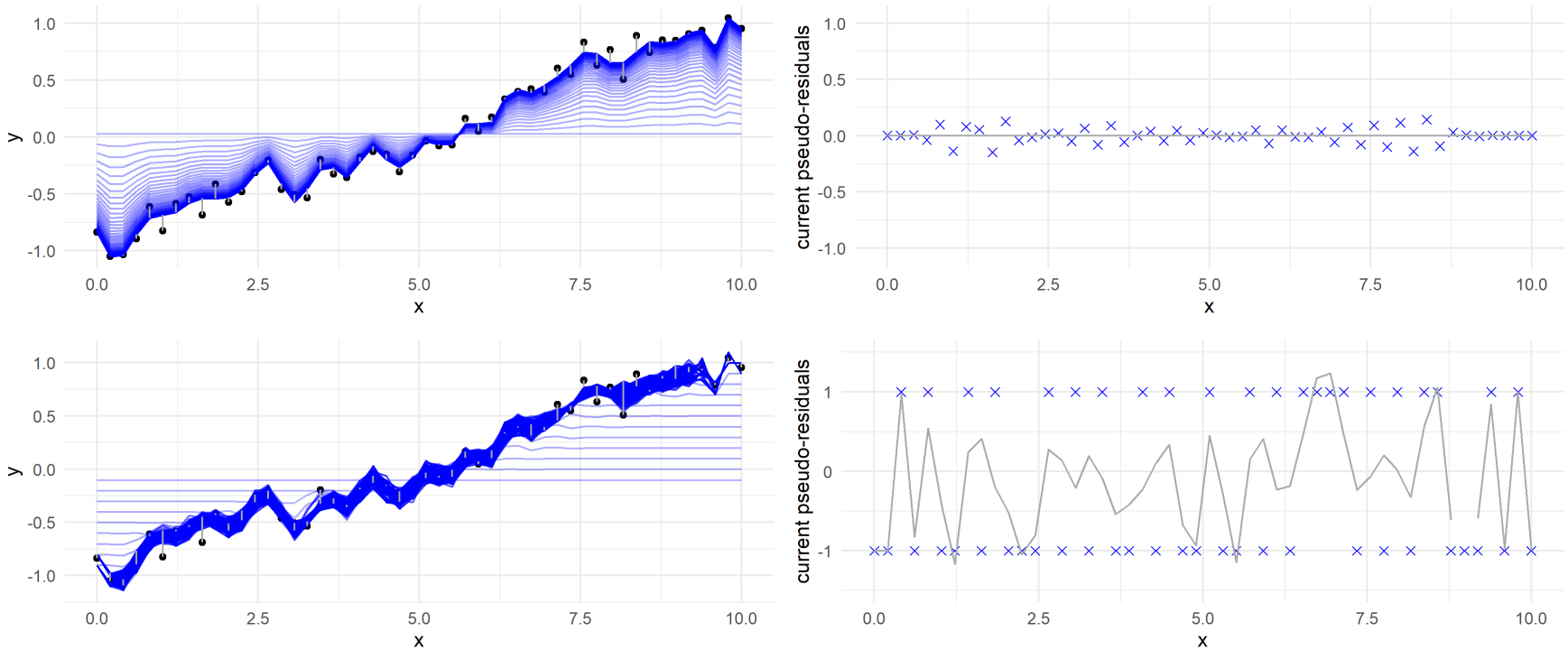


Iteration 10

The nature of the pseudo-residuals affects gradual model fit: as *L1* only considers residuals' sign, the corresponding base learners are less affected by very large or small values compared to *L2* and hence lead to more moderate changes.

GAM WITH *L2* VS *L1* LOSS

Top: *L2* loss, bottom: *L1* loss

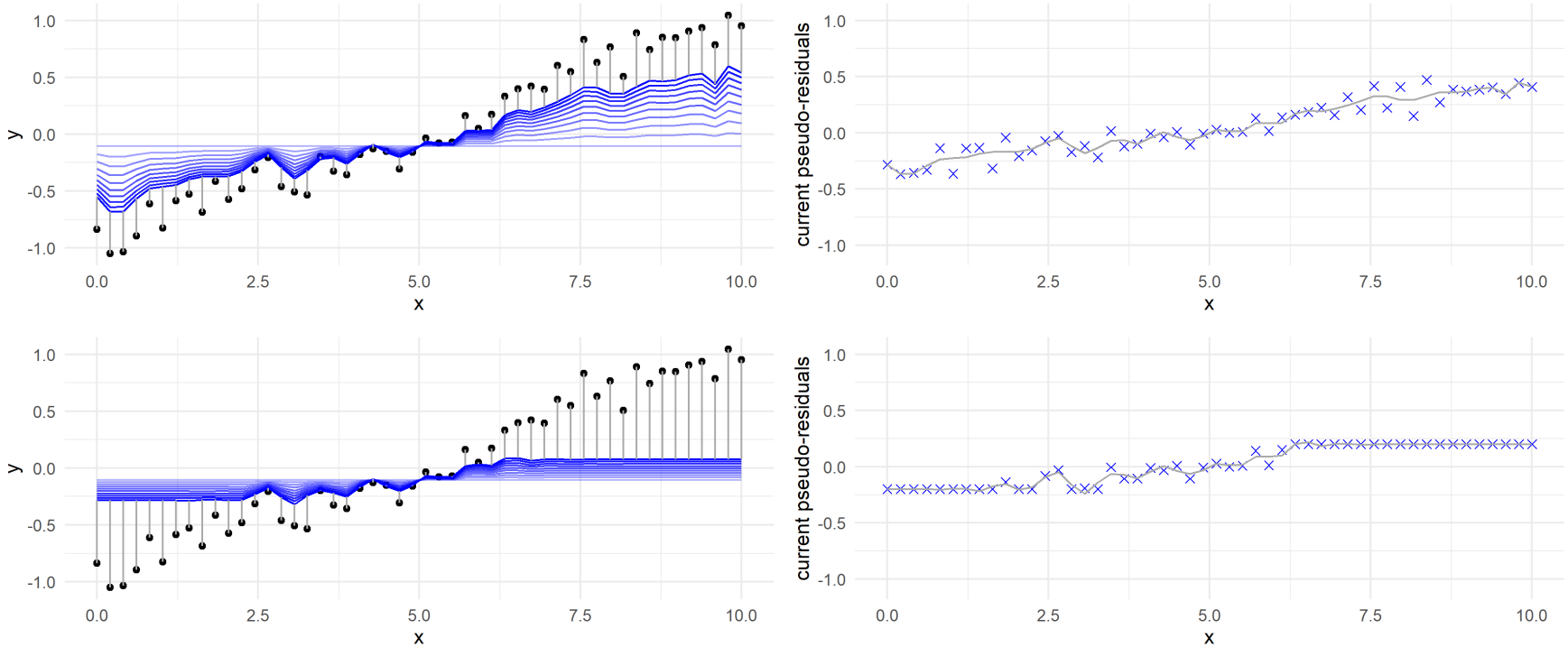


Iteration 100

The nature of the pseudo-residuals affects gradual model fit: as *L1* only considers residuals' sign, the corresponding base learners are less affected by very large or small values compared to *L2* and hence lead to more moderate changes.

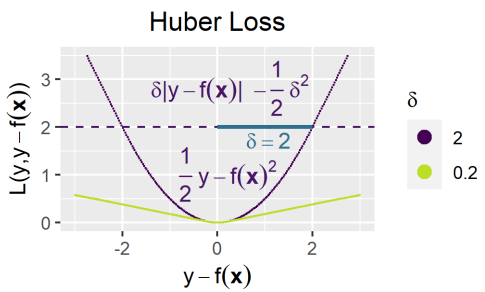
GAM WITH HUBER LOSS

We can also use Huber loss, which is closer to $L2$ for large δ values and closer to $L1$ for smaller δ values. Top: $\delta = 2$, bottom: $\delta = 0.2$.



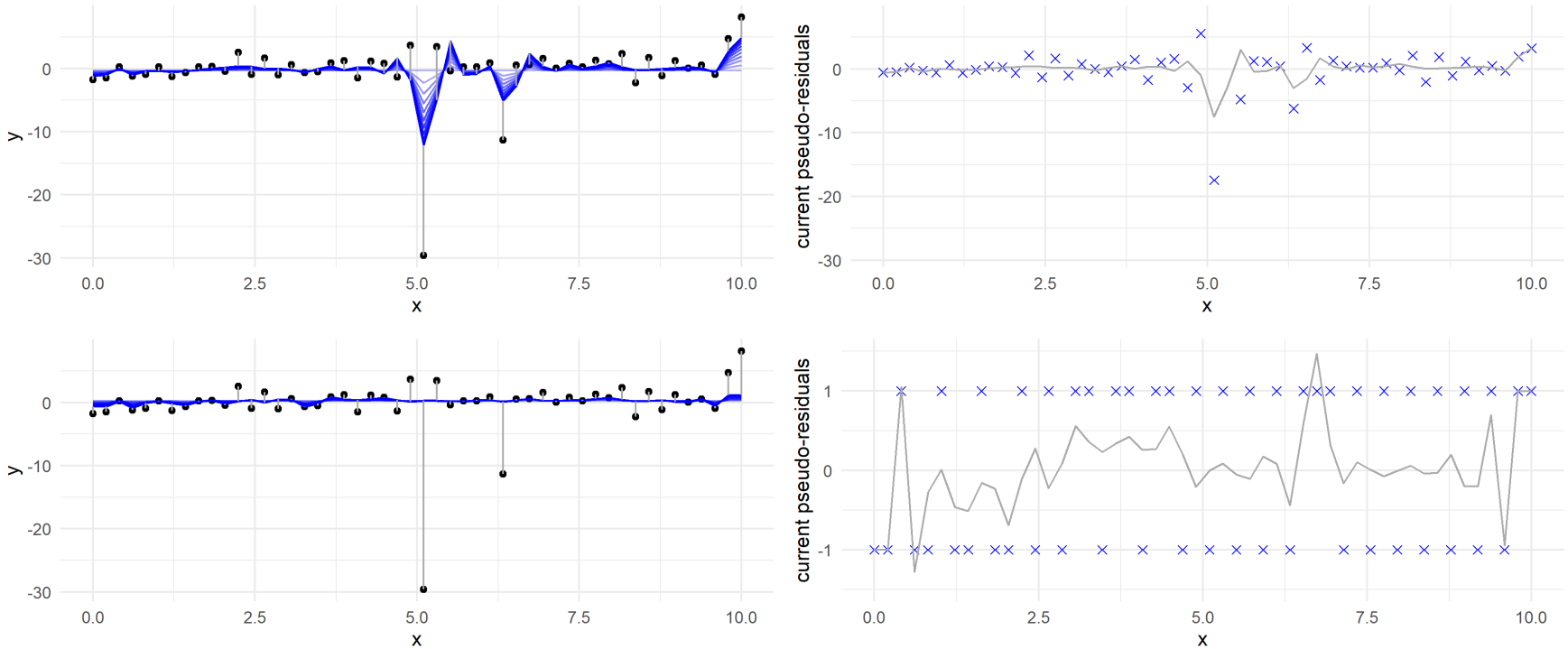
Iteration 10

We see how for smaller δ (bottom) pseudo-residuals are often effectively bounded, resulting in $L1$-like behavior, while the upper plot more closely resembles $L2$ loss.



GAM WITH OUTLIERS

Instead of normally distributed noise we can assume a t-distribution, leading to outliers in the observed target values. Top: $L2$, bottom: $L1$.

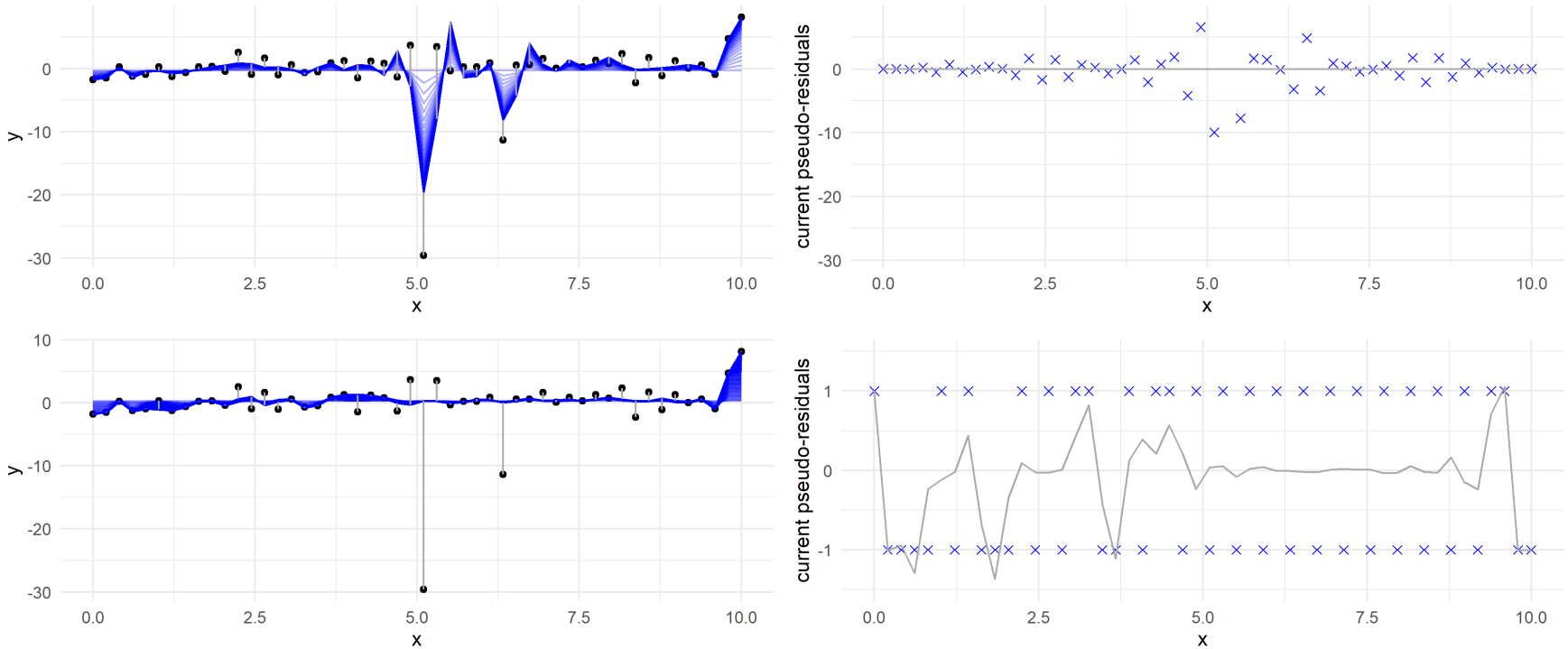


Iteration 10

$L2$ loss is affected by outliers rather strongly, whereas $L1$ solely considers residuals' sign and not their magnitude, resulting in a more robust model.

GAM WITH OUTLIERS

Instead of normally distributed noise we can assume a t-distribution, leading to outliers in the observed target values. Top: $L2$, bottom: $L1$.

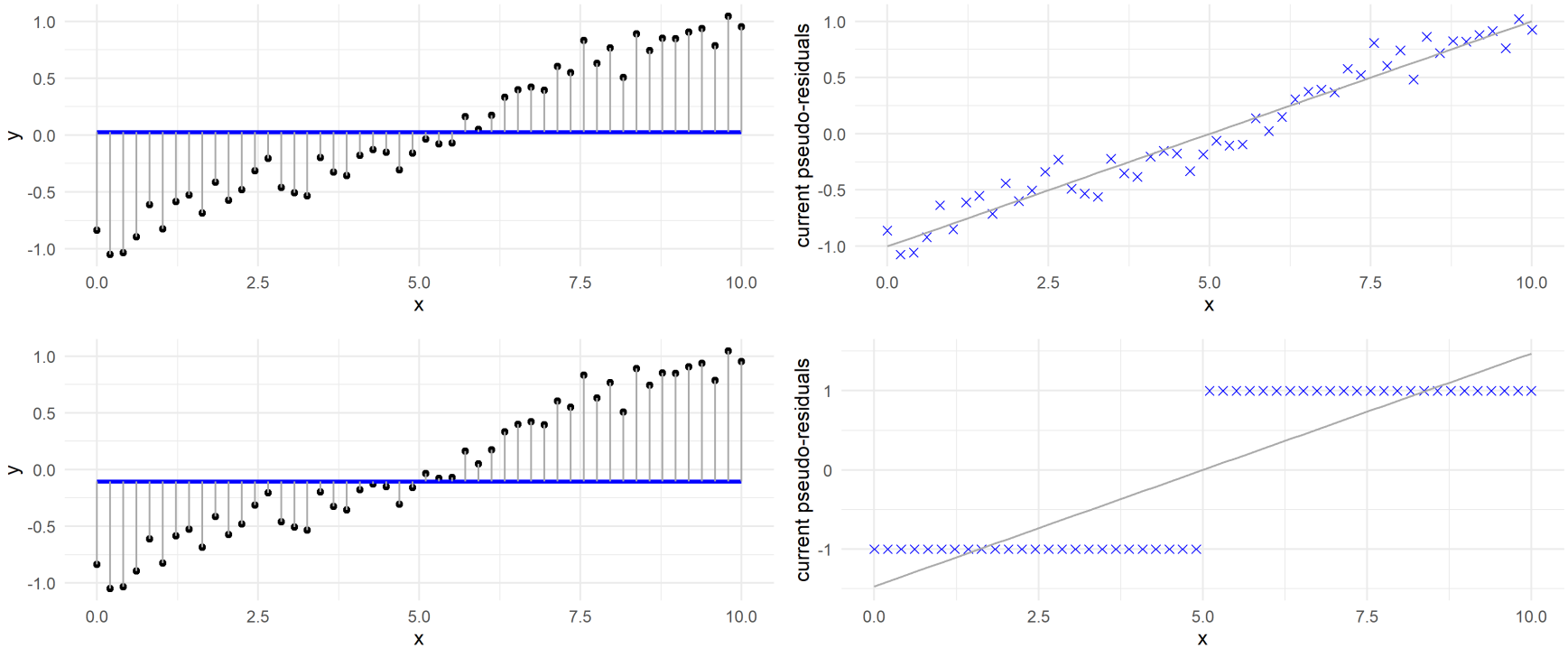


Iteration 100

$L2$ loss is affected by outliers rather strongly, whereas $L1$ solely considers residuals' sign and not their magnitude, resulting in a more robust model.

LM WITH $L2$ VS $L1$ LOSS

Instead of using a GAM as base learner we now use a simple linear model. Top: $L2$, bottom: $L1$.

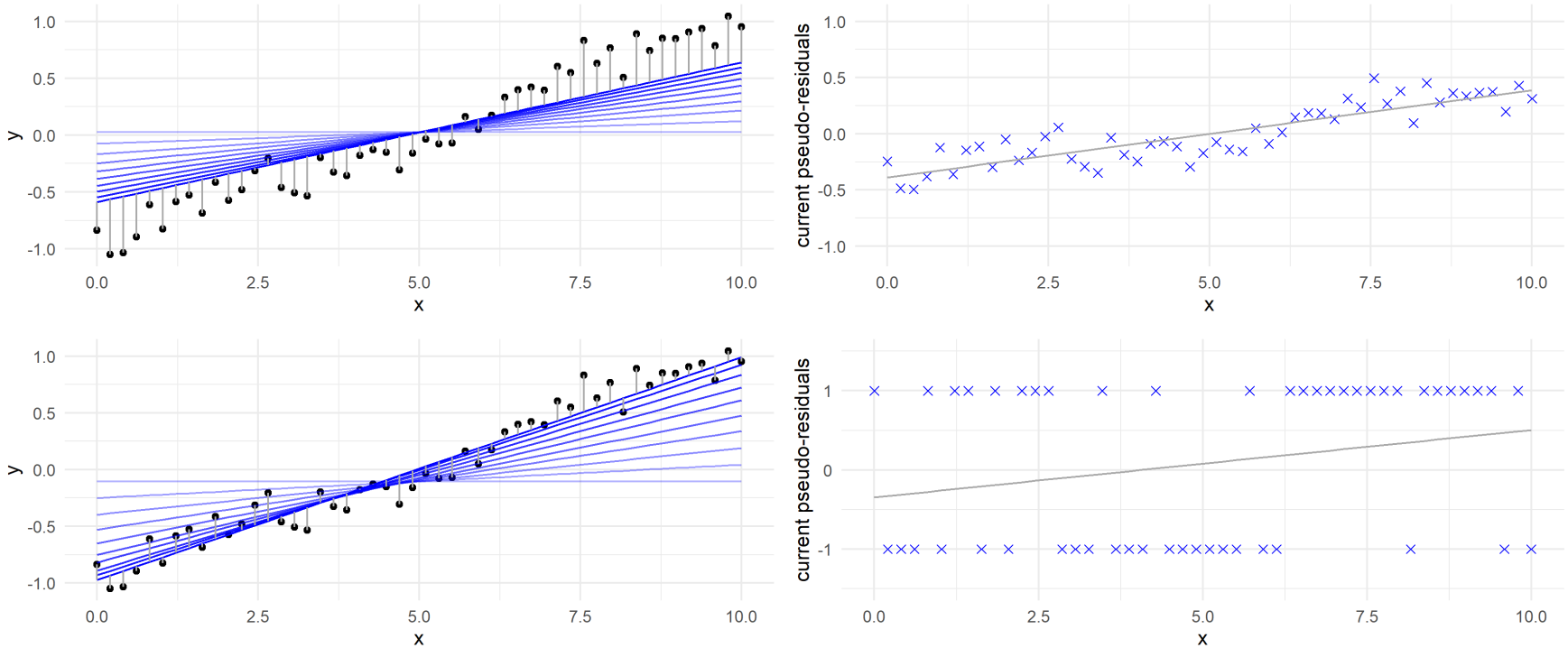


Iteration 1

For $L2$, as $\tilde{r}(f) = r(f)$, we find the optimal model in the very first iteration; only the multiplicative learning rate slows down optimization. In the $L1$ case the base learner LMs fit pseudo-residuals that differ from model residuals, leading to a less monotonic optimization path.

LM WITH *L2* VS *L1* LOSS

Instead of using a GAM as base learner we now use a simple linear model. Top: *L2*, bottom: *L1*.

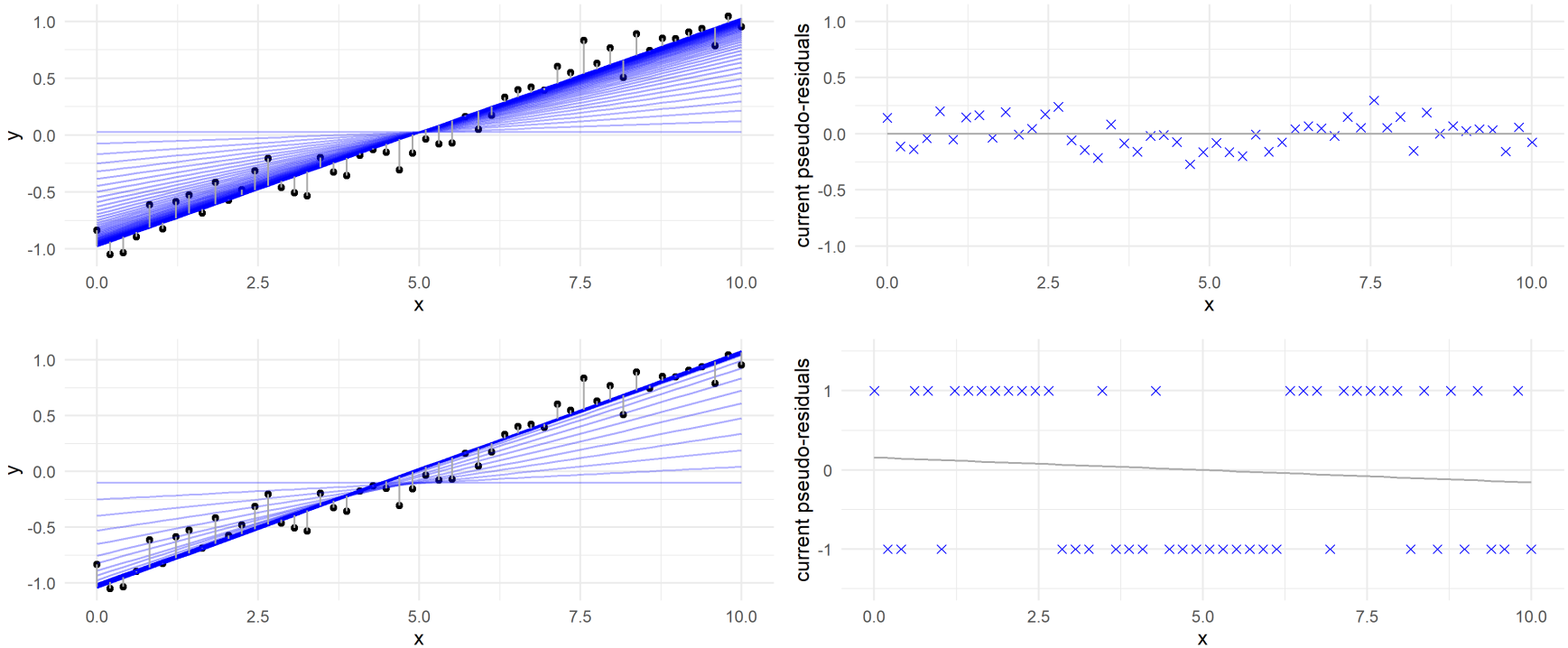


Iteration 10

For *L2*, as $\tilde{r}(f) = r(f)$, we find the optimal model in the very first iteration; only the multiplicative learning rate slows down optimization. In the *L1* case the base learner LMs fit pseudo-residuals that differ from model residuals, leading to a less monotonic optimization path.

LM WITH *L2* VS *L1* LOSS

Instead of using a GAM as base learner we now use a simple linear model. Top: *L2*, bottom: *L1*.

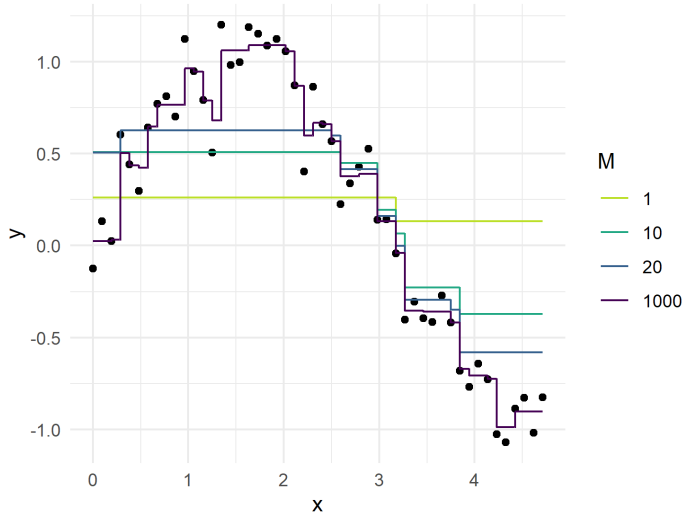


Iteration 100

For *L2*, as $\tilde{r}(f) = r(f)$, we find the optimal model in the very first iteration; only the multiplicative learning rate slows down optimization. In the *L1* case the base learner LMs fit pseudo-residuals that differ from model residuals, leading to a less monotonic optimization path.

Einführung in das Statistische Lernen

Gradient Boosting: Regularization



Learning goals

- Learn about three main regularization options: number of iterations, tree depth and shrinkage
- Understand how regularization influences model fit

REGULARIZATION AND SHRINKAGE

If GB runs for a large 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 learning rate $\beta^{[m]}$ of each iteration.

Note: If $\beta^{[m]}$ is found via line search, we multiply the next base learner by an additional constant **shrinkage parameter** $\nu \in (0, 1]$ to shorten the step length. In the case of a (commonly used) constant learning rate β, ν can be neglected by choosing a smaller value for β. Hence, on the following slides, we call β the shrinkage parameter or learning rate.

REGULARIZATION AND SHRINKAGE

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

In practice, a common recommendation to find a good first model without tuning both parameters is thus to choose β quite small and determine M by cross-validation.

It is probably best to tune all three parameters (M and β as well as the tree depths) jointly based on the training data via cross-validation or a related method.

STOCHASTIC GRADIENT BOOSTING

Stochastic gradient boosting 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.

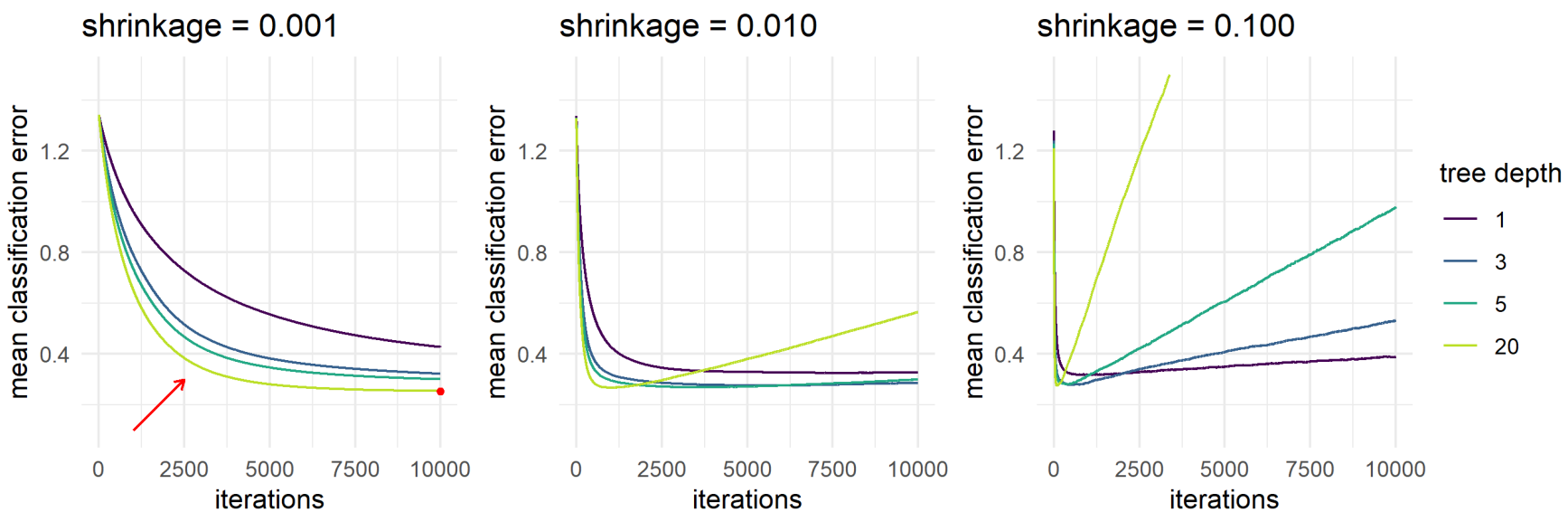
Especially for small training sets, this simple modification often leads to substantial empirical improvements. How large the improvements are depends on data structure, size of the dataset, base learner and size of the subsamples (so this is another tuning parameter).

EXAMPLE: SPAM DETECTION

We fit a gradient boosting model for different parameter values:

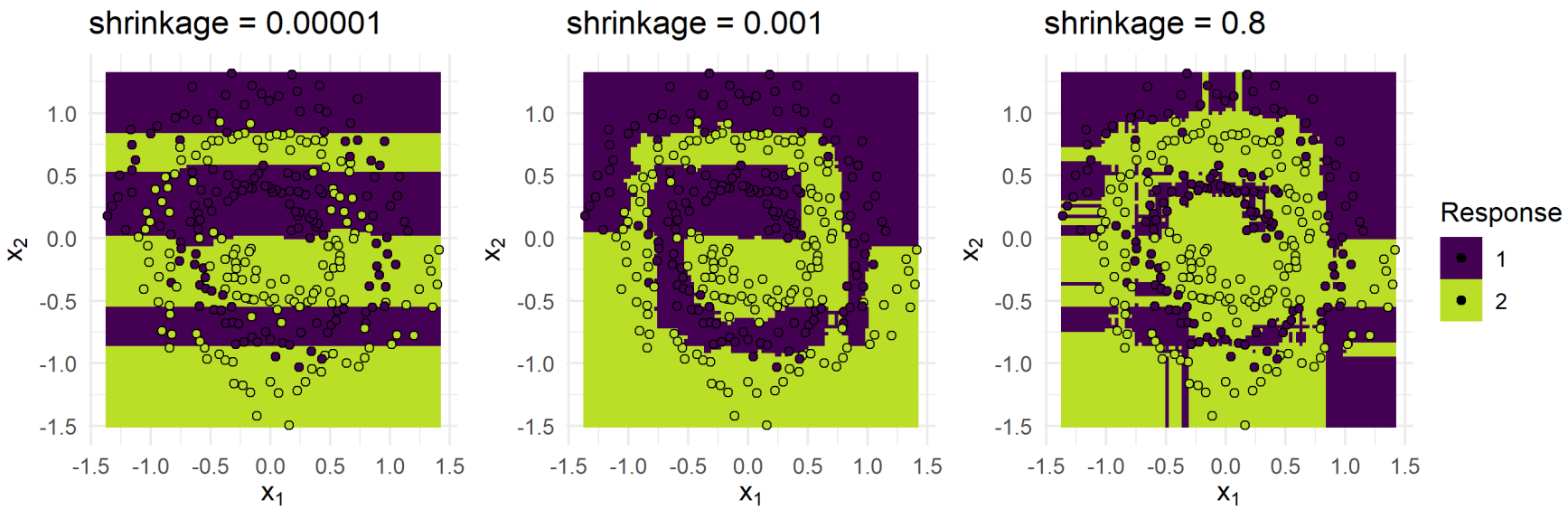
Parameter name	Values
Loss	Bernoulli (for classification)
Number of trees M	$\{0, 1, \ldots, 10000\}$
Shrinkage β	$\{0.001, 0.01, 0.1\}$
Max. tree depth	$\{1, 3, 5, 20\}$

We consider the 3-CV test error to find the optimal parameters (red):



EXAMPLE: SPIRALS DATA

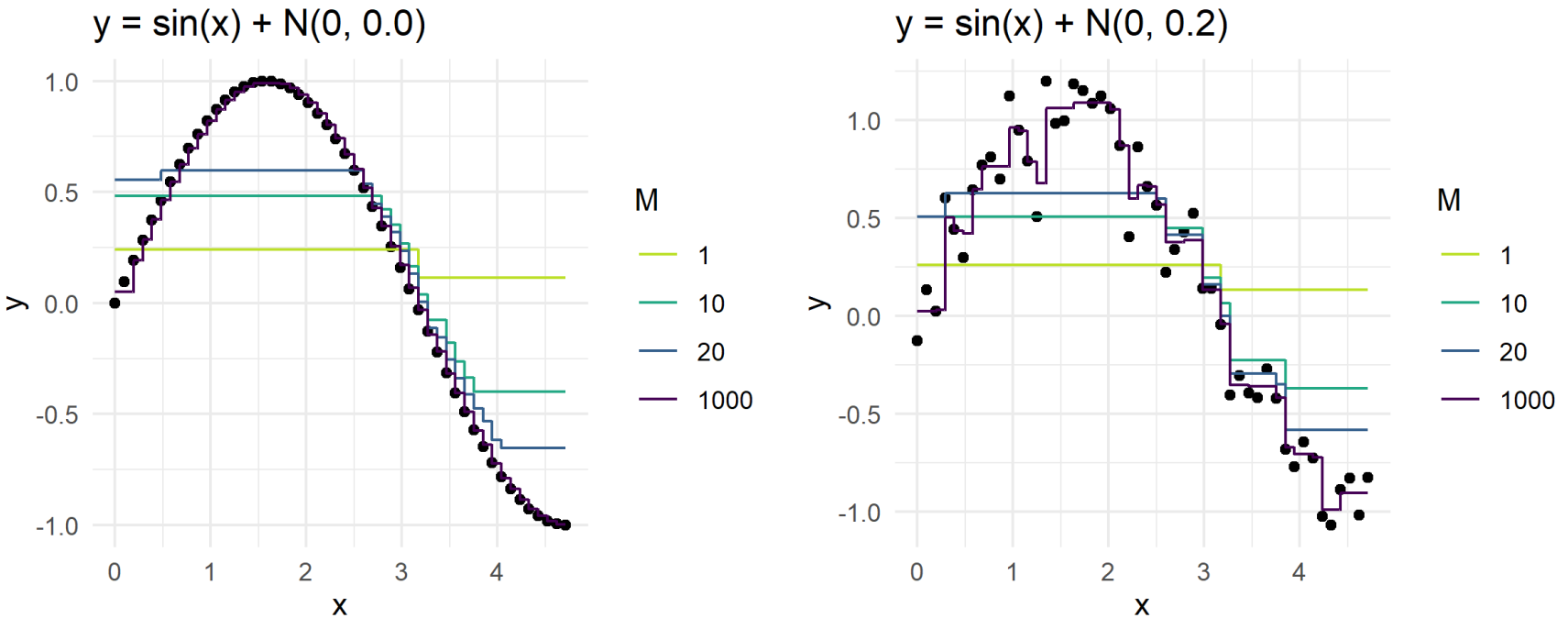
Consider the `spirals` data set with $sd = 0.1$ and $n = 300$. We examine the effect of the shrinkage parameter, holding the number of trees fixed at 10k and choosing a tree depth of 10:



We observe an oversmoothing effect in the left scenario with strong regularization (i.e., very small learning rate) and overfitting when regularization is too weak (right). $\beta = 0.001$ yields a pretty good fit.

EXAMPLE: SINUSOIDAL DATA

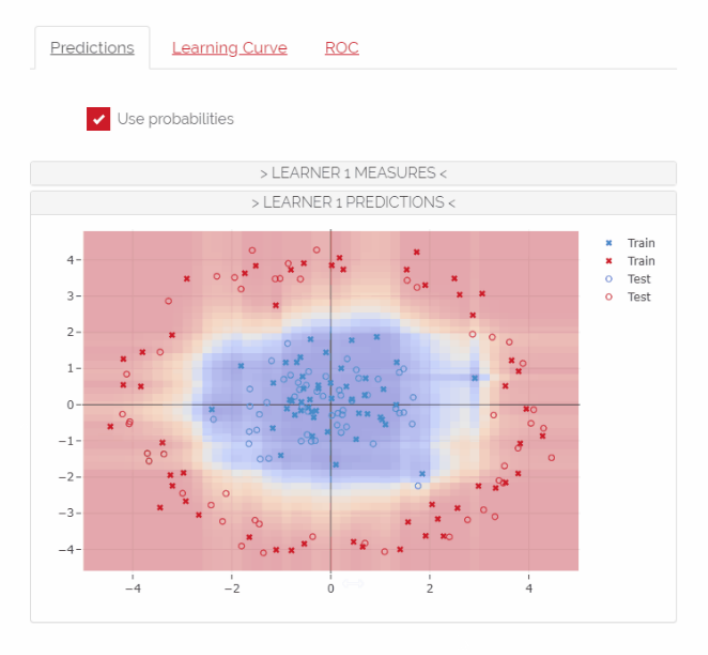
Here we choose a tree-stump base learner to model univariate data with sinusoidal behavior.



- Iterating this very simple base learner achieves a rather nice approximation of a smooth model in the end.
- Again, the model overfits the noisy case with less regularization.

Einführung in das Statistische Lernen

Gradient Boosting for Classification



Learning goals

- Transfering gradient boosting for regression to binary classification problems
- Introducing gradient boosting for multiclass problems

BINARY CLASSIFICATION

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

$$L(y, f(\mathbf{x})) = -y \cdot f(\mathbf{x}) + \log(1 + \exp(f(\mathbf{x}))).$$

Then,

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

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

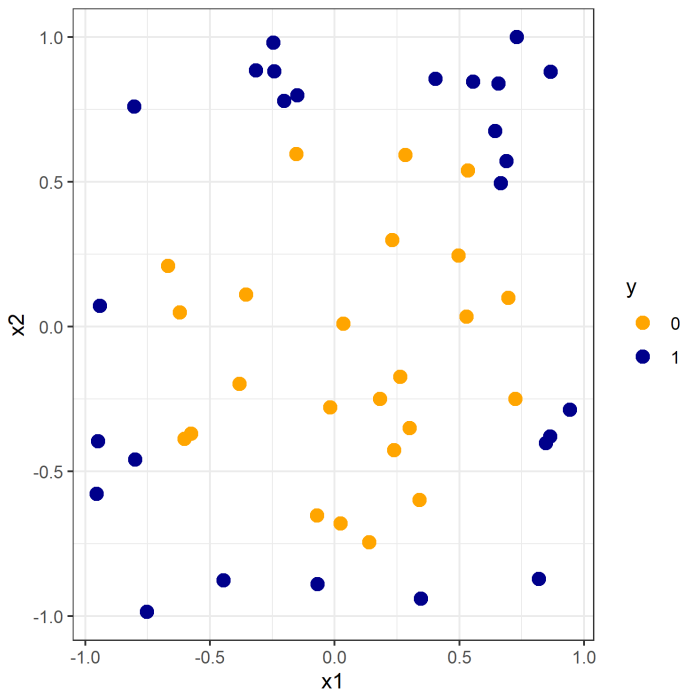
BINARY CLASSIFICATION

- Using this loss function, we can simply run GB as for regression. NB: Also here, we fit regression base learners against our numerical vector of pseudo-residuals with *L2* loss.
- We could also have used the exponential loss for classification with GB. It can be shown that the resulting GB algorithm is basically equivalent to AdaBoost. In practice there is no big difference, although Bernoulli loss makes a bit more sense from a theoretical (maximum likelihood) perspective.
- It follows that GB is a generalization of AdaBoost which can also use other loss functions and be used for different ML scenarios.

EXAMPLE

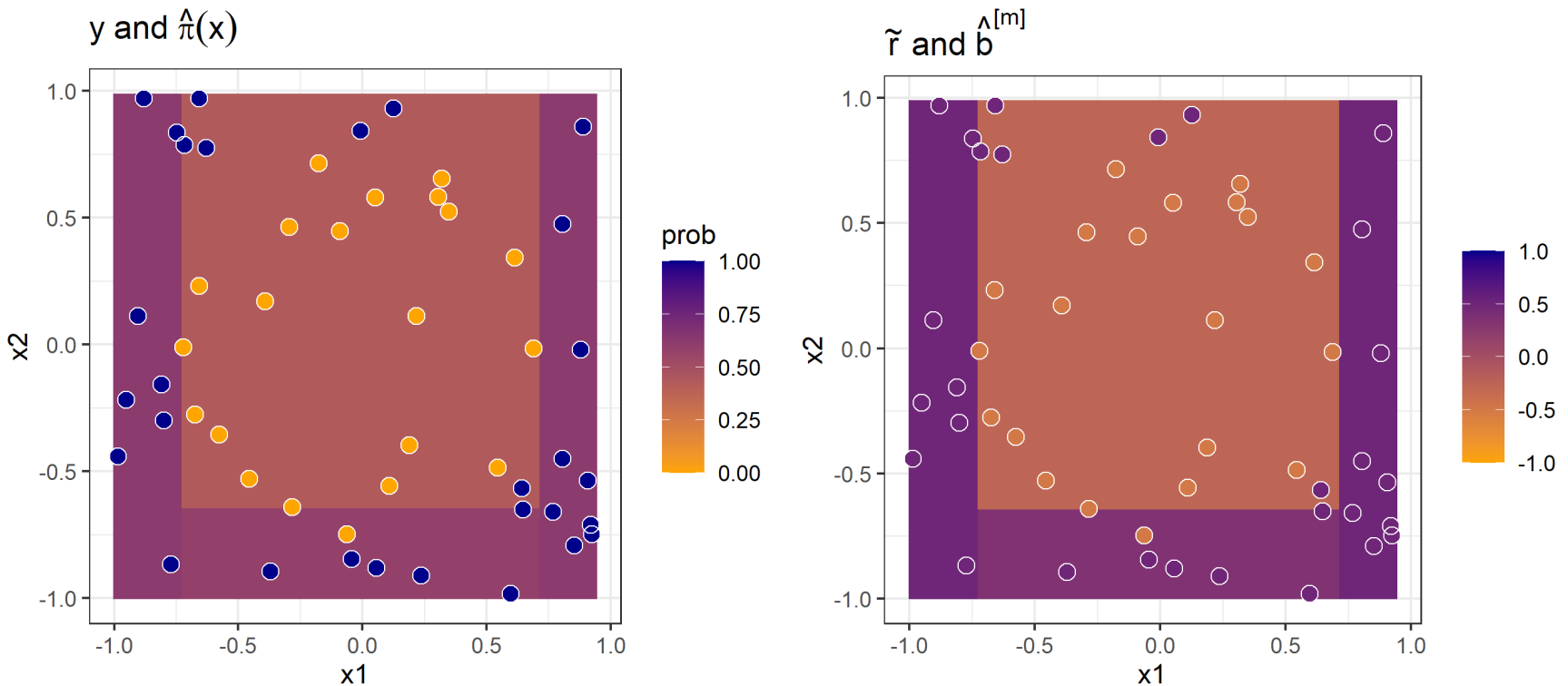
We now illustrate the boosting iterations for a classification example in a similar manner as we did for regression. However, we will now look at a simulation example with 2 instead of 1 influencial features and one binary target variable.

- We used the `mlbench` dataset `circle` with $n = 50$ observations.
- We used the Bernoulli loss to calculate pseudo-residuals and fitted in each iteration a base learner (regression tree with max. depth of 3) on them.
- We initialized with $f^{[0]} = log(1) = 0$.



EXAMPLE

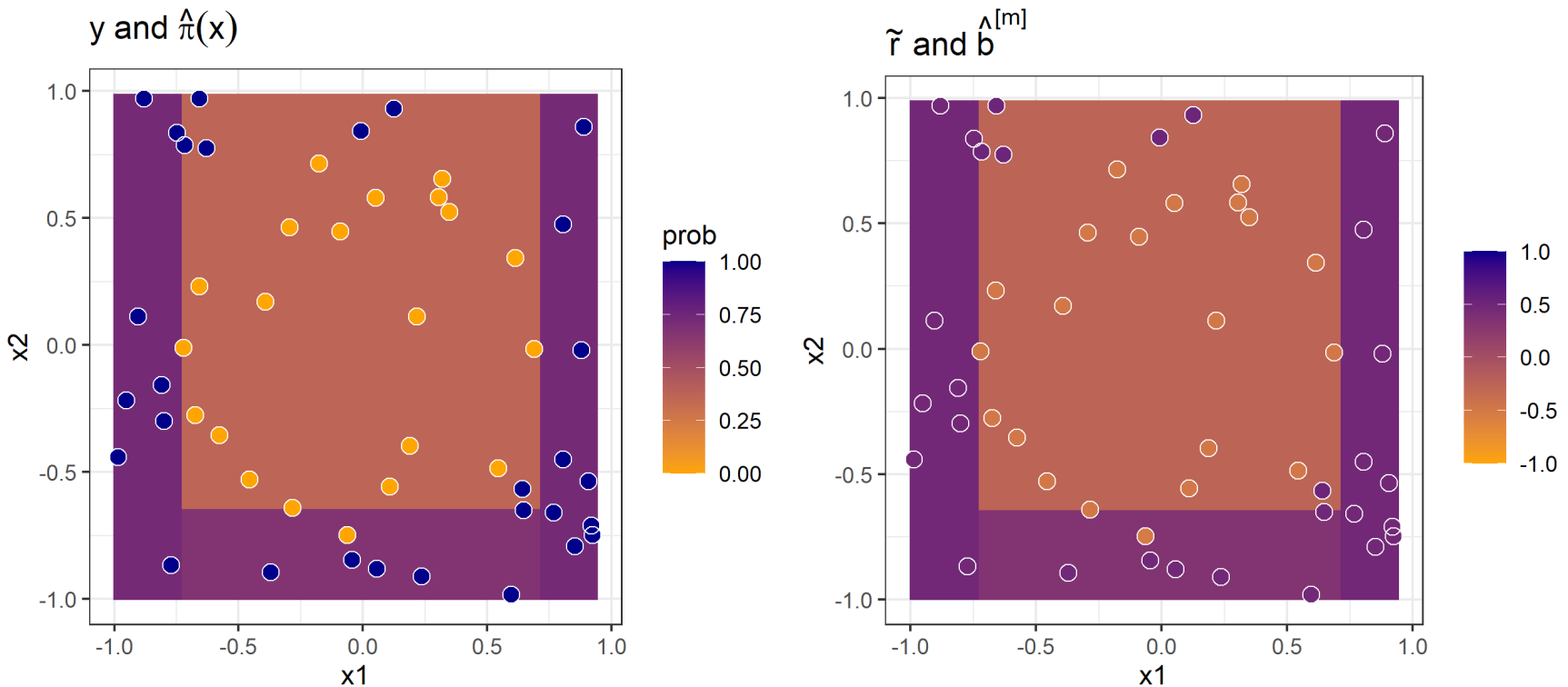
Left: Background color refers to prediction probabilites $\hat{\pi}^{[m]}$ and points to y (probabilities); Right: Background color refers to predictions of base learner $\hat{b}^{[m]}$ and points to $\tilde{r}$.



Iteration 1

EXAMPLE

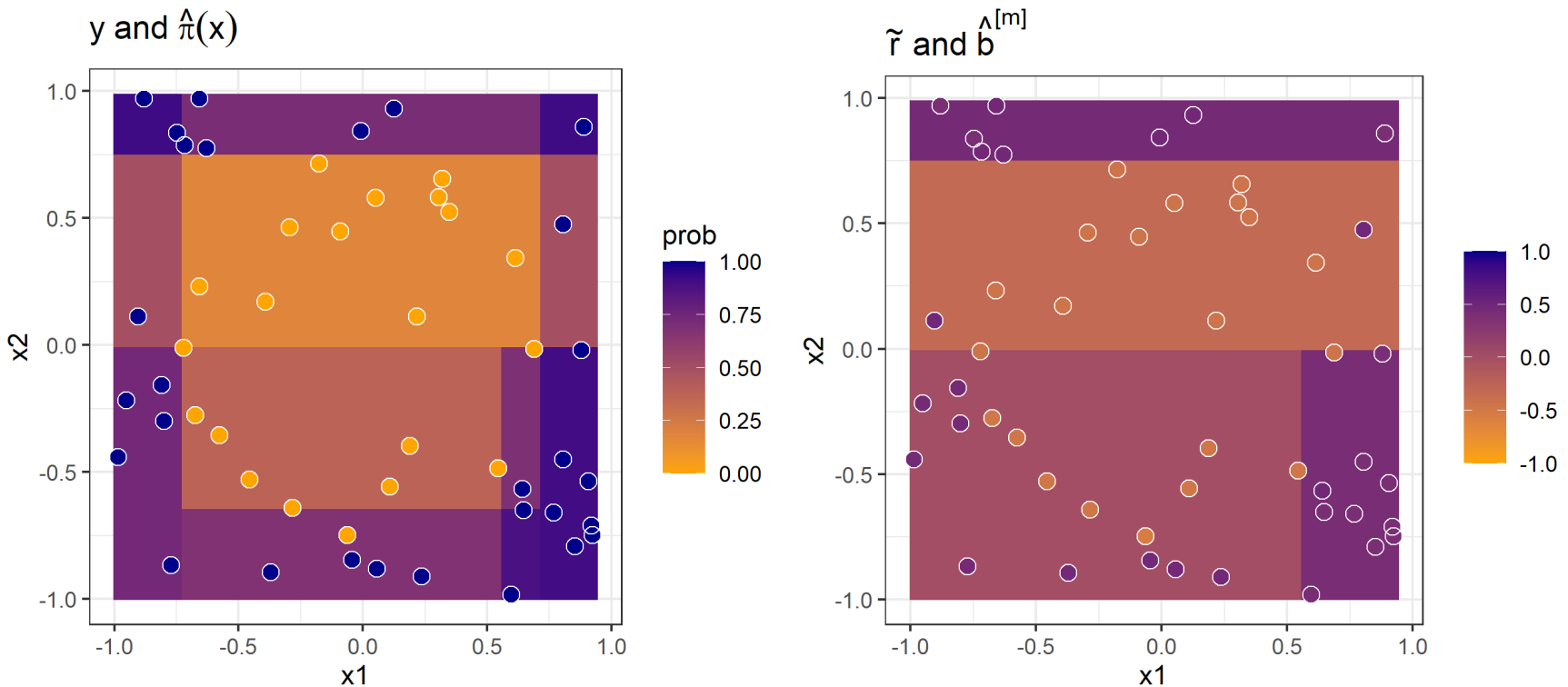
Left: Background color refers to prediction probabilites $\hat{\pi}^{[m]}$ and points to y (probabilities); Right: Background color refers to predictions of base learner $\hat{b}^{[m]}$ and points to $\tilde{r}$.



Iteration 2

EXAMPLE

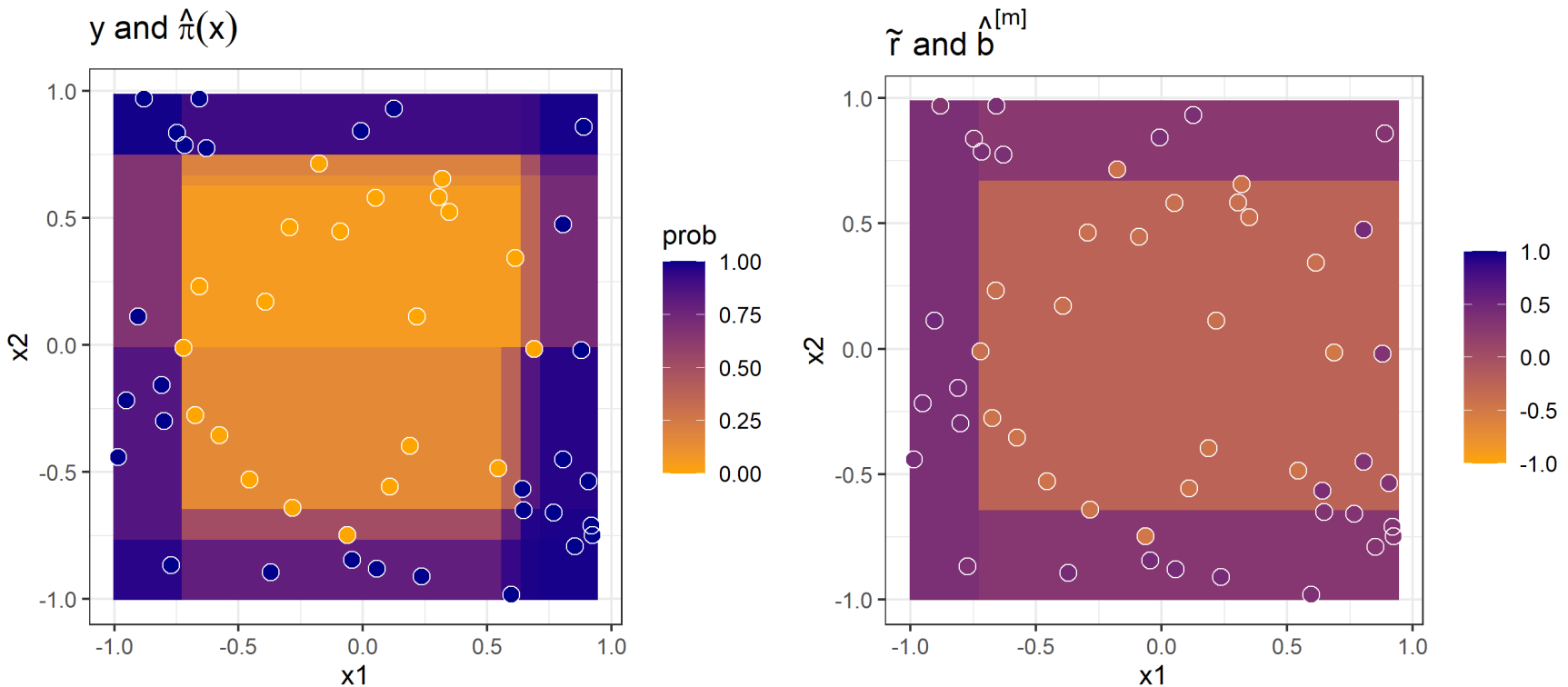
Left: Background color refers to prediction probabilites $\hat{\pi}^{[m]}$ and points to y (probabilities); Right: Background color refers to predictions of base learner $\hat{b}^{[m]}$ and points to $\tilde{r}$.



Iteration 5

EXAMPLE

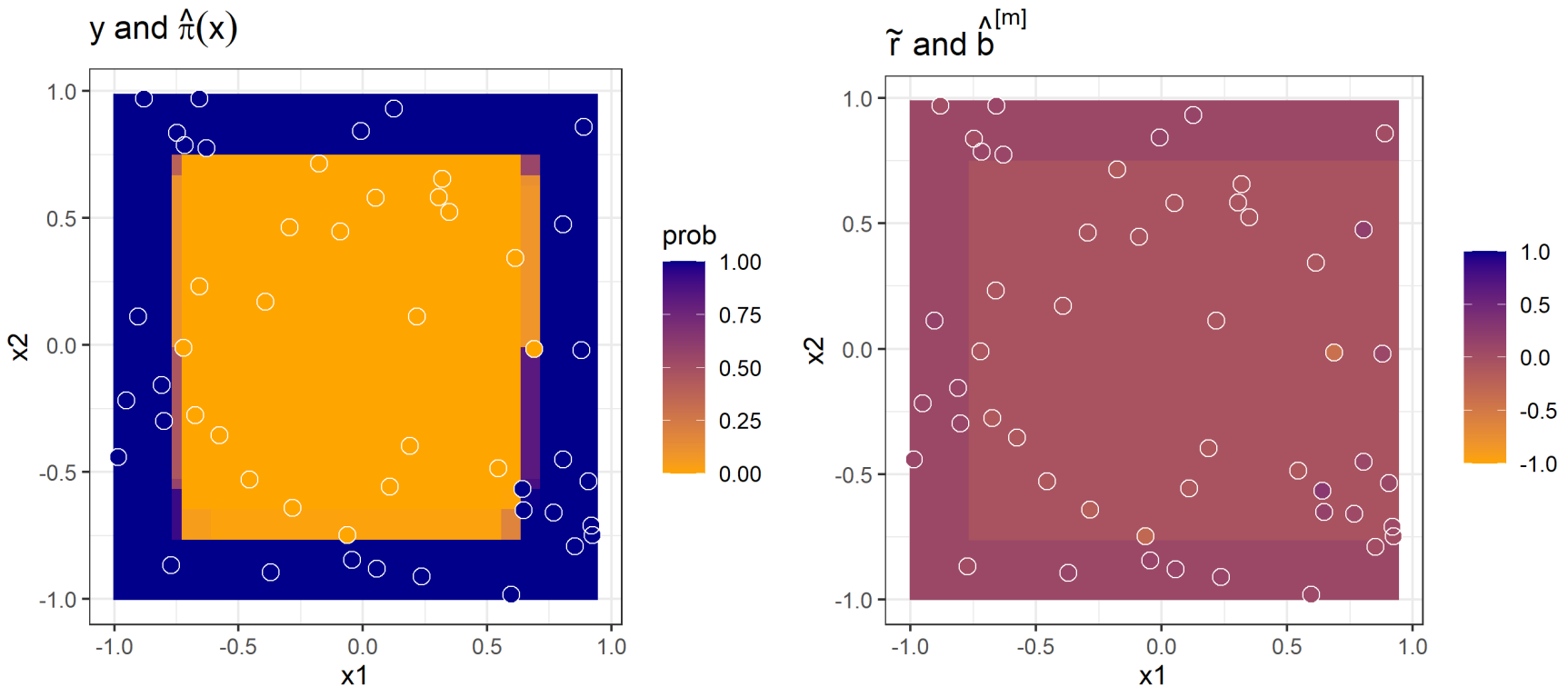
Left: Background color refers to prediction probabilites $\hat{\pi}^{[m]}$ and points to y (probabilities); Right: Background color refers to predictions of base learner $\hat{b}^{[m]}$ and points to $\tilde{r}$.



Iteration 10

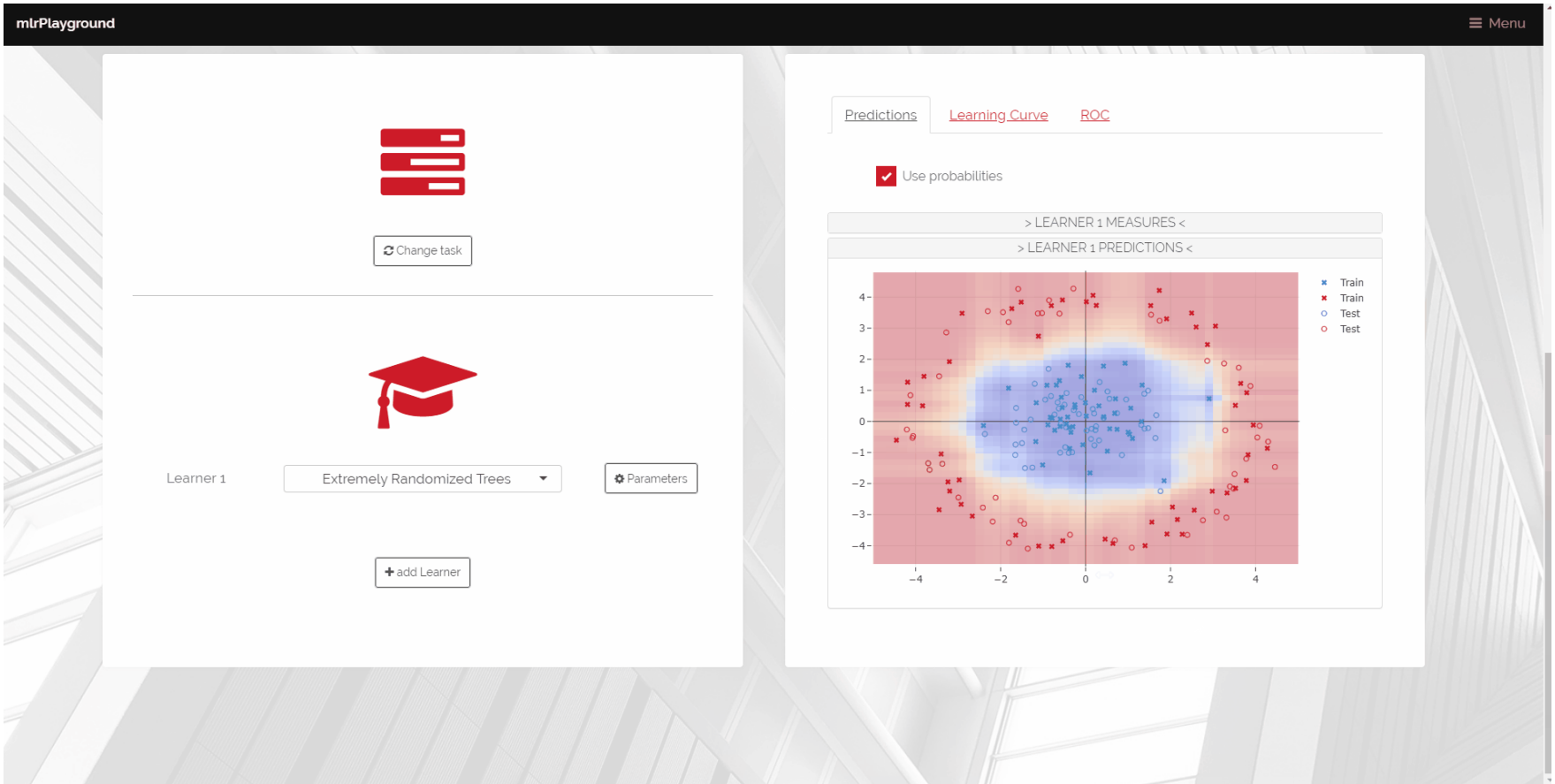
EXAMPLE

Left: Background color refers to prediction probabilites $\hat{\pi}^{[m]}$ and points to y (probabilities); Right: Background color refers to predictions of base learner $\hat{b}^{[m]}$ and points to $\tilde{r}$.



Iteration 100

MLRPLAYGROUND



▸ Open in browser.

MULTICLASS PROBLEMS

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

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

Multinomial loss L:

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

Pseudo-residuals:

$$-\frac{\partial L(y, f_1(\mathbf{x}), \ldots, f_g(\mathbf{x}))}{\partial f_k(x)} = \mathbb{1}_{\{y=k\}} - \pi_k(\mathbf{x}).$$

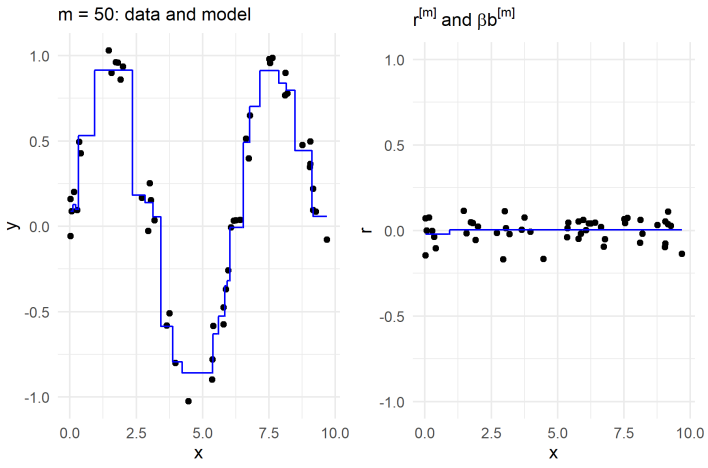
MULTICLASS PROBLEMS

Algorithm 1 GB for Multiclass

Initialize $f_k^{[0]}(\mathbf{x}) = 0,\ k = 1, \ldots, g$
for $m = 1 \to M$ **do**
    Set $\pi_k^{[m]}(\mathbf{x}) = \frac{\exp(f_k^{[m]}(\mathbf{x}))}{\sum_j \exp(f_j^{[m]}(\mathbf{x}))}, k = 1, \ldots, g$
    for $k = 1 \to g$ **do**
        For all i: Compute $\tilde{r}_k^{[m](i)} = \mathbb{1}_{\{y^{(i)}=k\}} - \pi_k^{[m]}(\mathbf{x}^{(i)})$
        Fit a regression base learner $\hat{b}_k^{[m]}$ to the pseudo-residuals $\tilde{r}_k^{[m](i)}$.
        Obtain $\hat{\beta}_k^{[m]}$ by constant learning rate or line-search.
        Update $\hat{f}_k^{[m]} = \hat{f}_k^{[m-1]} + \hat{\beta}_k^{[m]} \hat{b}_k^{[m]}$
    end for
end for
Output $\hat{f}_1^{[M]}, \ldots, \hat{f}_g^{[M]}$

Einführung in das Statistische Lernen

Gradient Boosting with Trees



Learning goals

- See how gradient boosting process is adapted for trees
- Understand relationship between model structure and interaction depth
- Understand multiclass extension for gradient boosting with trees

GRADIENT BOOSTING WITH 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

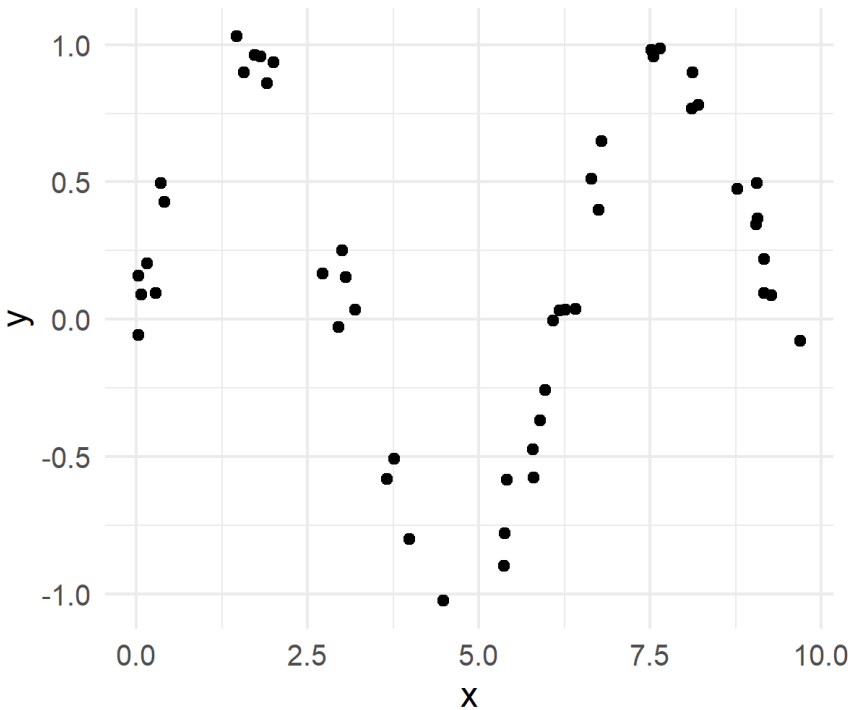
- No problems with categorical features.
- No problems with outliers in feature values.
- No problems with missing values.
- No problems with monotone transformations of features.
- Trees (and stumps!) can be fitted quickly, even for large n.
- Trees have a simple, built-in type of variable selection.

The gradient-boosted trees method retains all of them, and strongly improves the trees' predictive power. Furthermore, it is possible to adapt gradient boosting to tree learners in a targeted manner.

EXAMPLE 1

Simulation setting:

- Given: one feature *x* and one numeric target variable *y* of 50 observations.
- *x* is uniformly distributed between 0 and 10.
- *y* depends on *x* as follows: $y^{(i)} = \sin\left(x^{(i)}\right) + \epsilon^{(i)}$ with $\epsilon^{(i)} \sim \mathcal{N}(0, 0.01)$, $\forall i \in \{1, \ldots, 50\}$.



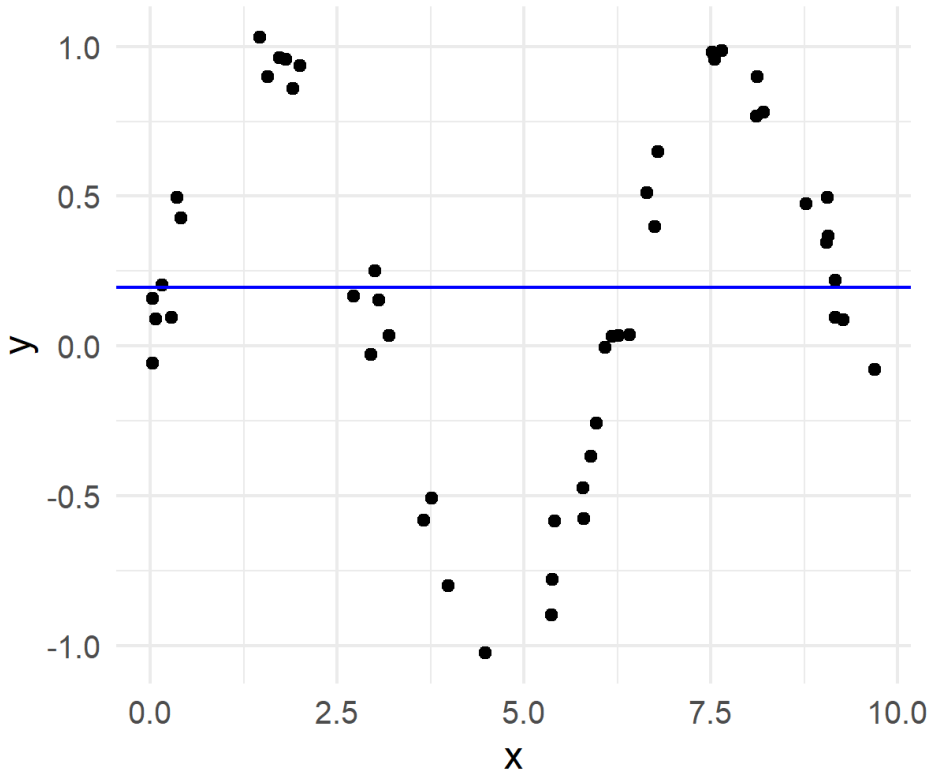
Aim: we want to fit a gradient boosting model to the data by using stumps as base learners.

Since we are facing a regression problem, we use *L2* loss.

EXAMPLE 1

Iteration 0: initialization by optimal constant (mean) prediction $\hat{f}^{[0](i)}(x) = \bar{y} \approx 0.2$.

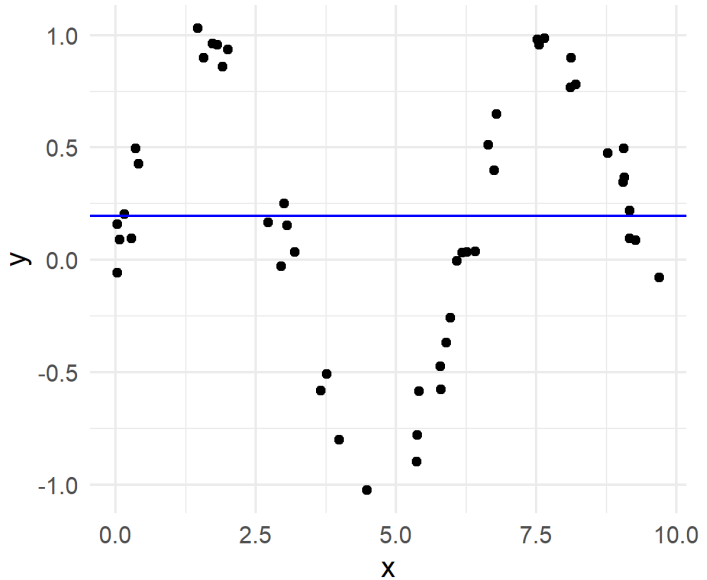
i	$x^{(i)}$	$y^{(i)}$	$\hat{f}^{[0]}$
1	0.03	0.16	0.20
2	0.03	-0.06	0.20
3	0.07	0.09	0.20
⋮	⋮	⋮	⋮
50	9.69	-0.08	0.20



EXAMPLE 1

Iteration 1: (1) Calculate pseudo-residuals $\tilde{r}^{[m](i)}$ and (2) fit a regression stump $b^{[m]}$.

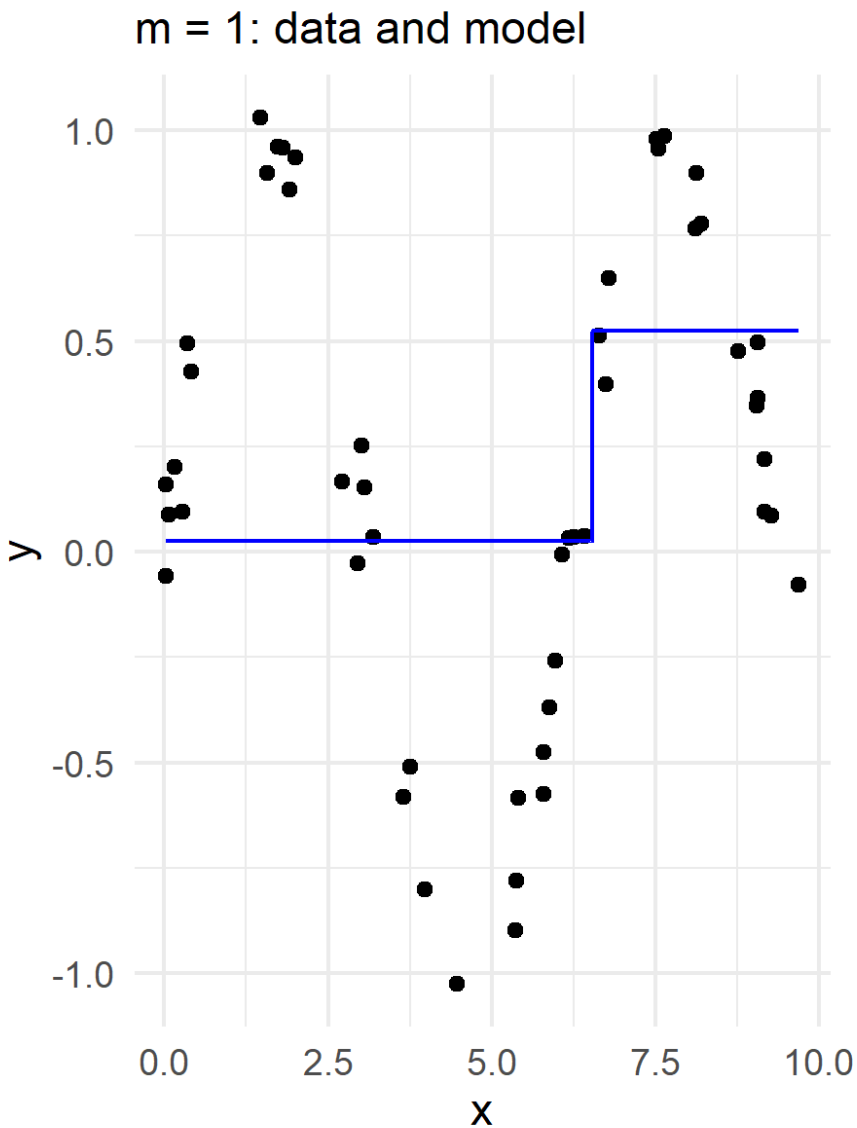
i	$x^{(i)}$	$y^{(i)}$	$\hat{f}^{[0]}$	$\tilde{r}^{[1](i)}$	$\hat{b}^{[1](i)}$
1	0.03	0.16	0.20	-0.04	-0.17
2	0.03	-0.06	0.20	-0.25	-0.17
3	0.07	0.09	0.20	-0.11	-0.17
⋮	⋮	⋮	⋮	⋮	⋮
50	9.69	-0.08	0.20	-0.27	0.33

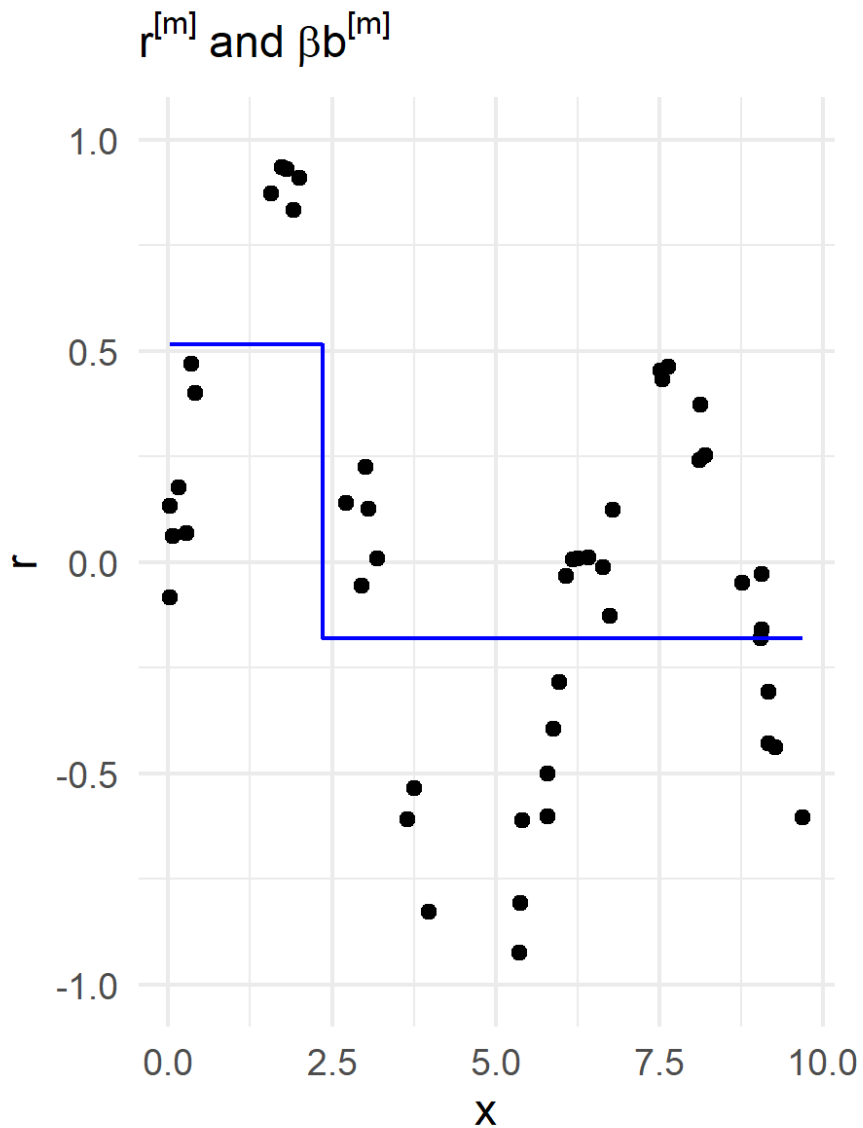


(3) Update model by $\hat{f}^{[1]}(x) = \hat{f}^{[0]}(x) + \hat{b}^{[1]}$.

EXAMPLE 1

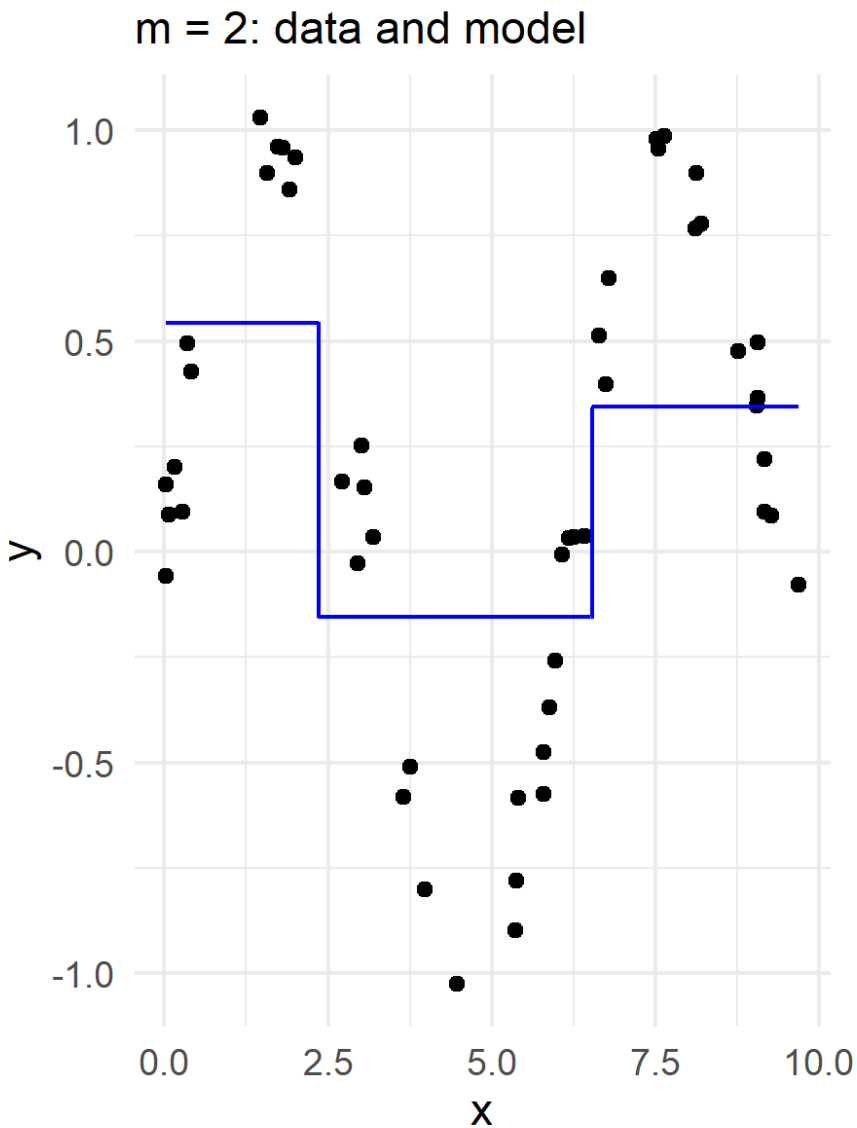
Repeat step (1) to (3):

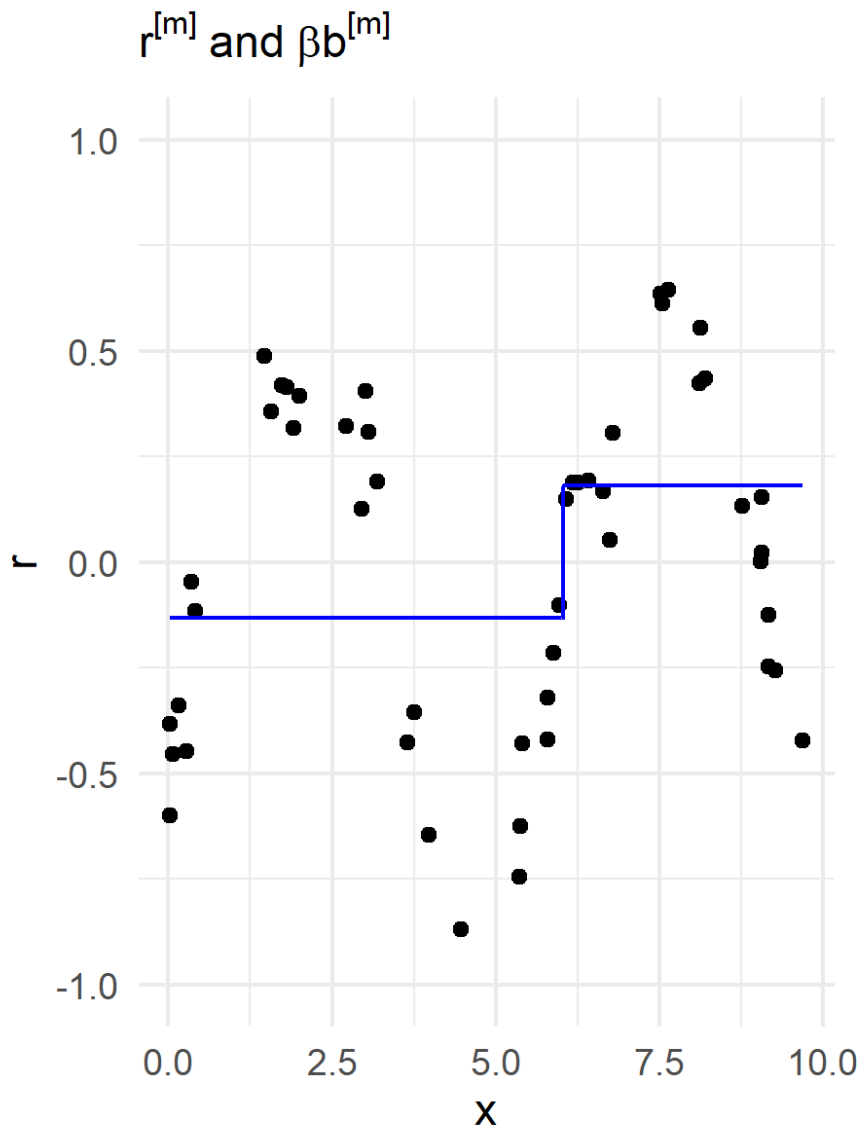




EXAMPLE 1

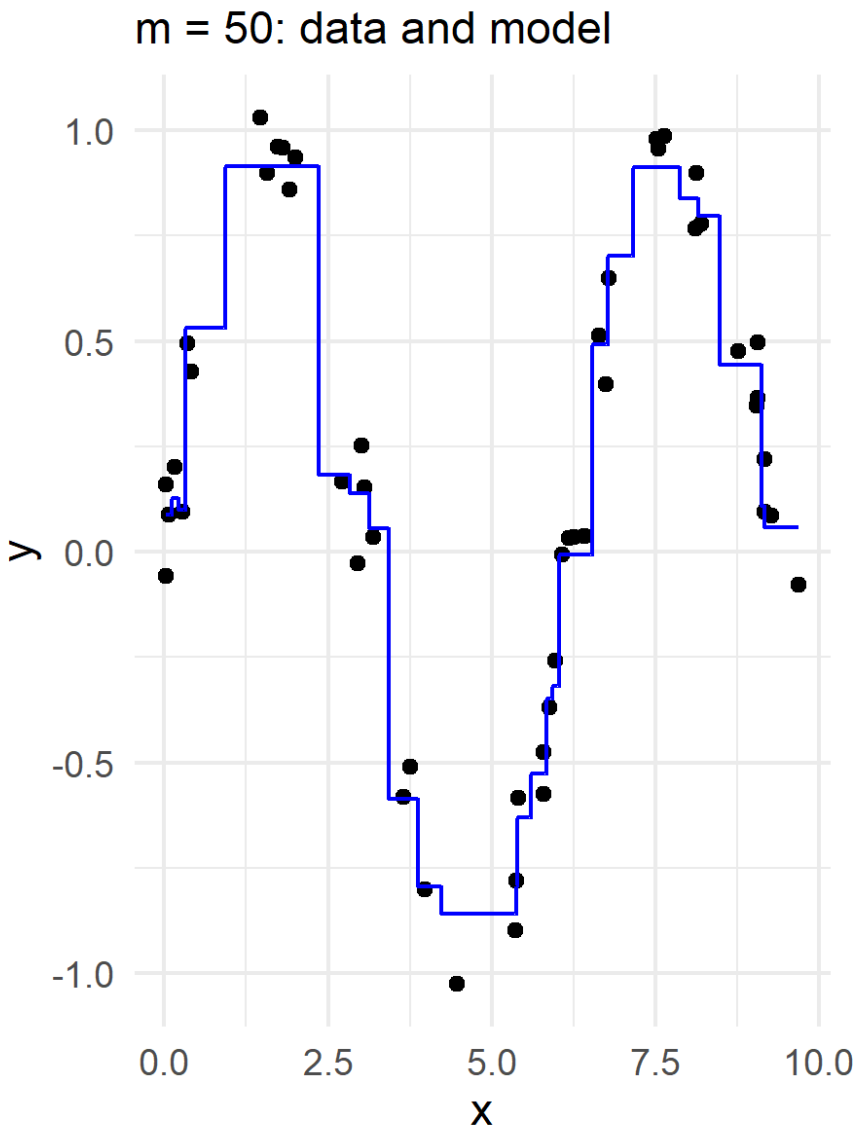
Repeat step (1) to (3):

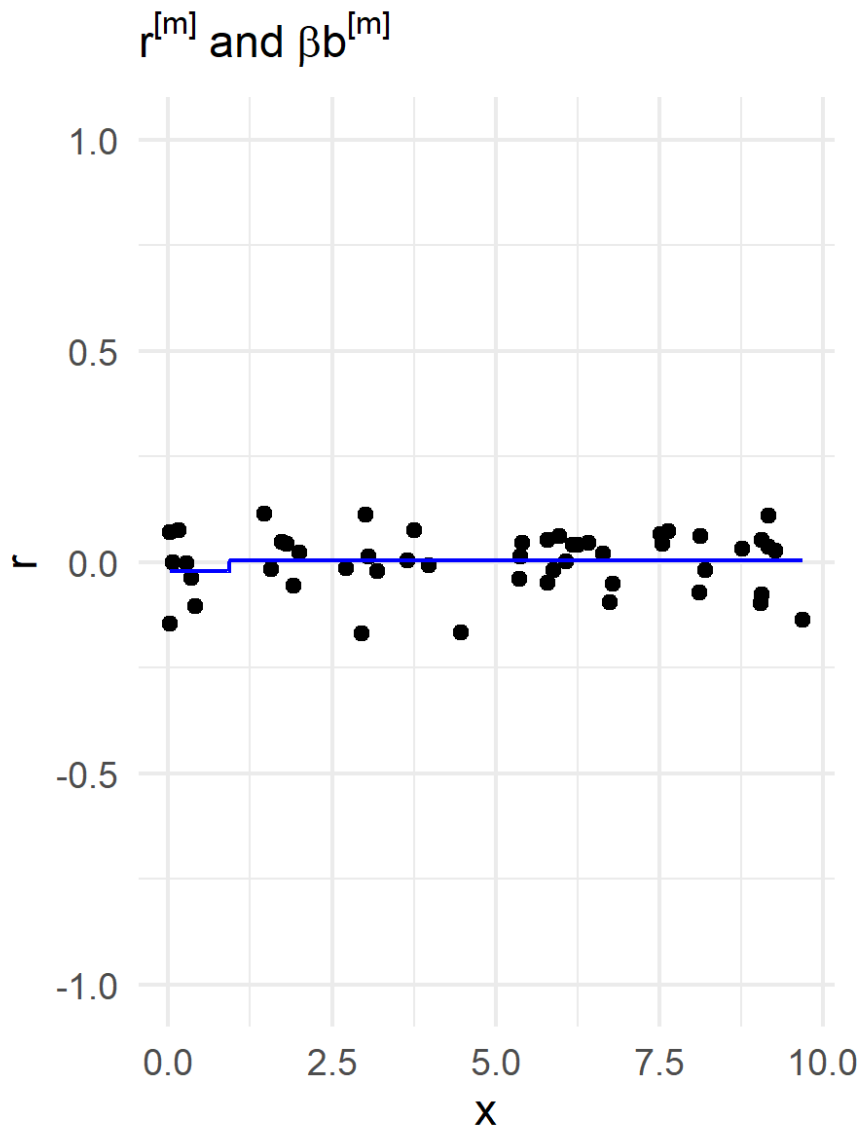




EXAMPLE 1

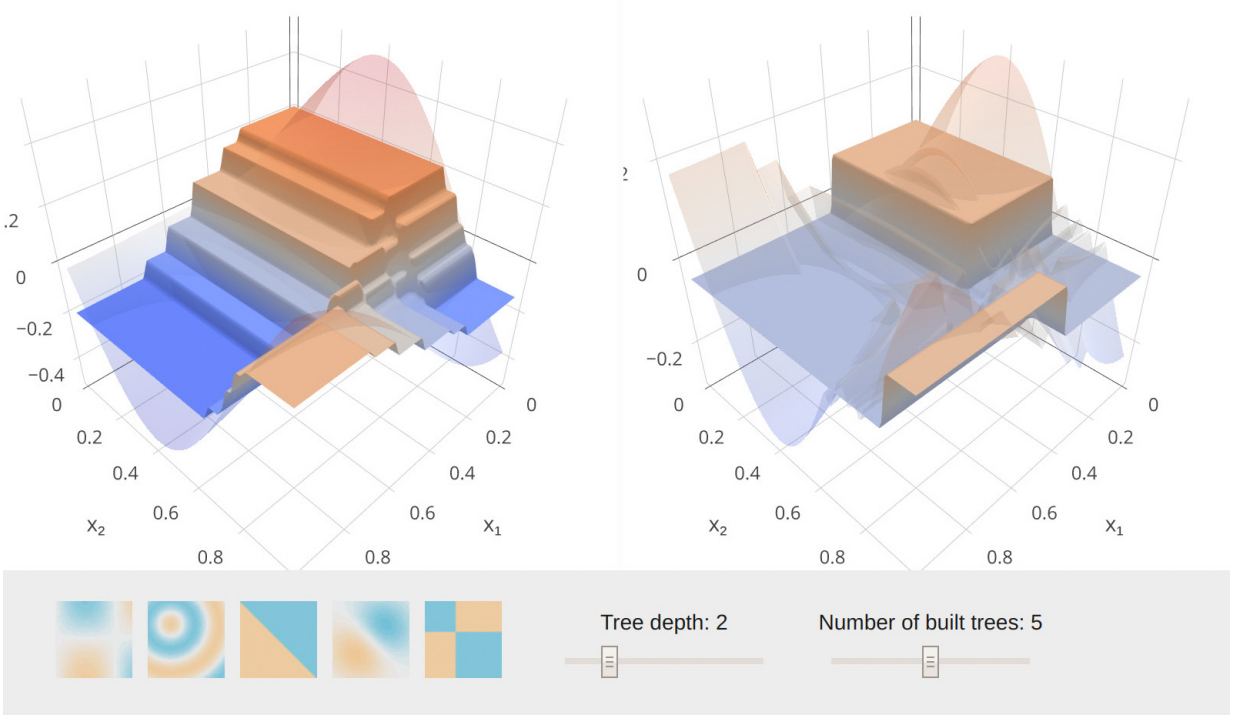
Repeat step (1) to (3):





EXAMPLE 2

This website shows on various 3D examples how tree depth and number of iterations influence the model fit of a GBM with trees.



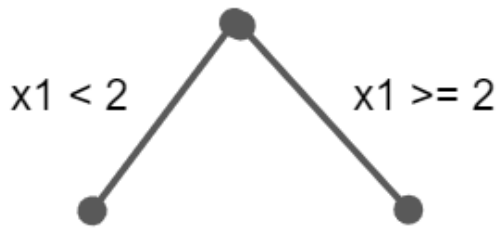
MODEL STRUCTURE AND INTERACTION DEPTH

The model structure of a gradient boosting model with trees is influenced by the chosen tree / interaction depth of $b^{[m]}(\mathbf{x})$.

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

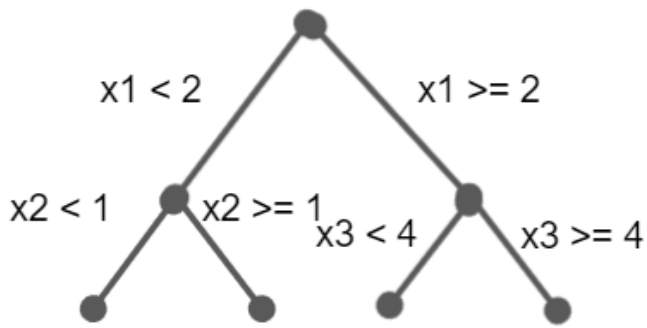
When using stumps (depth = 1), the resulting model is an additive model (GAM) without any interactions:

$$f(\mathbf{x}) = f_0 + \sum_{j=1}^{p} f_j(x_j)$$



When also including trees with a depth of 2, 2-way interactions are included and we get:

$$f(\mathbf{x}) = f_0 + \sum_{j=1}^{p} f_j(x_j) + \sum_{j \neq k} f_{j,k}(x_j, x_k)$$



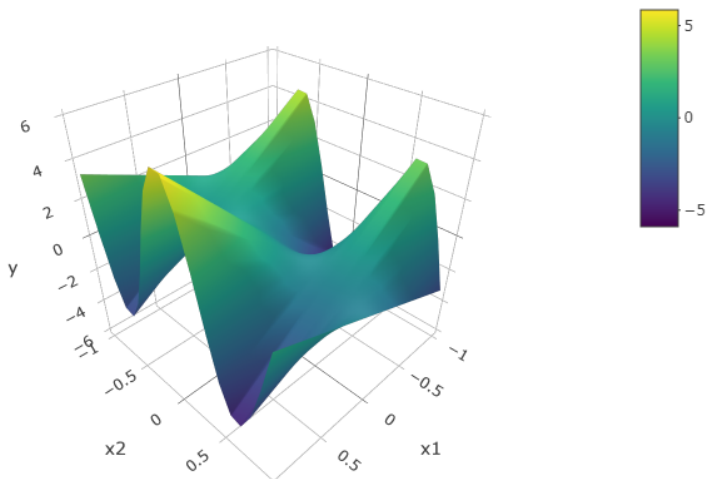
with f_0 being a constant intercept.

MODEL STRUCTURE AND INTERACTION DEPTH

Simulation setting:

- Given: two features x_1 and x_2 and one numeric target variable y of 500 observations.
- x_1 and x_2 are uniformly distributed between -1 and 1.
- Target function: $y^{(i)} = x_1^{(i)} - x_2^{(i)} + 5\cos(5x_2^{(i)}) \cdot x_1^{(i)} + \epsilon^{(i)}$ with $\epsilon^{(i)} \sim \mathcal{N}(0, 1)$, $\forall i \in \{1, \ldots, 500\}$.

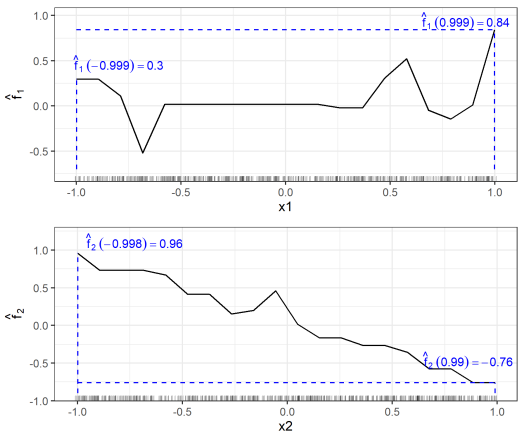
We fit two tree-based GBMs, one with an interaction depth (ID) of 1 (GAM) and one with an interaction depth of 2 (all possible interactions included).

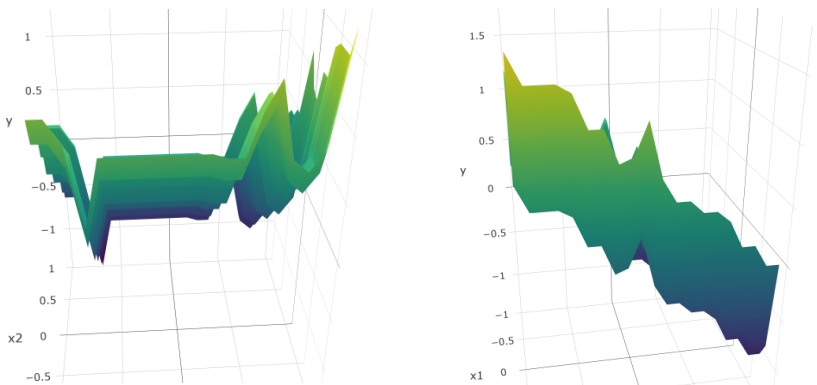


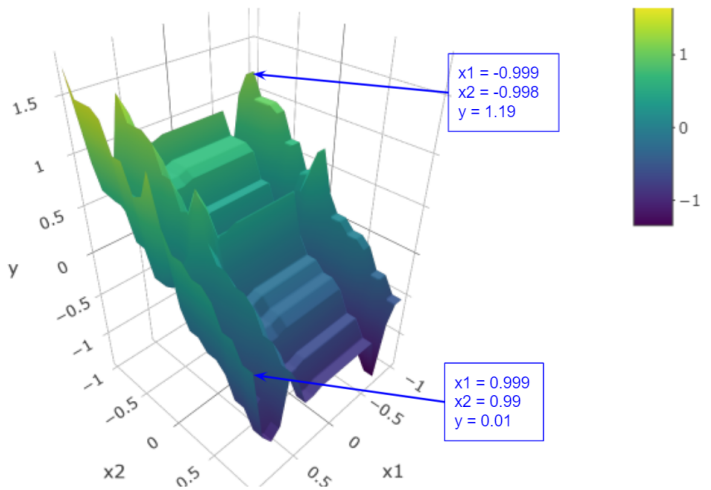
MODEL STRUCTURE AND INTERACTION DEPTH

GBM with interaction depth of 1 (GAM)

No interactions are modelled: Marginal effects of x_1 and x_2 add up to joint effect (plus the constant intercept $\hat{f}_0 = -0.07$).





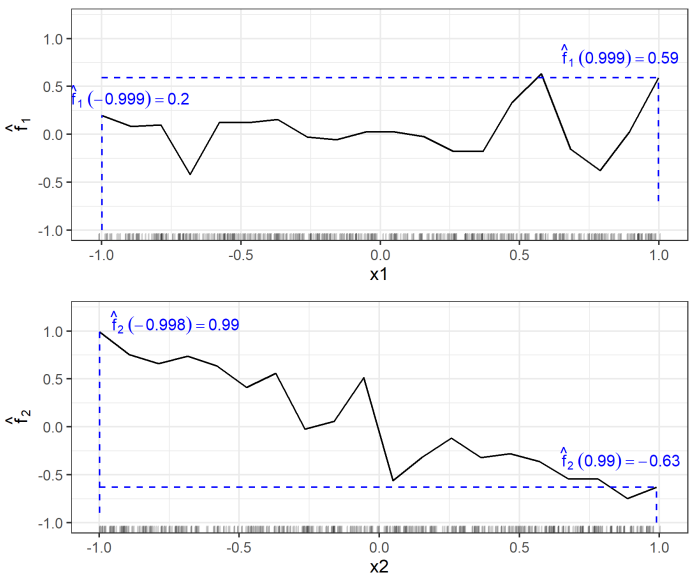


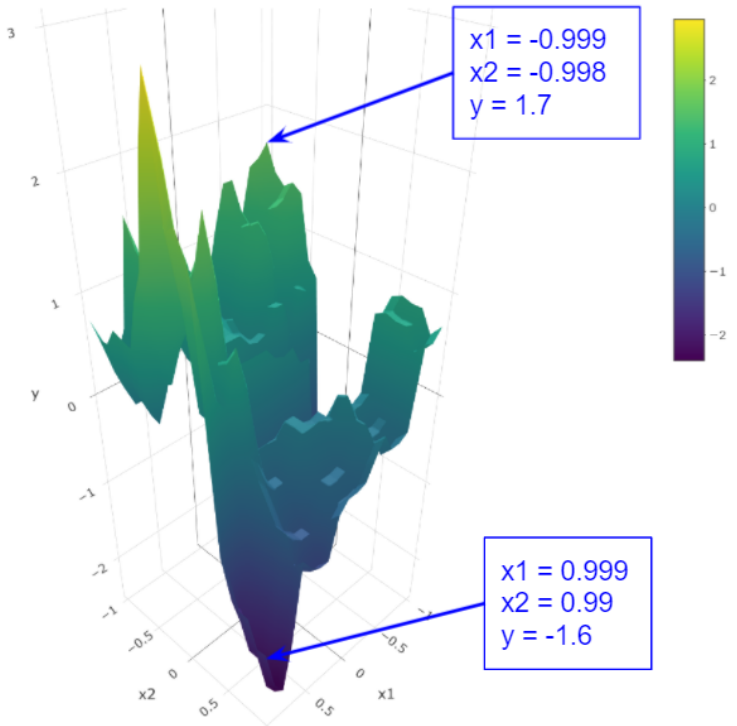
$$\hat{f}(-0.999, -0.998)$$
$$= \hat{f}_0 + \hat{f}_1(-0.999) + \hat{f}_2(-0.998)$$
$$= -0.07 + 0.3 + 0.96 = 1.19$$

MODEL STRUCTURE AND INTERACTION DEPTH

GBM with interaction depth of 2

Interactions between x_1 and x_2 are modelled: Marginal effects of x_1 and x_2 do NOT add up to joint effect due to interaction effects.





THEORETICAL BACKGROUND

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

For a fitted tree with regions R_t, the special additive structure can be exploited in boosting:

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

With $\tilde{c}_t^{[m]} = \beta^{[m]} \cdot c_t^{[m]}$ in the case that $\beta^{[m]}$ is a constant learning rate

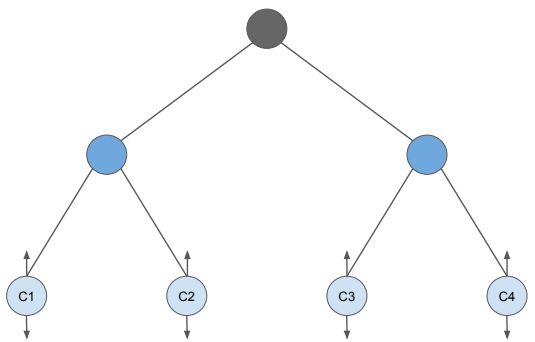
THEORETICAL BACKGROUND

We do the same steps as before: (1) calculate the pseudo-residuals, (2) fit a tree against pseudo-residuals, **but now** we keep only the structure of the tree and optimize the c parameter in a (further) post-hoc step.

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

We can determine/change all $\tilde{c}_t^{[m]}$ individually and directly L-optimally:

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



THEORETICAL BACKGROUND

An alternative approach ist to directly fit a loss-optimal tree. The risk function is then defined by:

$$\mathcal{R}(\mathcal{N}') = \sum_{i \in \mathcal{N}'} L(y^{(i)}, f^{[m-1]}(\mathbf{x}^{(i)}) + c)$$

with $\mathcal{N}'$ being the index set of a specific (left or right) node after splitting and c being a constant value added to the current model for this node. Thus, instead of having a two-step approach of first fitting a tree to the pseudo-residuals of the current model and then finding the optimal value for c, we now directly build a tree that finds c loss-optimally. Since c is unknown, it needs to be determined, which can either be done by a line search or by taking the derivative:

$$\frac{\partial \mathcal{R}(\mathcal{N}')}{\partial c} = \sum_{i \in \mathcal{N}'} \frac{\partial L(y^{(i)}, f^{[m-1]}(\mathbf{x}^{(i)}) + c)}{\partial f|_{f=f^{[m-1]}+c}} = 0$$

THEORETICAL BACKGROUND

Algorithm 1 Tree Algorithm for Gradient Boosting.

Input: All observations $\mathcal{N}$ and risk function $\mathcal{R}$
Output: $\mathcal{N}_l^{j^*,s^*}$ and $\mathcal{N}_r^{j^*,s^*}$
for $j = x_1 \ldots x_p$ **do**
    for every split s on feature j **do**
        $\mathcal{N}_l^{j,s} = \{i \in \mathcal{N}\}_{j^{(i)} \leq s}$
        $\mathcal{N}_r^{j,s} = \{i \in \mathcal{N}\}_{j^{(i)} > s}$
        Find c which minimizes $\mathcal{R}$ for each node
        $\mathcal{I}(j,s) = \mathcal{R}(\mathcal{N}_l^{j,s}) + \mathcal{R}(\mathcal{N}_r^{j,s})$
    end for
end for
$(j^*, s^*) \in \arg\min_{j,s} \mathcal{I}(j,s)$

The tree algorithm based on the CART algorithm of Breiman shows one partitioning step based on the risk function we introduced before.