LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Introduction to Machine Learning

Regularization

Department of Statistics – LMU Munich

Motivation

EXAMPLE: OVERFITTING

- Assume we want to predict the daily maximum **ozone level** in LA given a data set containing 50 observations.
- The data set contains 12 features describing time conditions (e.g., weekday, month), the weather (e. g. temperature at different weather stations, humidity, wind speed) or geographic variables (e. g. the pressure gradient).
- We fit a linear regression model using **all** of the features

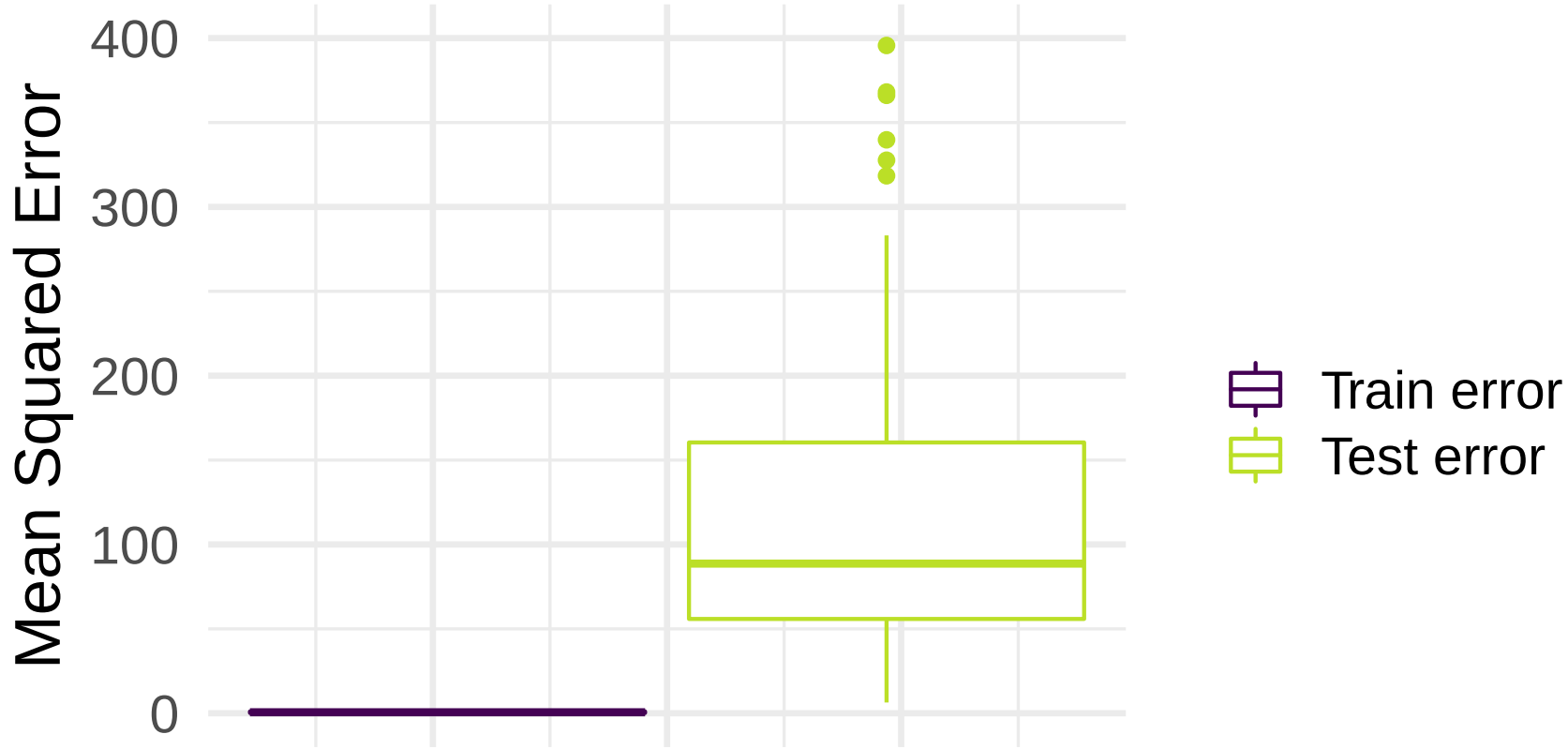
$$f(\mathbf{x} \mid \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_{12} x_{12}$$

  with the *L*2-loss.
- We evaluate the performance with 10 times 10-fold CV.

We use (a subset of) the `Ozone` data set from the `mlbench` package. This way, we artificially create a "high-dimensional" dataset by reducing the number of observations drastically while keeping the number of features fixed.

EXAMPLE: OVERFITTING

While our model fits the training data almost perfectly (left), it generalizes poorly to new test data (right). We overfitted.



AVOID OVERFITTING

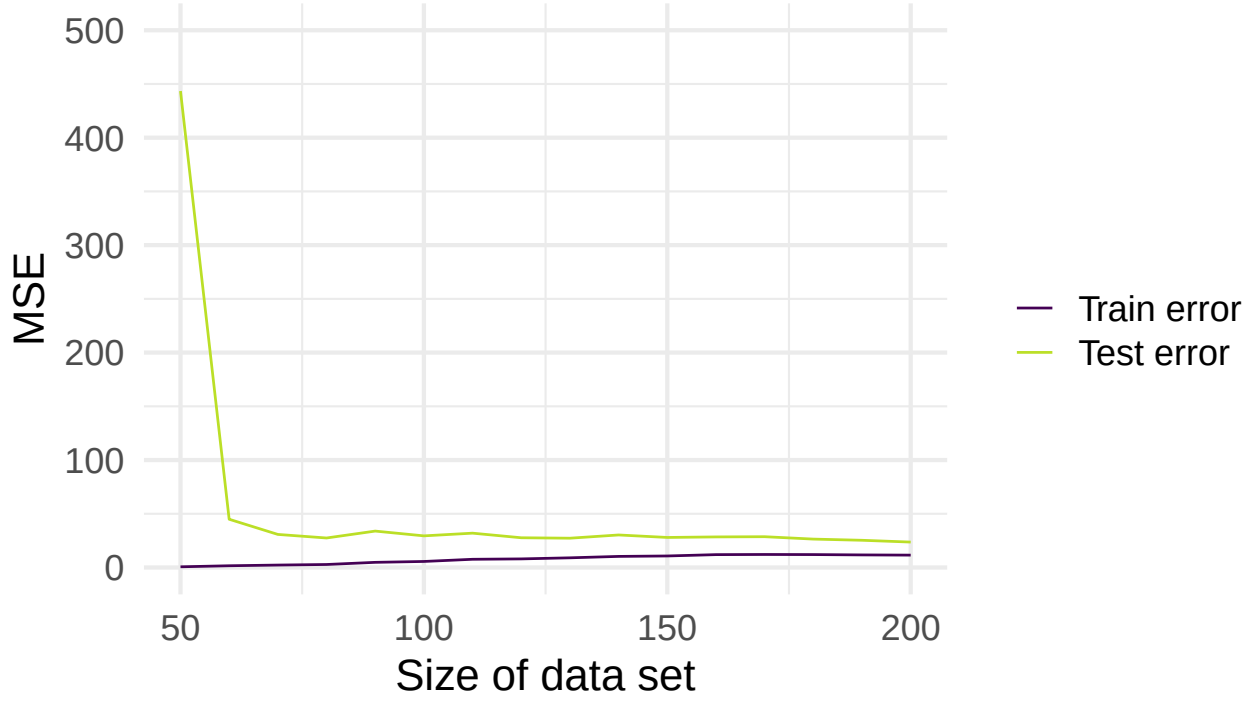
Why can **overfitting** happen? And how to avoid it?

1. Not enough data
   $\rightarrow$ collect **more data**
2. Data is noisy
   $\rightarrow$ collect **better data** (reduce noise)
3. Models are too complex
   $\rightarrow$ use **less complex models**
4. Aggressive loss optimization
   $\rightarrow$ **optimize less**

AVOID OVERFITTING

Approach 1: Collect more data

We explore the model's performance for increased data set size by 10 times 10-fold CV. The fit worsens slightly, but the test error decreases.



Good idea, but often not feasible in practice.

Note: We initially only used 50 out of 200 train observations to simulate the collection of more data.

AVOID OVERFITTING

Approach 2: Reduce **complexity**

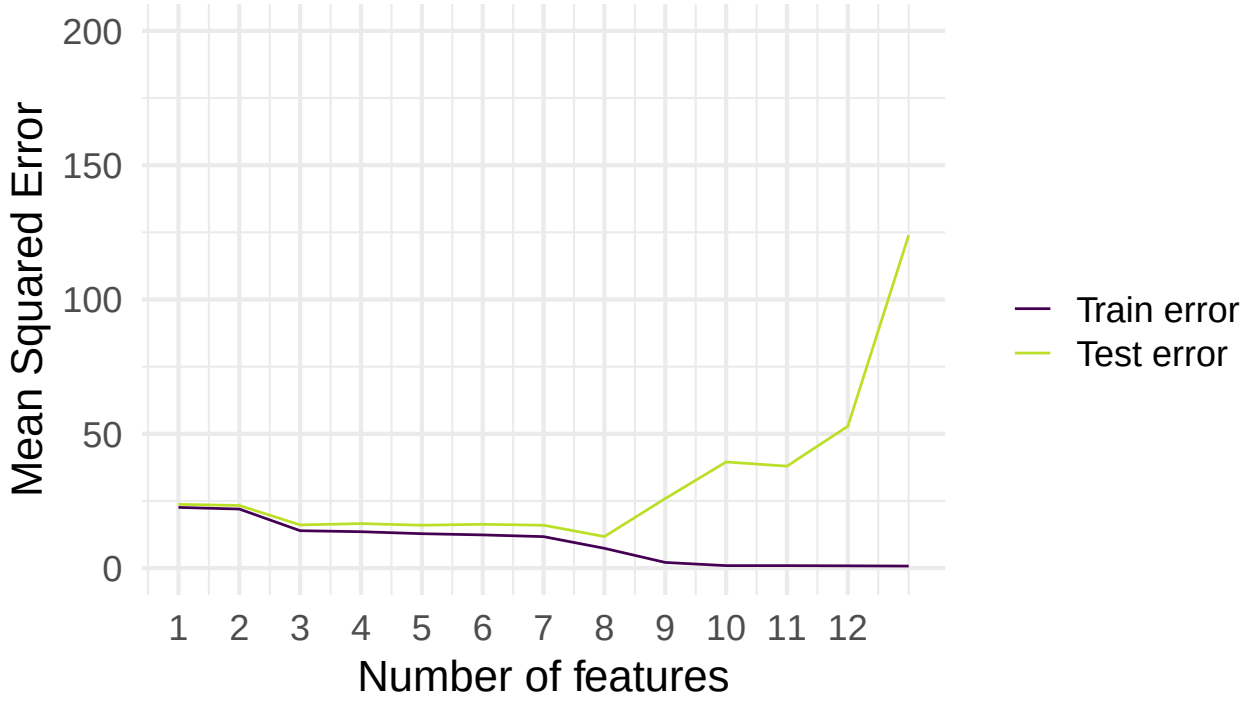
We try the simplest model we can think of: the constant model. For the *L2*-loss, the optimal constant model is mean over all $y^{(i)}$:

$$f(\mathbf{x} \mid \boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} y^{(i)} =: \bar{y}$$

We then increase the complexity of the model step-by-step by adding one feature at a time.

AVOID OVERFITTING

We can control the complexity of the model by including/excluding features. We can try out all feature combinations and investigate the model fit.



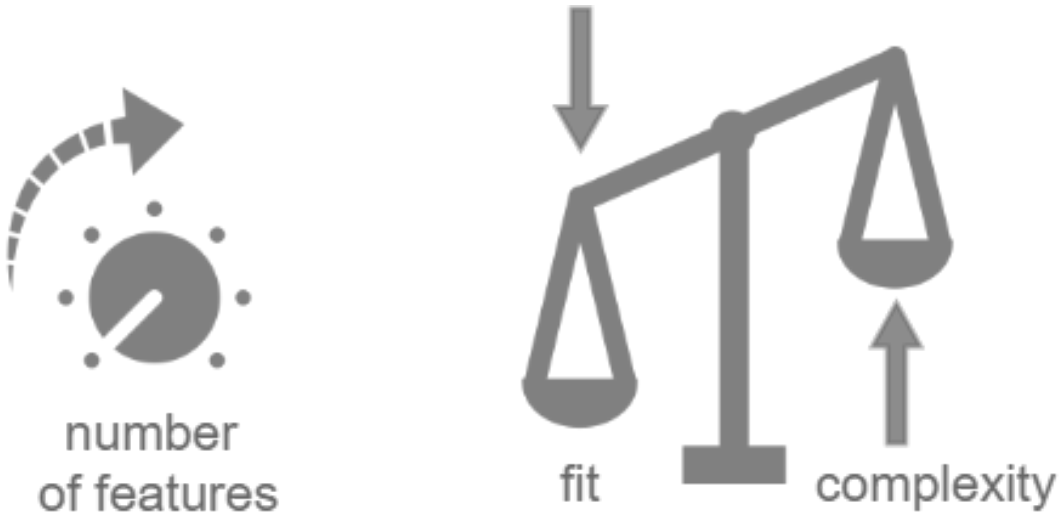
Note: For simplicity, we added the features in one specific order - but there are $2^{12} = 4096$ potential feature combinations.

AVOID OVERFITTING

We have contradictory goals

- **maximizing the fit** (minimize the empirical risk $\mathcal{R}_{\text{emp}}(f)$)
- **minimizing the complexity** of the model (e.g. min #features)

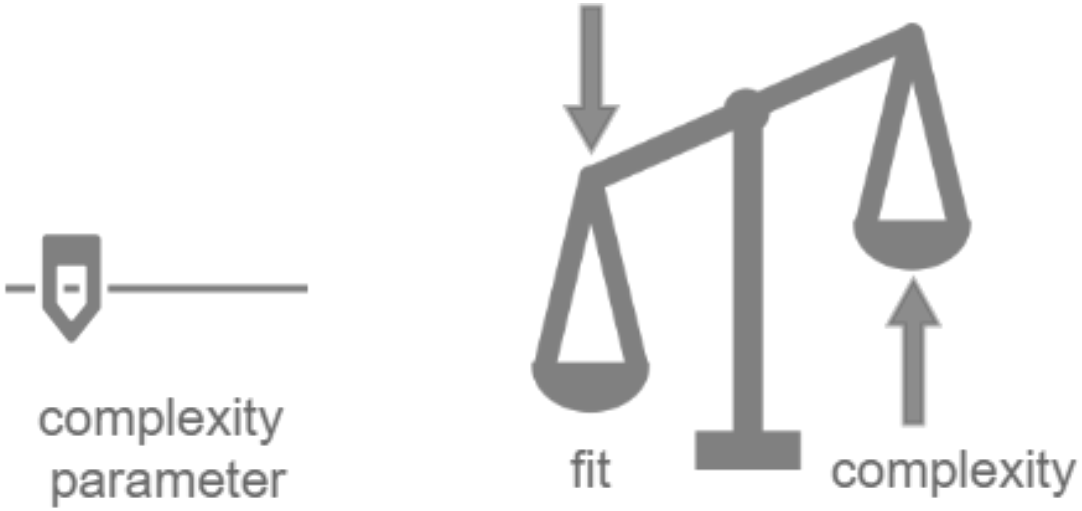
We need to find the “sweet spot”.



AVOID OVERFITTING

Until now, we can either add a feature completely or not at all.

Instead of controlling the complexity in a discrete way by specifying the number of features, we might prefer to control the complexity **on a continuum** from simple to complex.



Regularized Empirical Risk Minimization

REGULARIZED EMPIRICAL RISK MINIMIZATION

Recall, empirical risk minimization with a complex hypothesis set tends to overfit. A major tool to handle overfitting is **regularization**.

In the broadest sense, regularization refers to any modification made to a learning algorithm that is intended to reduce its generalization error but not its training error.

Explicitly or implicitly, such modifications represent the preferences we have regarding the elements of the hypothesis set.

REGULARIZED EMPIRICAL RISK MINIMIZATION

Commonly, regularization takes the following form:

$$\mathcal{R}_{\text{reg}}(f) = \mathcal{R}_{\text{emp}}(f) + \lambda \cdot J(f) = \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right) + \lambda \cdot J(f)$$

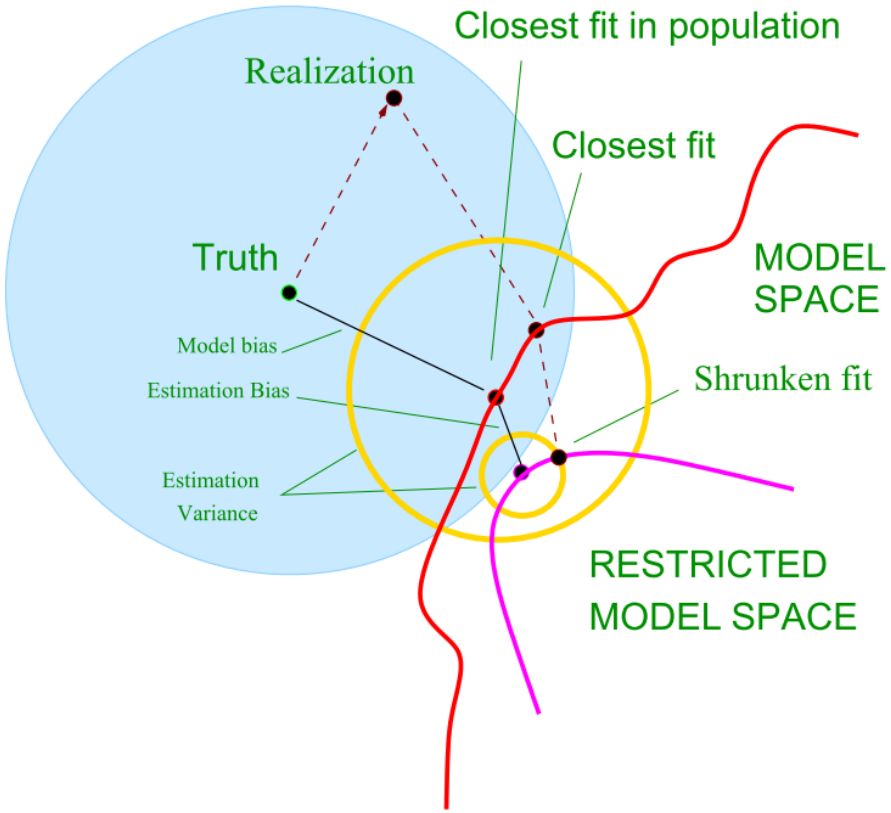
- $J(f)$ is called **complexity penalty**, **roughness penalty** or **regularizer**.
- It measures the "complexity" of a model and penalizes it in the fit.
- As for $\mathcal{R}_{\text{emp}}$, often $\mathcal{R}_{\text{reg}}$ and J are defined on $\boldsymbol{\theta}$ instead of f, so $\mathcal{R}_{\text{reg}}(\boldsymbol{\theta}) = \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) + \lambda \cdot J(\boldsymbol{\theta})$.

REGULARIZED EMPIRICAL RISK MINIMIZATION

Remark:

- Note that we now face an optimization problem with two criteria:
  1. models should fit well (low empirical risk),
  2. but not be too complex (low $J(f)$).
- We decide to combine the two in a weighted sum and to control the trade-off via the complexity control parameter λ.
- λ is hard to set manually and is usually selected via cross-validation (see later).
- $\lambda = 0$: the regularized risk $\mathcal{R}_{\text{reg}}(f)$ reduces to the simple empirical $\mathcal{R}_{\text{emp}}(f)$.
- If λ goes to infinity, we stop caring about the loss / fit and models become as "simple" as possible.

REGULARIZED EMPIRICAL RISK MINIMIZATION



Hastie, The Elements of Statistical Learning, 2009

Regularization in Linear Modeling

REGULARIZATION IN THE LINEAR MODEL

Recall, for the (unregularized) linear model we optimize

$$\mathcal{R}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x} \right)^2 .$$

Recall that the problem has a closed form solution:

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X} \mathbf{y}.$$

- Coefficient estimates for ordinary least squares with $f(\mathbf{x} \mid \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}$ usually require a full rank design matrix $\mathbf{X}$, otherwise $\mathbf{X}^\top \mathbf{X}$ is not invertible.
- When features are highly correlated (i.e. $\mathbf{X}$ is close to "being not a full rank matrix"), the least-squares estimate becomes highly sensitive to random errors in the observed response, producing a large variance in the fit.

RIDGE REGRESSION

Ridge regression is a **shrinkage method**, as it shrinks the regression coefficients $\boldsymbol{\theta}$ towards 0, by putting an L_2-penalty$^{(*)}$ on it:

$$\begin{aligned} \hat{\boldsymbol{\theta}}_{\text{Ridge}} &= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x} \right)^2 + \lambda ||\boldsymbol{\theta}||_2^2 \\ &= \arg\min_{\boldsymbol{\theta}} \left(\mathbf{y} - \mathbf{X}\boldsymbol{\theta} \right)^\top \left(\mathbf{y} - \mathbf{X}\boldsymbol{\theta} \right) + \lambda \boldsymbol{\theta}^\top \boldsymbol{\theta}. \end{aligned}$$

Optimization is possible (as in the normal LM) in analytical form:

$$\hat{\boldsymbol{\theta}}_{\text{Ridge}} = (\mathbf{X}^\top \mathbf{X} + \lambda \boldsymbol{I})^{-1} \mathbf{X}^\top \mathbf{y}$$

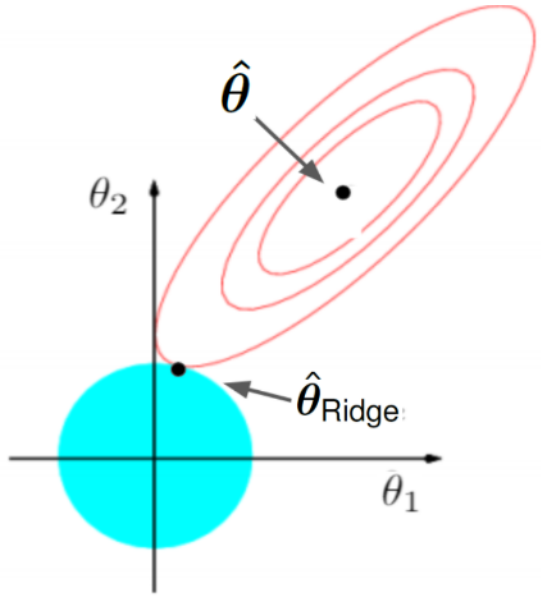
$^{(*)}$ $J(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2 = \theta_1^2 + \theta_2^2 + ... + \theta_p^2$

RIDGE REGRESSION

The problem can be formulated equivalently as a constrained optimization problem (for an appropriate choice of t):

$$\min_{\boldsymbol{\theta}} \quad \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x} \right)^2$$

$$\text{subject to:} \quad \|\boldsymbol{\theta}\|_2^2 \leq t$$

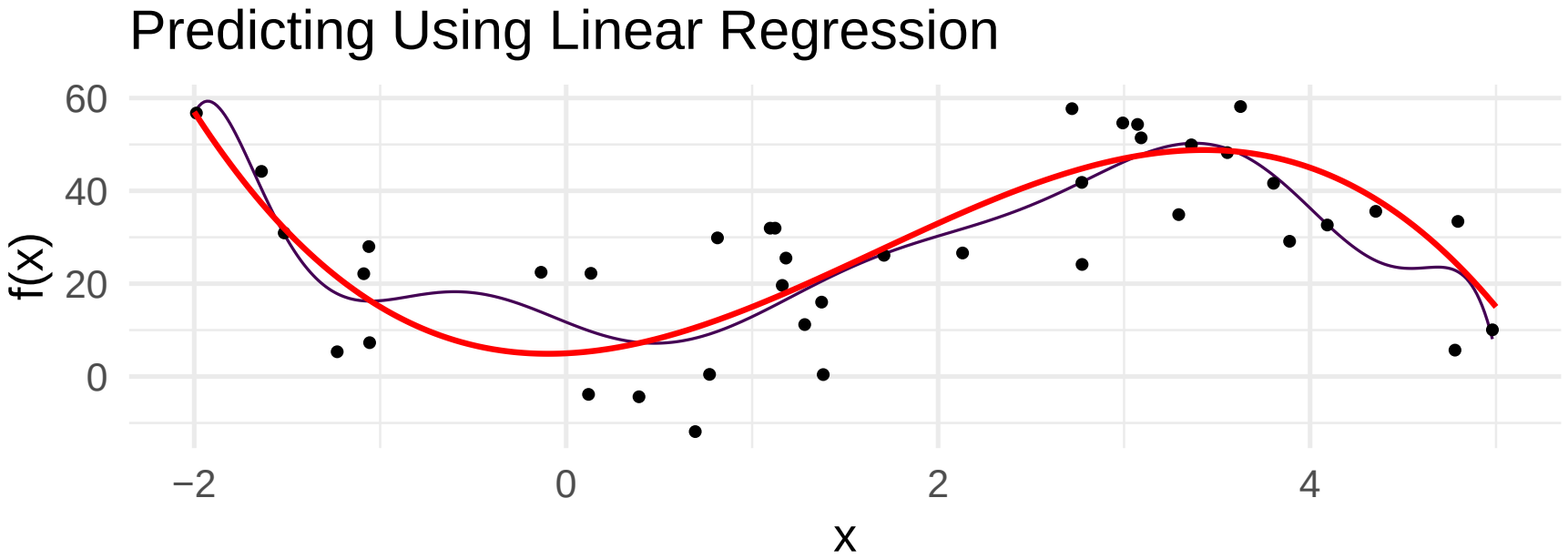


EXAMPLE: POLYNOMIAL RIDGE REGRESSION

Again, let us consider a dth-order polynomial

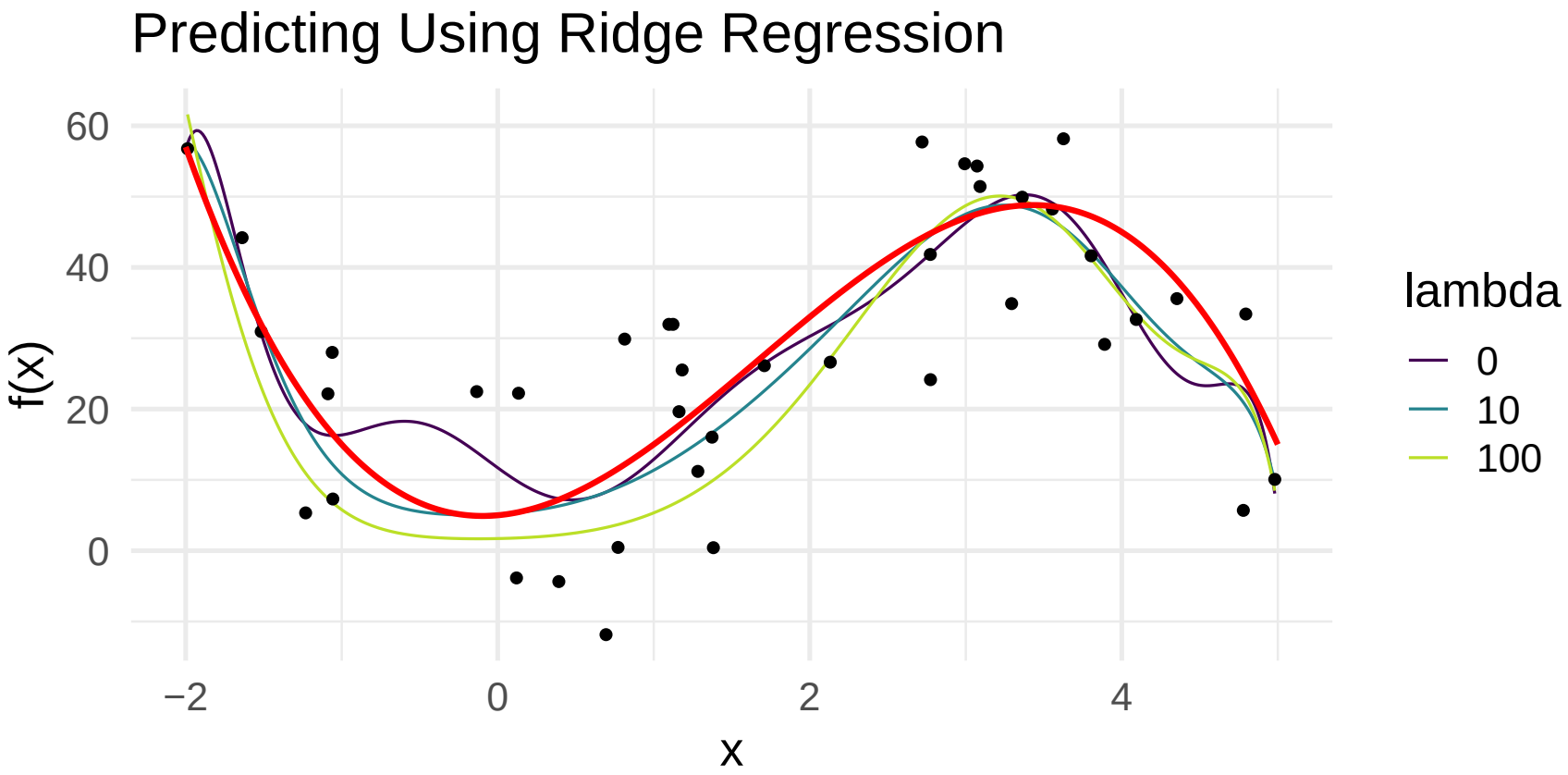
$$f(x) = \theta_0 + \theta_1 x + \cdots + \theta_d x^d = \sum_{j=0}^{d} \theta_j x^j.$$

True relationship is $f(x) = 5 + 2x + 10x^2 - 2x^3 + \epsilon$
Making predictions for $d = 10$ with linear regression tends to overfit:



EXAMPLE: POLYNOMIAL RIDGE REGRESSION

Using Ridge Regression with different penalty terms approximates the real data generating process better:



LASSO REGRESSION

Another shrinkage method is the so-called **Lasso regression**, which uses a L_1 penalty on $\boldsymbol{\theta}$ $^{(*)}$:

$$\begin{aligned} \hat{\boldsymbol{\theta}}_{\text{Lasso}} &= \arg\min_{\boldsymbol{\theta}} \left\{ \sum_{i=1}^{n} \left(y^{(i)} - f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta} \right) \right)^2 + \lambda ||\boldsymbol{\theta}||_1 \right\} \\ &= \arg\min_{\boldsymbol{\theta}} \left(\mathbf{y} - \mathbf{X}\boldsymbol{\theta} \right)^\top \left(\mathbf{y} - \mathbf{X}\boldsymbol{\theta} \right) + \lambda \|\boldsymbol{\theta}\|_1 . \end{aligned}$$

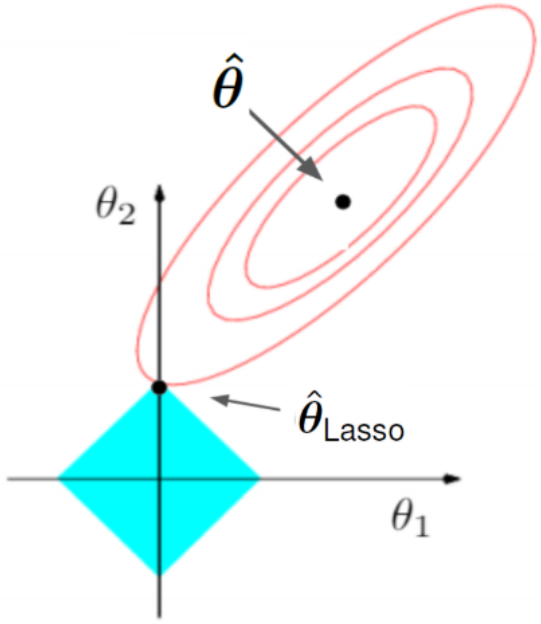
Note that optimization now becomes much harder. $\mathcal{R}_{\text{reg}}(\boldsymbol{\theta})$ is still convex, but we have moved from an optimization problem with an analytical solution towards an undifferentiable problem.

$^{(*)}$ $J(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1 = |\theta_1| + |\theta_2| + ... + |\theta_p|$

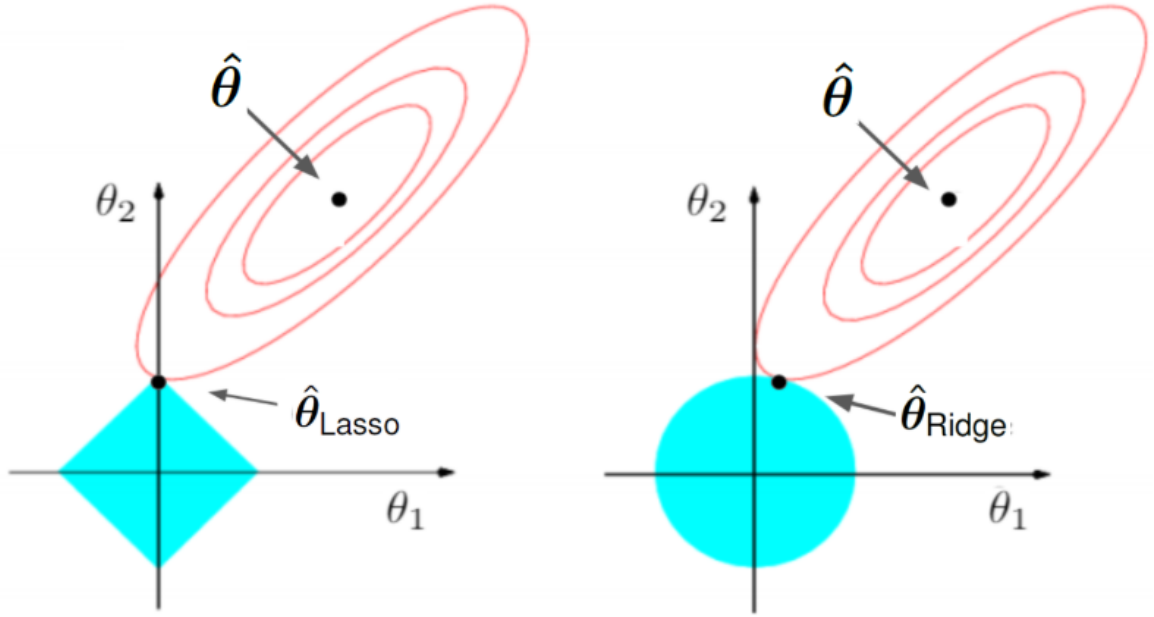
LASSO REGRESSION

Analogously, the problem can be formulated equivalently as a constrained optimization problem (for an appropriate choice of t):

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \sum_{i=1}^{n} \left(y^{(i)} - f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta} \right) \right)^2 \\ \text{subject to:} \quad & \|\boldsymbol{\theta}\|_1 \leq t \end{aligned}$$

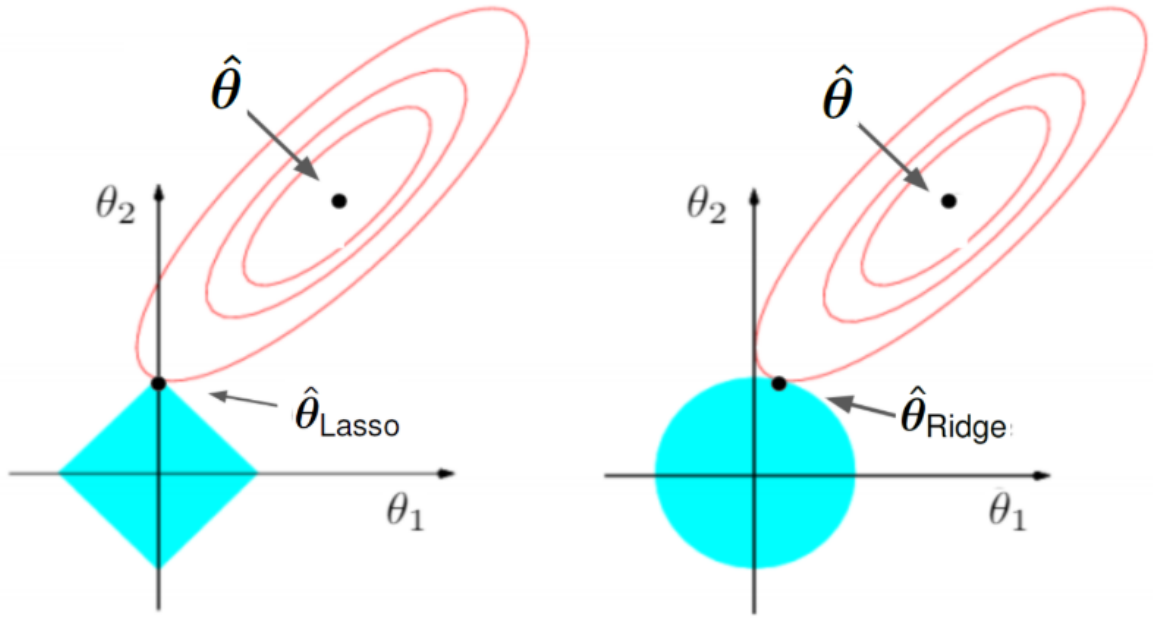


LASSO VS. RIDGE REGRESSION



- The ellipses are the level curves of the unregularized loss $\mathcal{R}_{\text{emp}}(\theta)$ and $\hat{\theta}$ is the location of the minimum.
- The regions in cyan represent the values of θ that satisfy the inequality constraint.
- As λ increases, the area of the region shrinks.

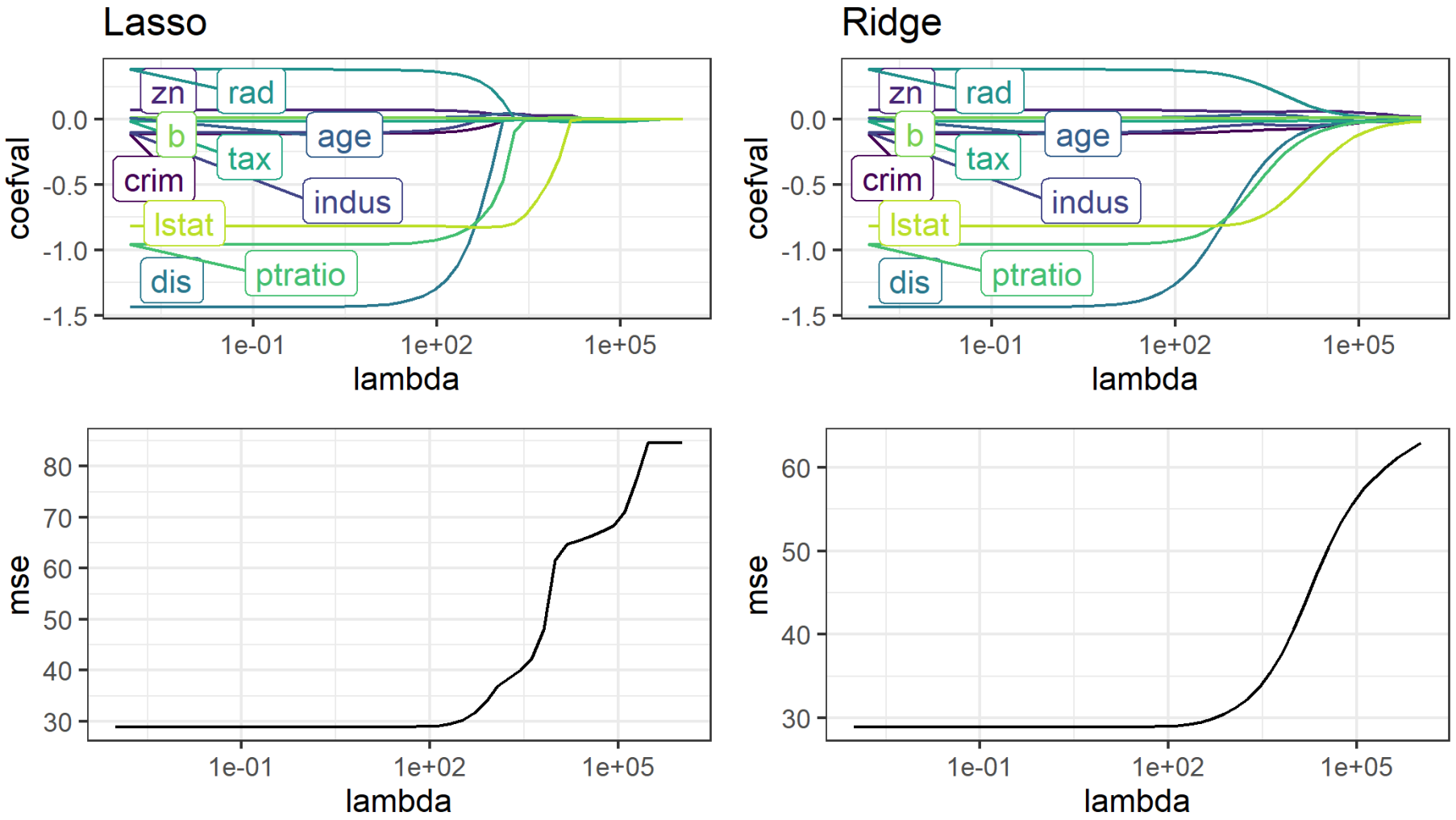
LASSO VS. RIDGE REGRESSION



- In both cases, the solution which minimizes $\mathcal{R}_{\text{reg}}(\boldsymbol{\theta})$ is always a point on the boundary of the feasible region (for sufficiently large λ).
- In the case of Lasso, the solution is more likely to be one of the vertices of the constraint region. This induces sparsity and is a form of variable selection.
- As expected, $\hat{\boldsymbol{\theta}}_{\text{Lasso}}$ and $\hat{\boldsymbol{\theta}}_{\text{Ridge}}$ have smaller parameter norms than $\hat{\boldsymbol{\theta}}$.

LASSO VS. RIDGE REGRESSION

Coefficient paths for different λ values for Ridge and Lasso, on Boston Housing (few features removed for readability) along with the cross-validated MSE. We cannot really overfit here with an unregularized linear model as the task is so low-dimensional.



LASSO VS. RIDGE REGRESSION

Comments regarding Ridge / Lasso implementations:

- Note that very often we do not include θ_0 in the penalty term $J(\boldsymbol{\theta})$ (but this can be implementation-dependent).
- These methods are typically not equivariant under scaling of the inputs, so one usually standardizes the features.

Comments on **Ridge vs. Lasso:**

- Often neither one is overall better.
- Lasso can set some coefficients to zero, thus performing variable selection, while ridge regression usually leads to smaller estimated coefficients, but still dense $\boldsymbol{\theta}$ vectors.

LASSO VS. RIDGE REGRESSION

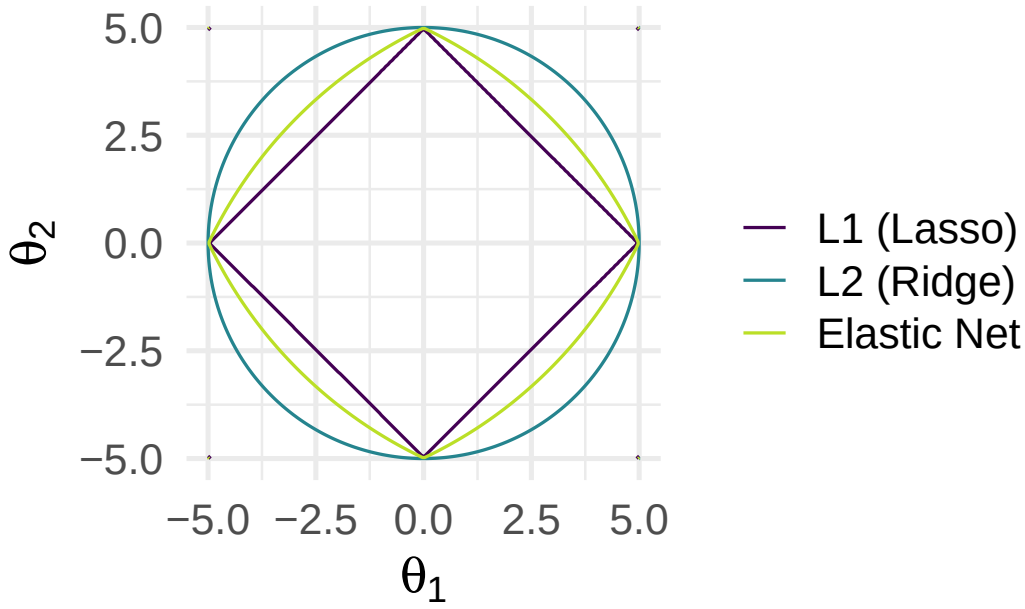
- Both methods can handle correlated predictors, but they solve the multicollinearity issue differently:
  - In ridge regression, the coefficients of correlated predictors are similar
  - In Lasso, one of the correlated predictors has a larger coefficient, while the rest are (nearly) zeroed.
- Lasso tends to do well if there are a small number of significant parameters and the others are close to zero (ergo: when only a few predictors actually influence the response)
- Ridge works well if there are many large parameters of about the same value (ergo: when most predictors impact the response).

Idea: Combine the two!

ELASTIC NET

Elastic Net combines the L_1 and L_2 penalty:

$$\mathcal{R}_{\text{elnet}}(\boldsymbol{\theta}) = \sum_{i=1}^{n} (y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)})^2 + \lambda_1 ||\boldsymbol{\theta}||_1 + \lambda_2 \|\boldsymbol{\theta}\|_2^2.$$



LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Introduction to Machine Learning

Boosting - Basic

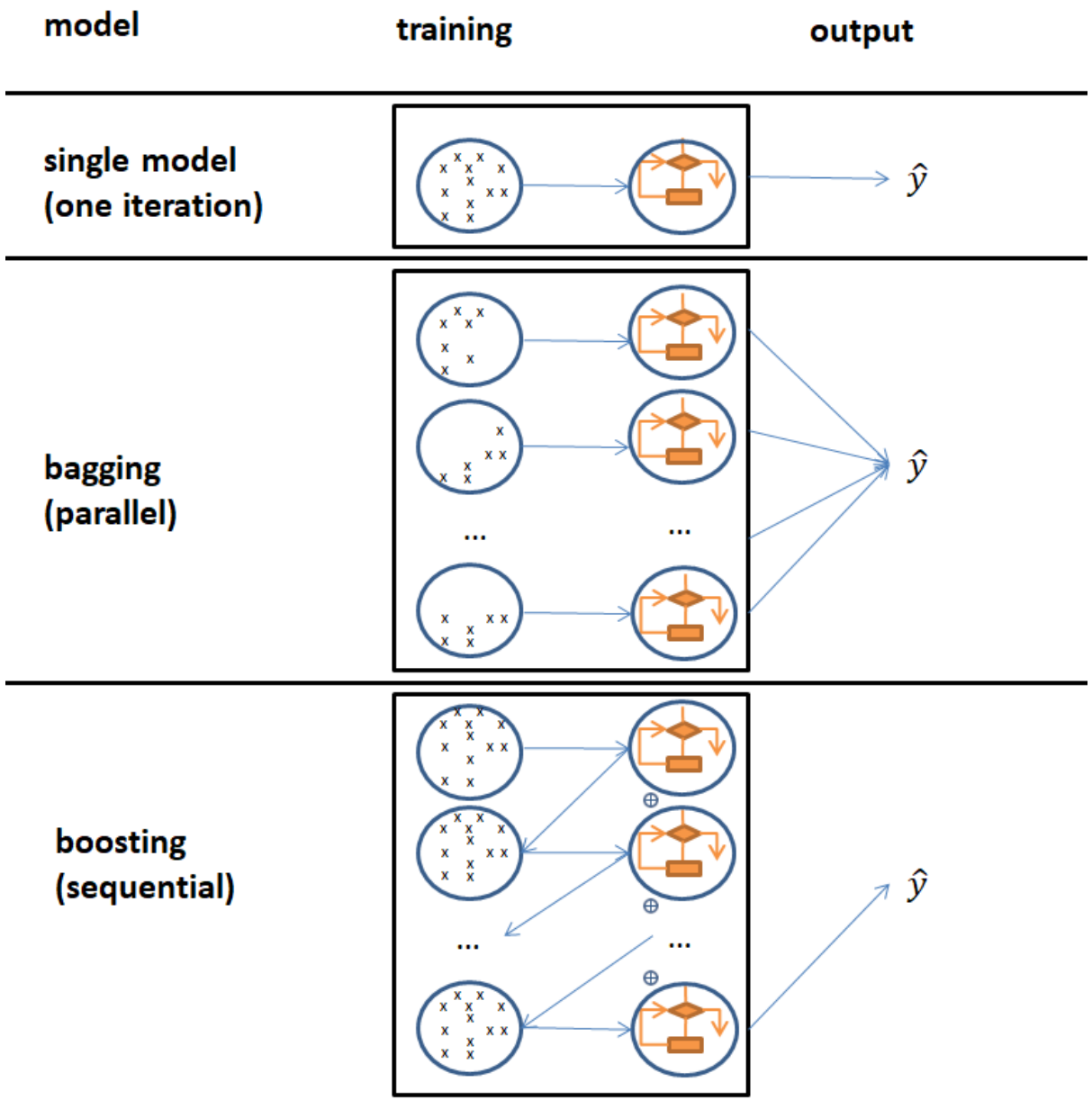
Department of Statistics – LMU Munich

Introduction

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.

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.”

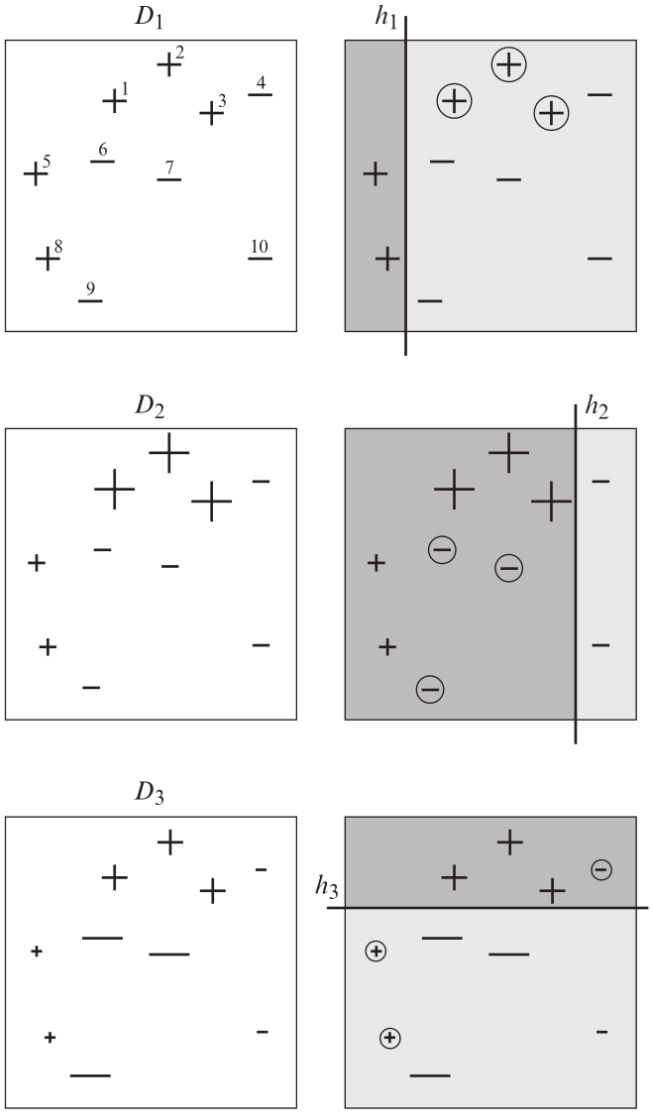
THE BOOSTING ANSWER - ADABOOST

- 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 $b(\mathbf{x}, \theta^{[m]})$, instead.
- Predictions from all base models $b^{[m]}$ are combined to form:

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

- 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 $h(\mathbf{x}) = \text{sign}(f(\mathbf{x})) \in \{-1, 1\}$.

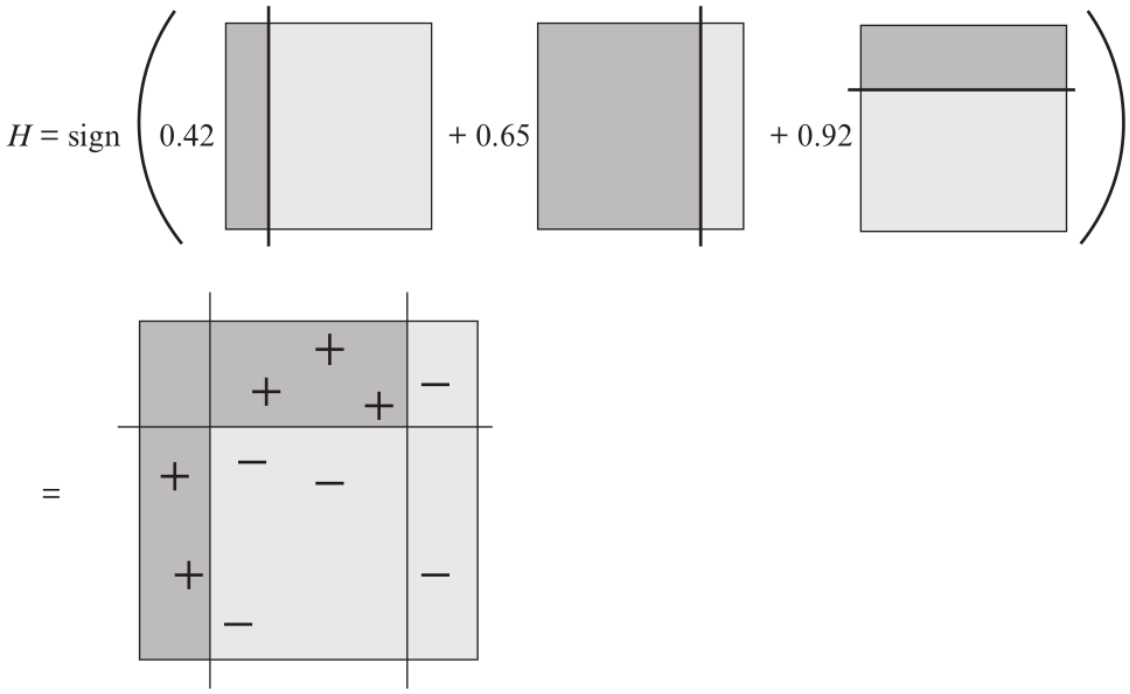
ADABOOST ILLUSTRATION



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

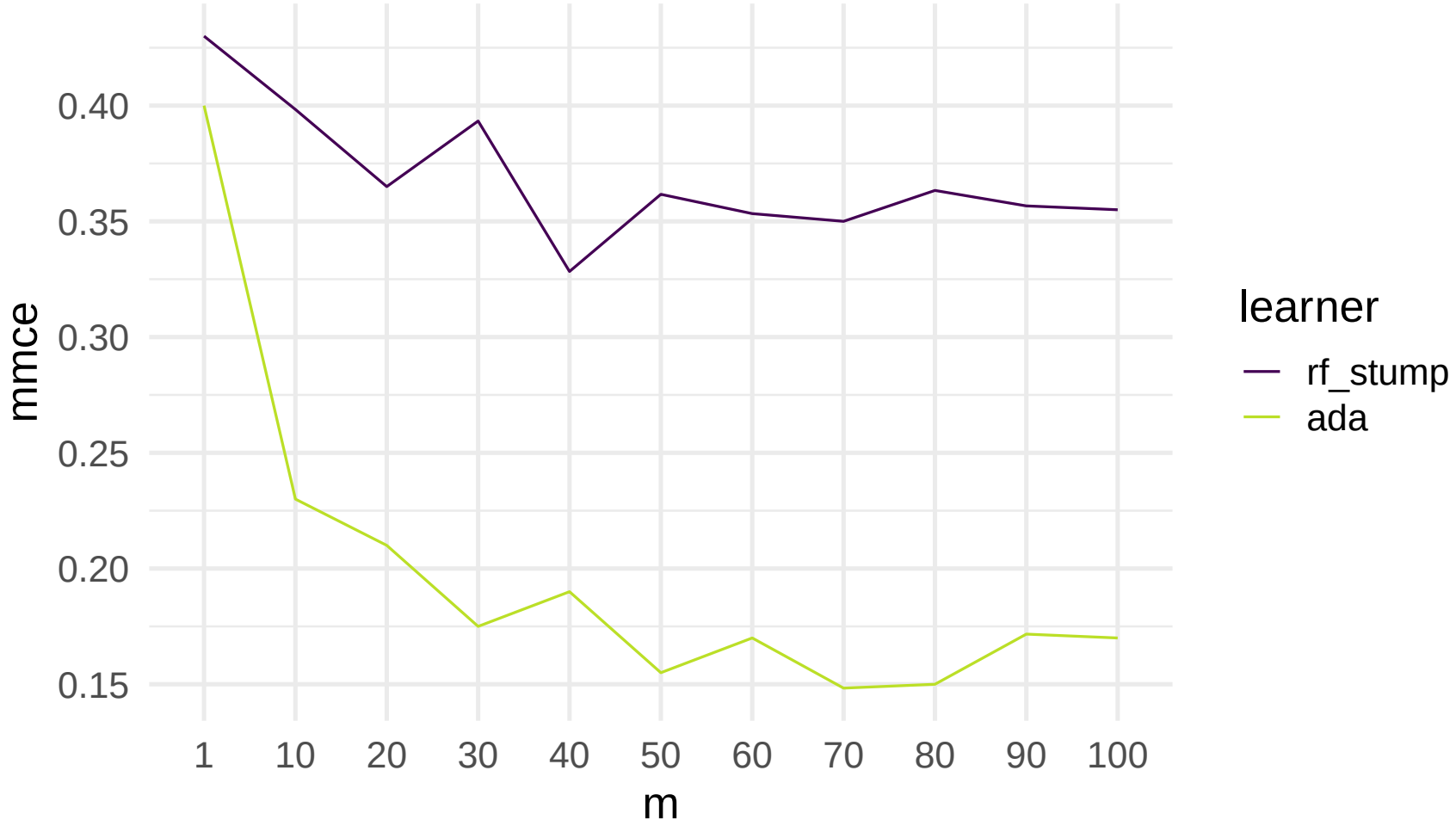


Schapire, Boosting, 2012.

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

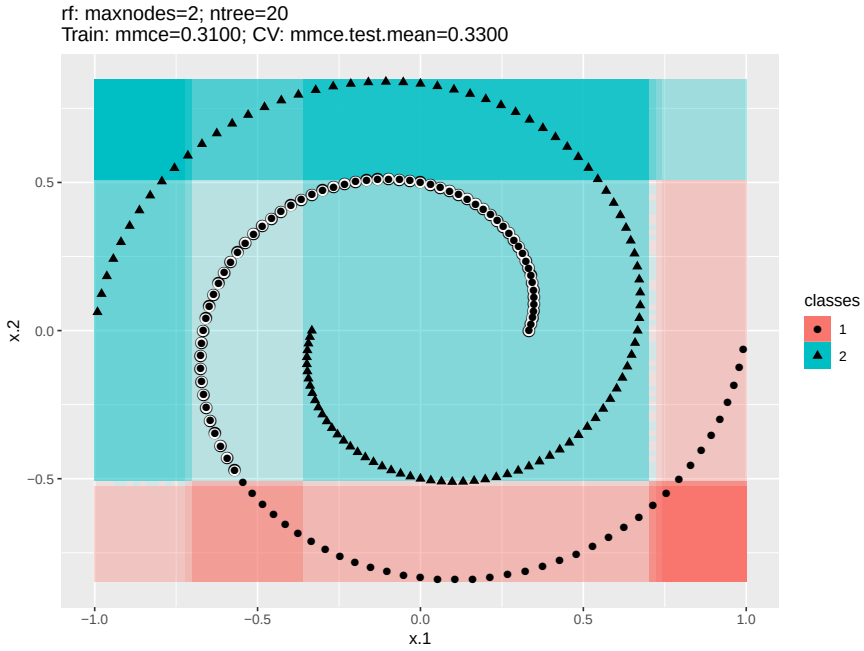
EXAMPLE: BAGGING VS BOOSTING

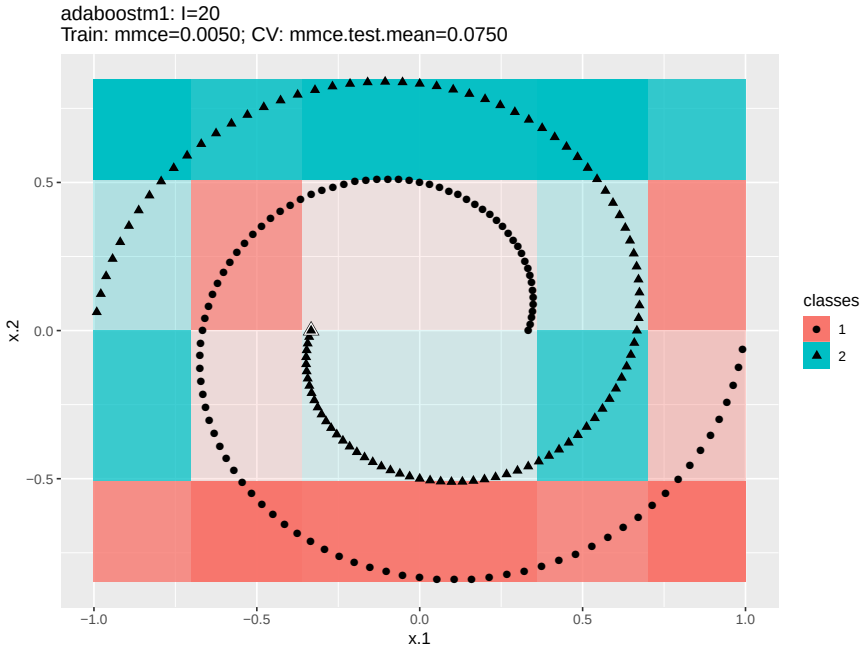
Random Forest versus AdaBoost (both with stumps) on Spirals data from mlbench ($n = 200$, sd$= 0$). Performance (mmce) is measured with 3×10 repeated CV.



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.

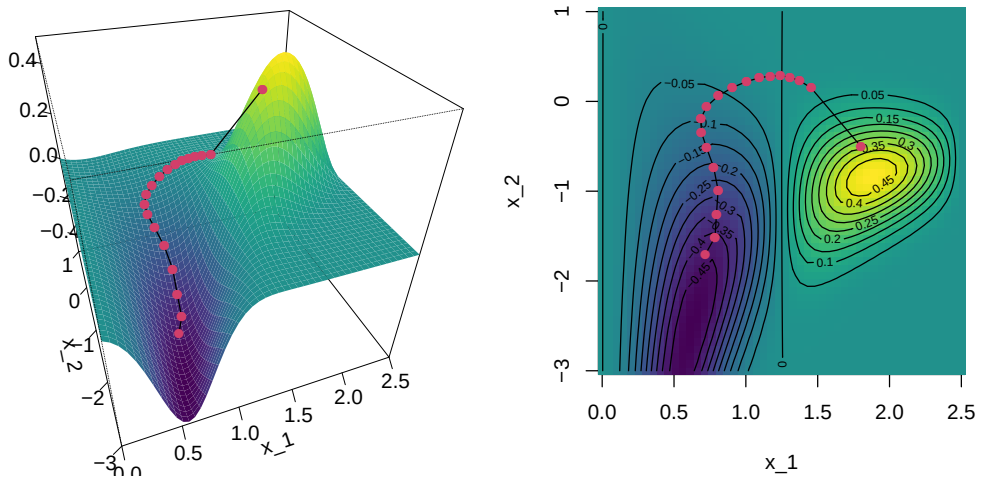
Gradient Boosting

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:

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

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 $\hat{f}^{[0]}(\mathbf{x})$
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 $\hat{f}^{[m]}(\mathbf{x}) \leftarrow \hat{f}^{[m-1]}(\mathbf{x}) + \hat{\beta}^{[m]} b\left(\mathbf{x}, \hat{\theta}^{[m]}\right)$
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.

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

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 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:

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

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...?

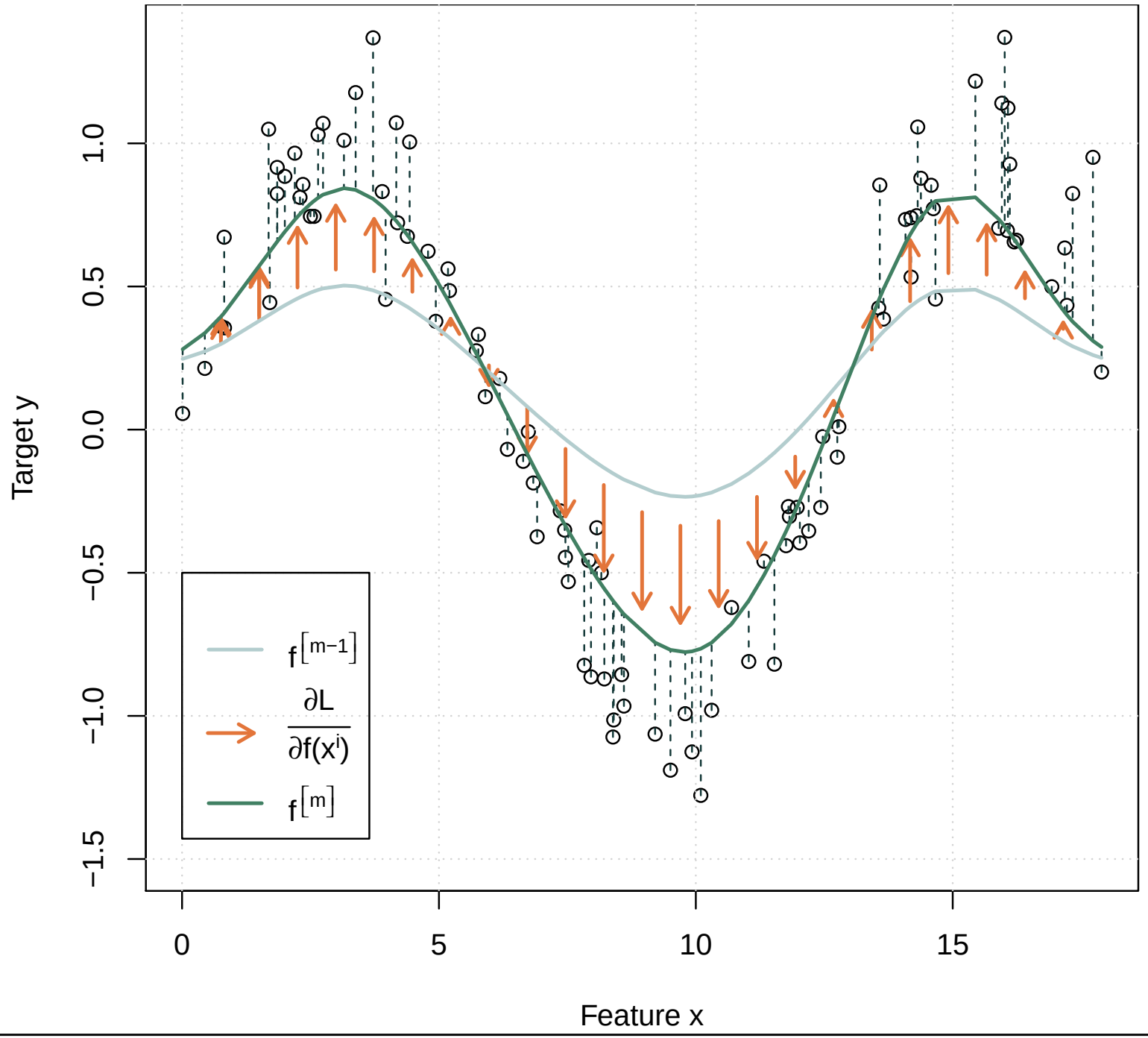
So, we restrict our additive components to $b\left(\mathbf{x}, \theta^{[m]}\right) \in \mathcal{B}$.
The pseudo-residuals are calculated exactly as stated above, then we fit a regression model $b(\mathbf{x}, \theta^{[m]})$ to them:

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

So, evaluated on the training data, our $b(\mathbf{x}, \theta^{[m]})$ 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

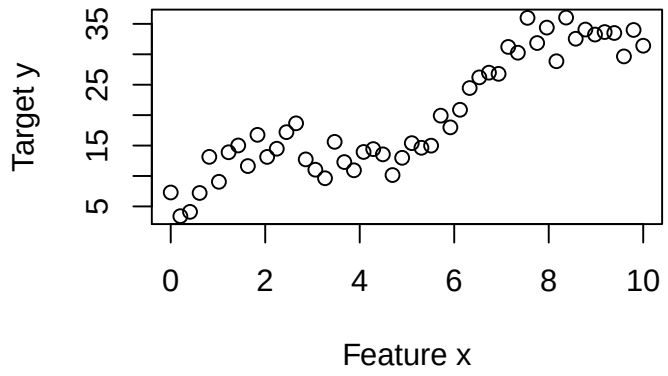
Algorithm 2 Gradient Boosting Algorithm.

1: Initialize $\hat{f}^{[0]}(\mathbf{x}) = \arg\min_\theta \sum_{i=1}^{n} L(y^{(i)}, b(\mathbf{x}^{(i)}, \theta))$
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: $\hat{\beta}^{[m]} = \arg\min_\beta \sum_{i=1}^{n} L(y^{(i)}, f^{[m-1]}(x) + \beta b(\mathbf{x}, \hat{\theta}^{[m]}))$
7: Update $\hat{f}^{[m]}(\mathbf{x}) = \hat{f}^{[m-1]}(\mathbf{x}) + \hat{\beta}^{[m]} b(\mathbf{x}, \hat{\theta}^{[m]})$
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.

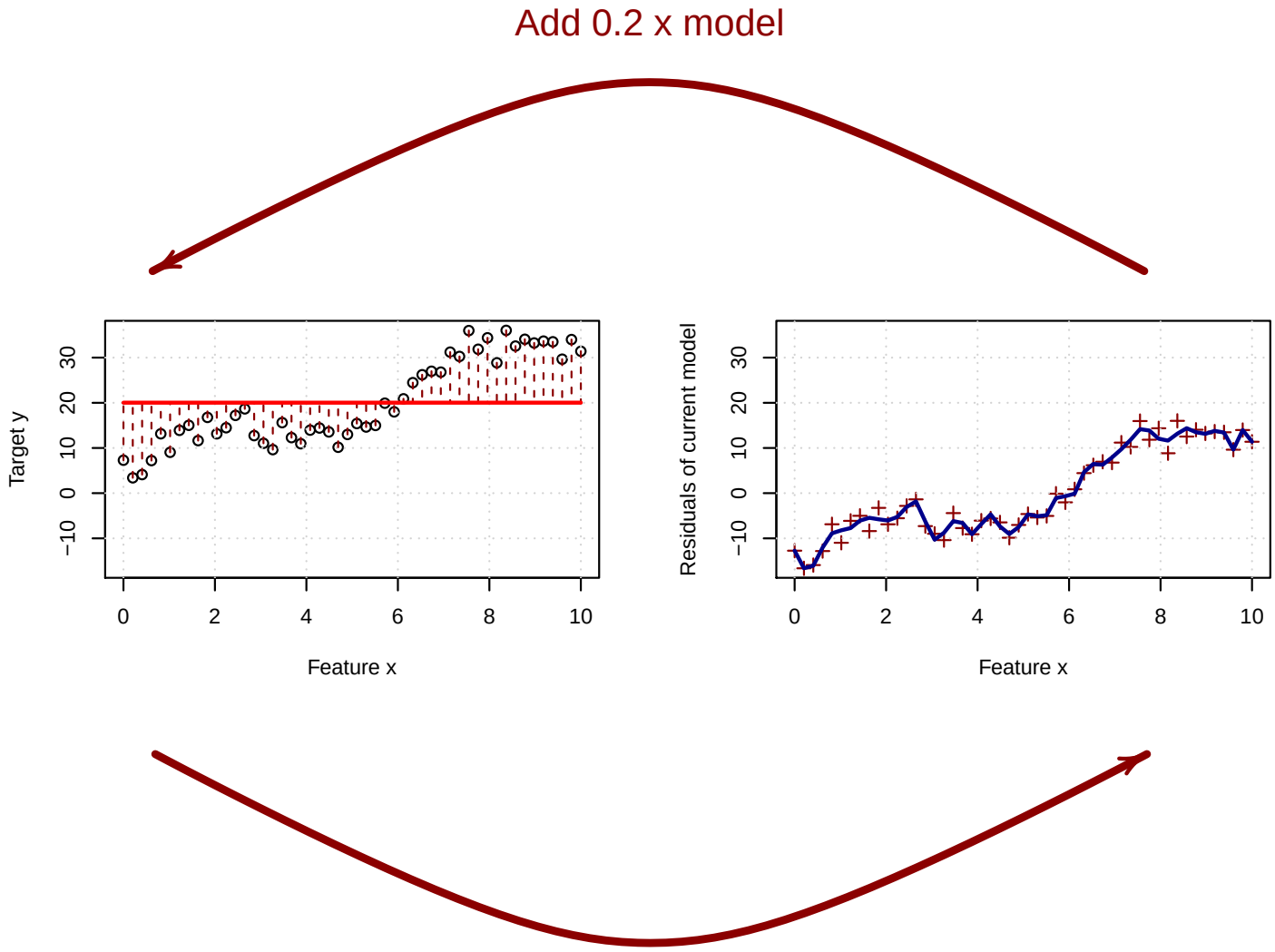
GRADIENT BOOSTING ILLUSTRATION - 1

Assume one feature x and a target y.



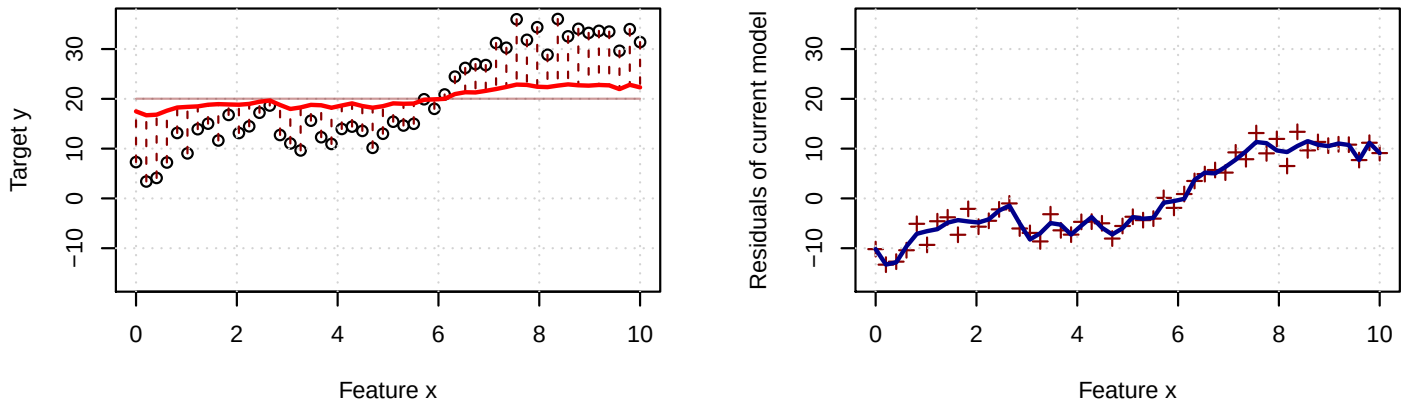
1. We start with the simplest model, the optimal constant w.r.t. the L_2 loss (mean of the target variable).
2. To improve the model we calculate the pointwise residuals on the training data $y^{(i)} - f\left(\mathbf{x}^{(i)}\right)$ and fit a GAM fitted on the residuals.
3. The GAM fitted on the residuals 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



GRADIENT BOOSTING ILLUSTRATION - 1

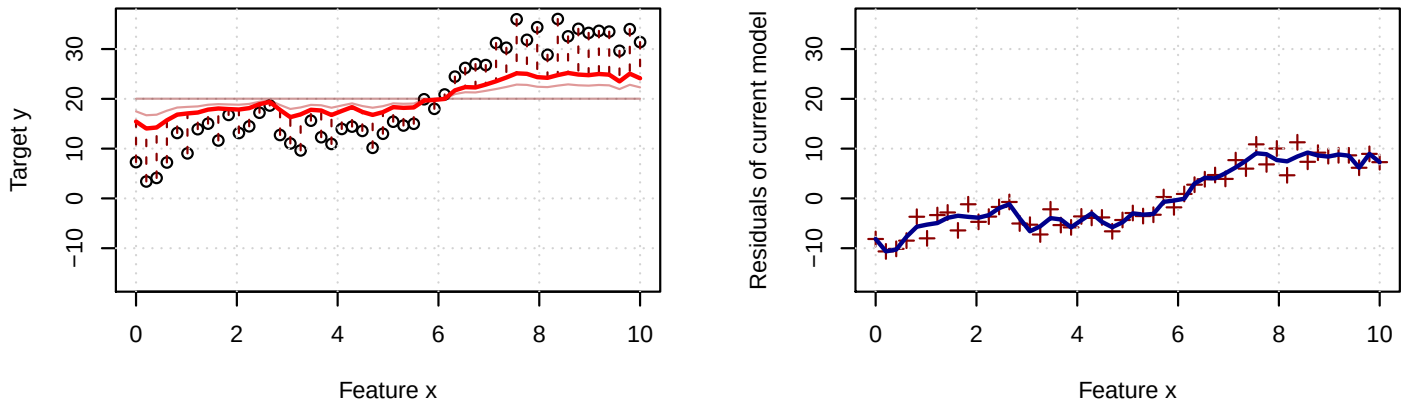
Add 0.2 x model



Calculate the residuals of the current model

GRADIENT BOOSTING ILLUSTRATION - 1

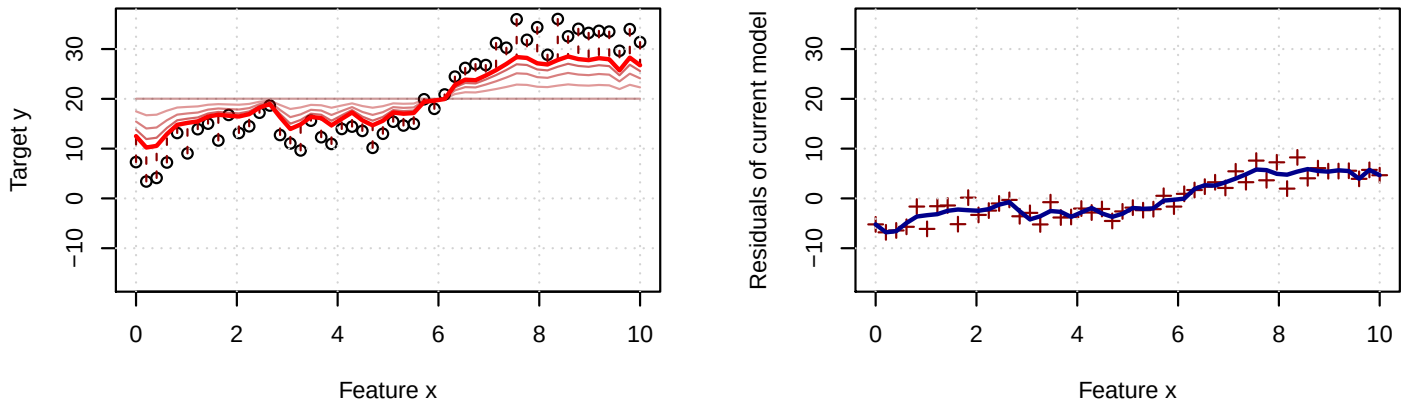
Add 0.2 x model



Calculate the residuals of the current model

GRADIENT BOOSTING ILLUSTRATION - 1

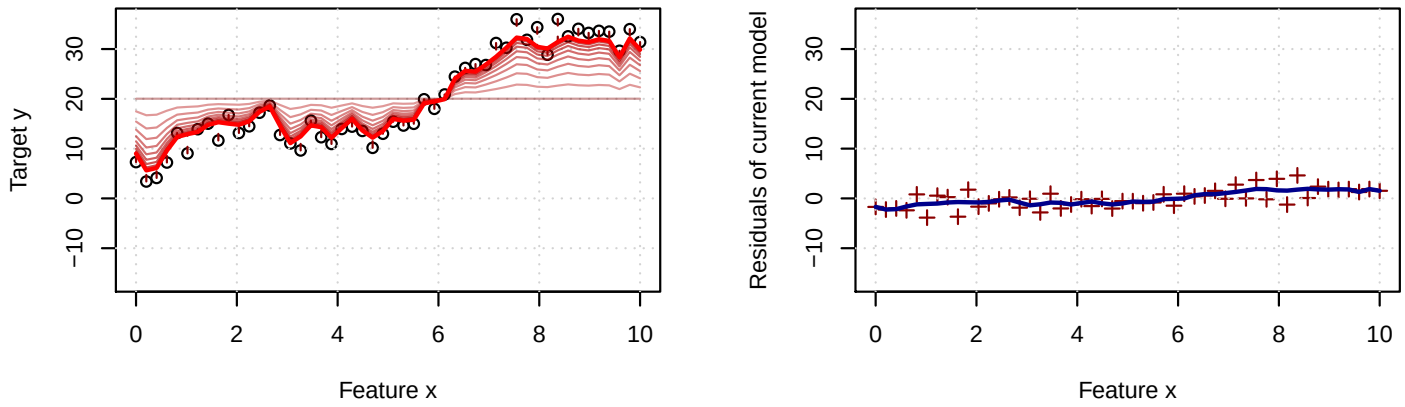
Add 0.2 x model



Calculate the residuals of the current model

GRADIENT BOOSTING ILLUSTRATION - 1

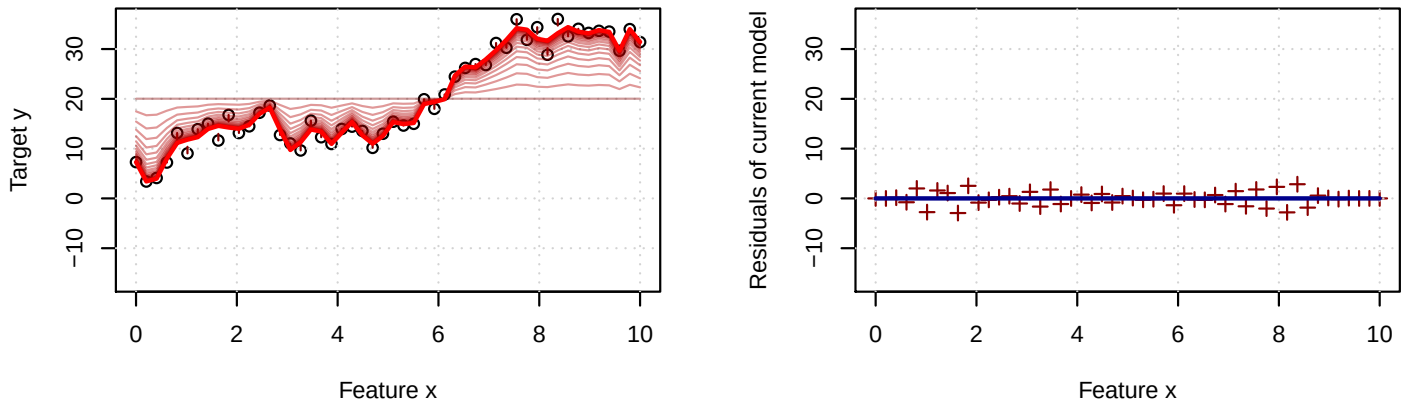
Add 0.2 x model



Calculate the residuals of the current model

GRADIENT BOOSTING ILLUSTRATION - 1

Add 0.2 x model



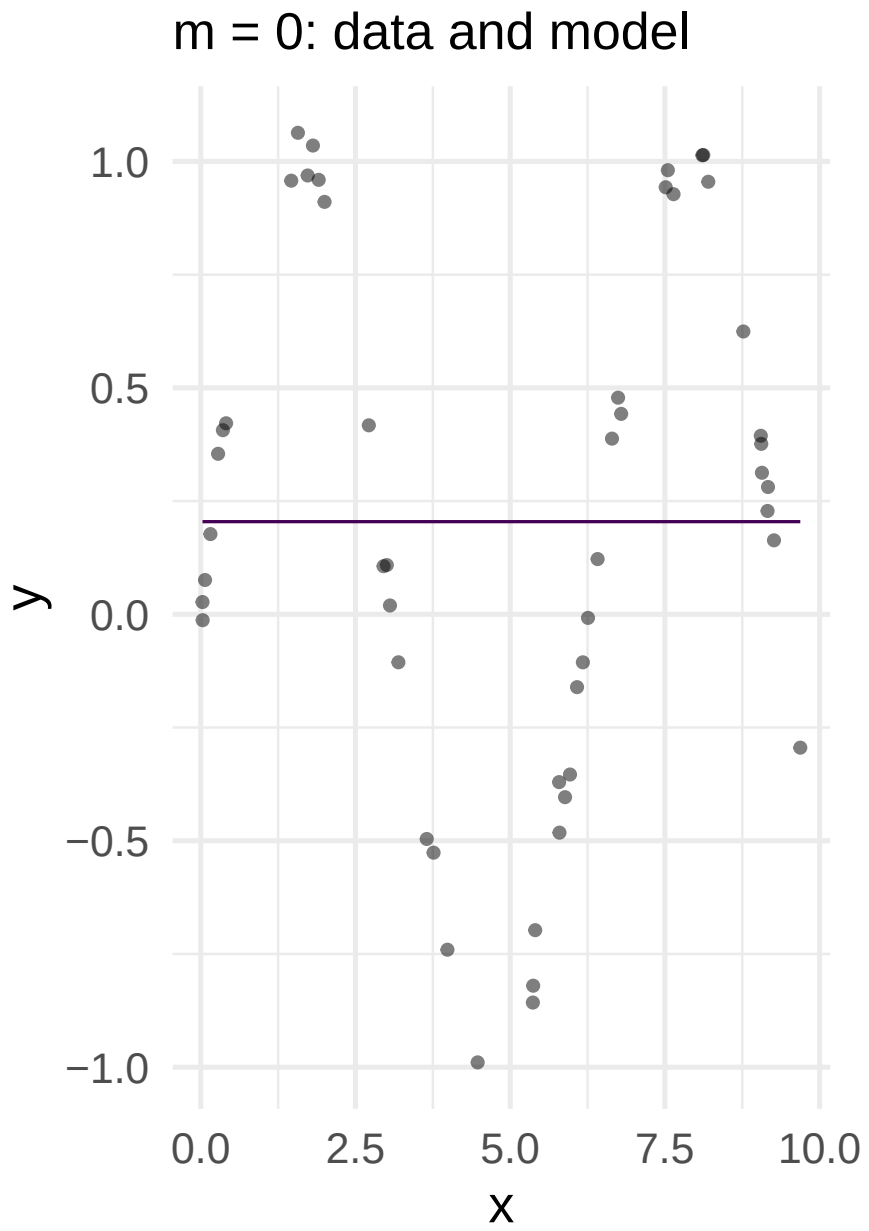
Calculate the residuals of the current model

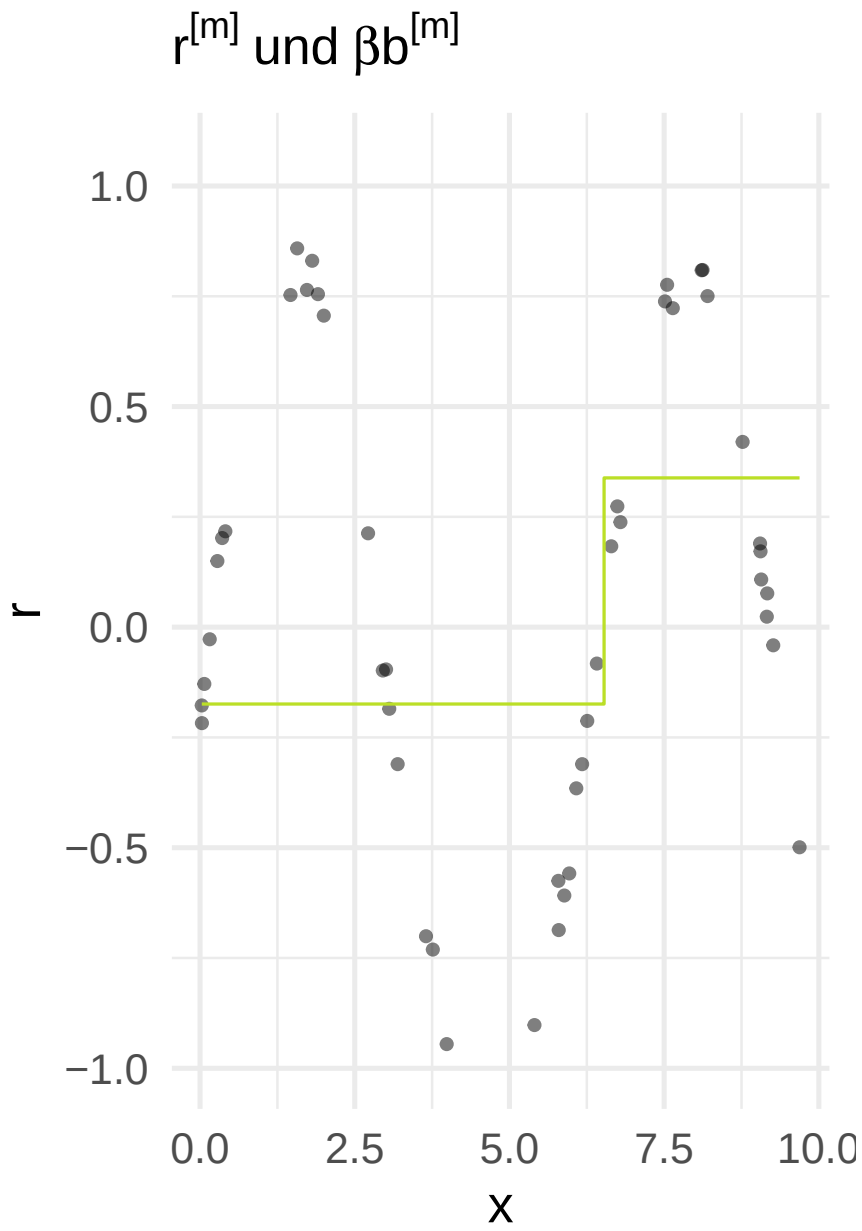
GRADIENT BOOSTING ILLUSTRATION - 2

We will consider a regression problem on the following slides:

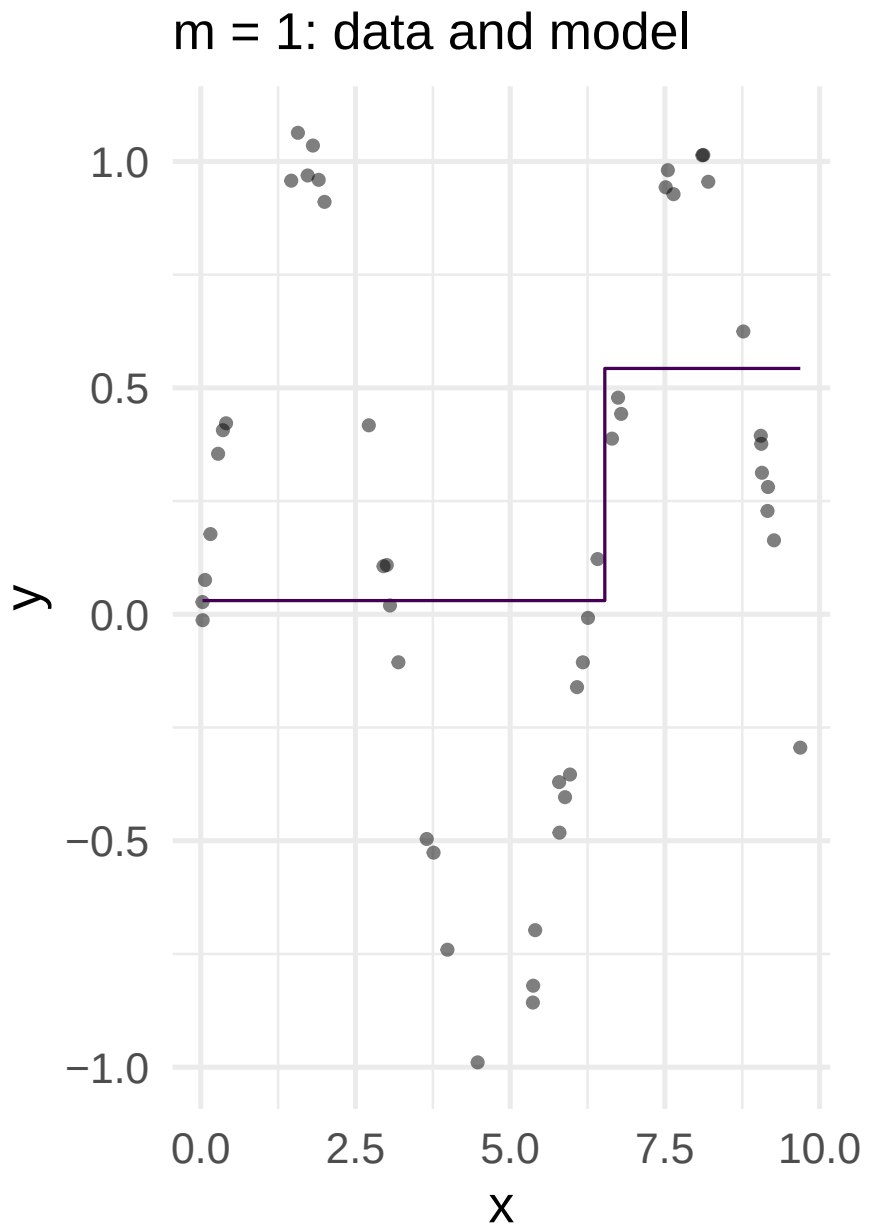
- We draw points from a sine-function with some additional noise.
- We use squared loss for L.
- Our base learner are tree stumps.
- The left plot shows the additive model learnt so far with the data, the right plot shows the residuals to which we fit the next base learner.
- The plot is generated by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

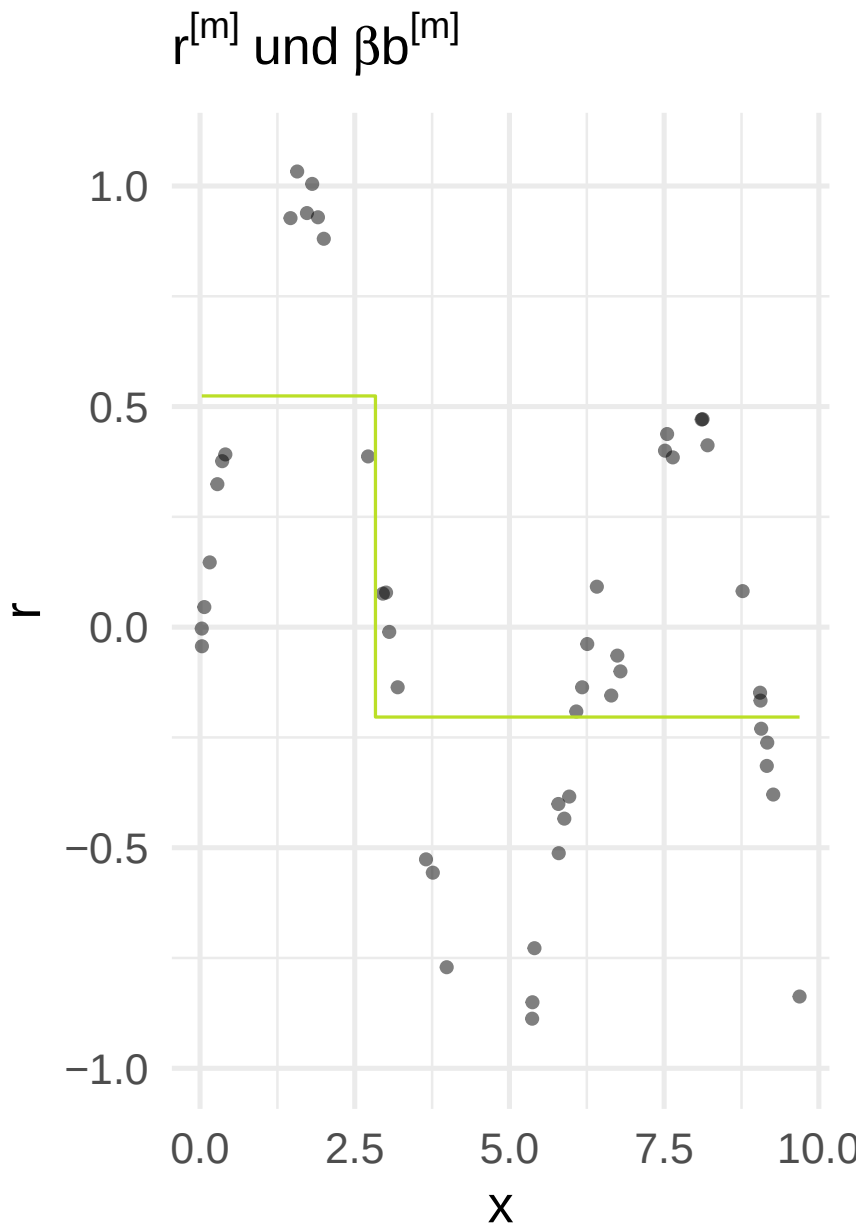
GRADIENT BOOSTING ILLUSTRATION - 2



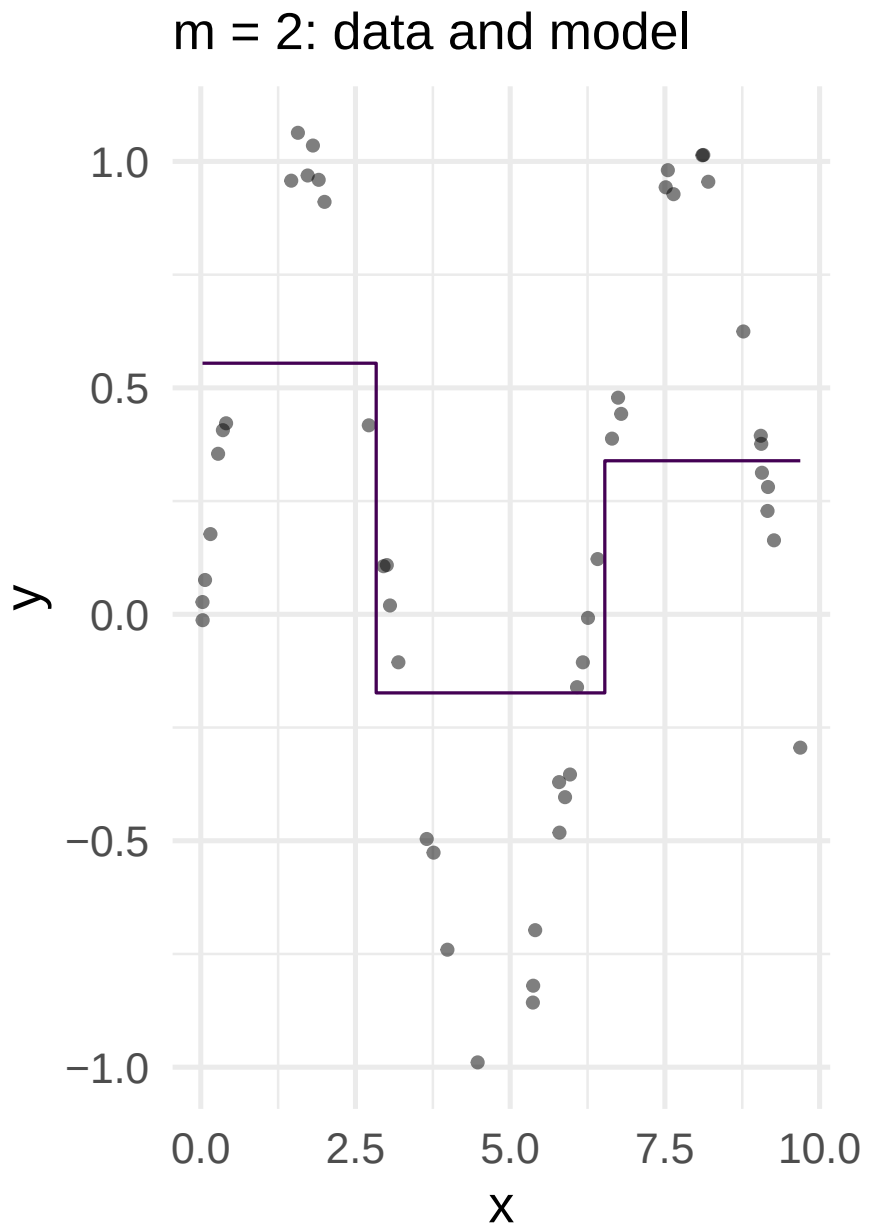


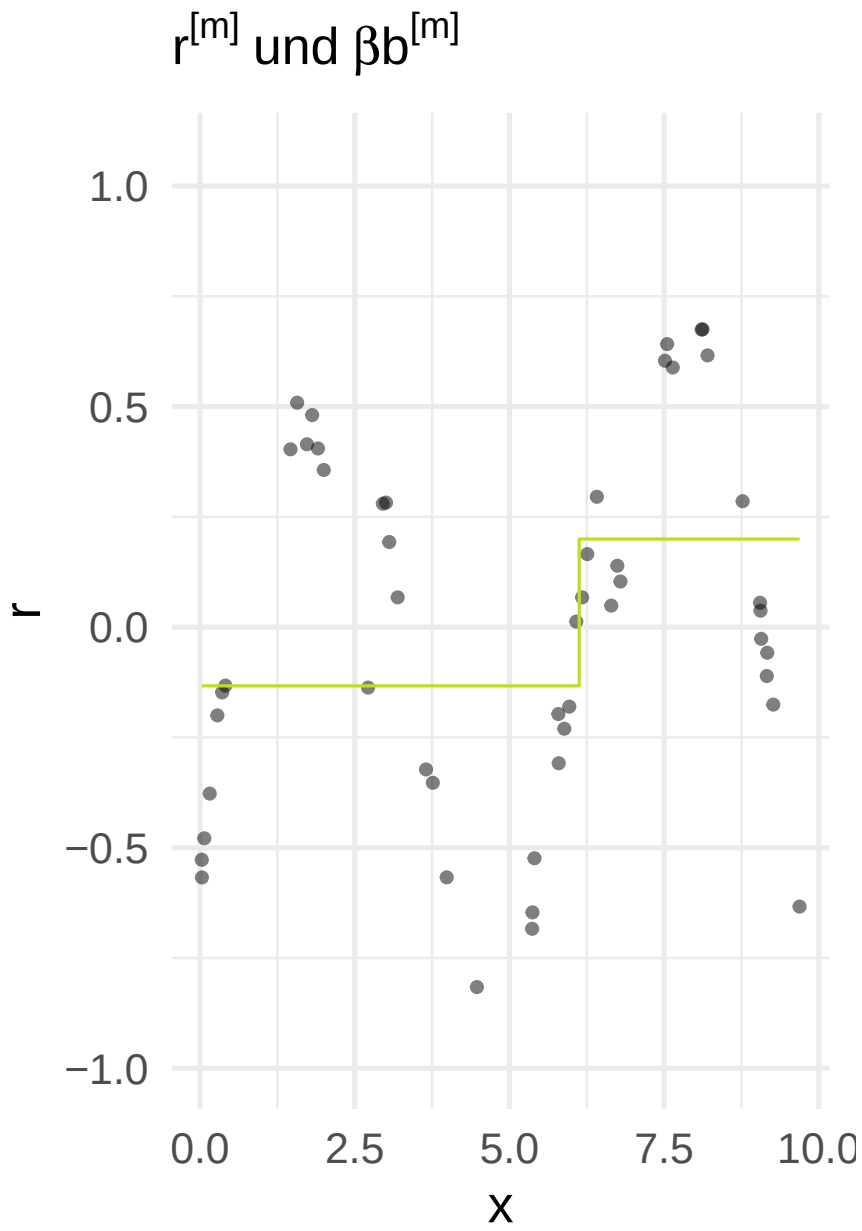
GRADIENT BOOSTING ILLUSTRATION - 2



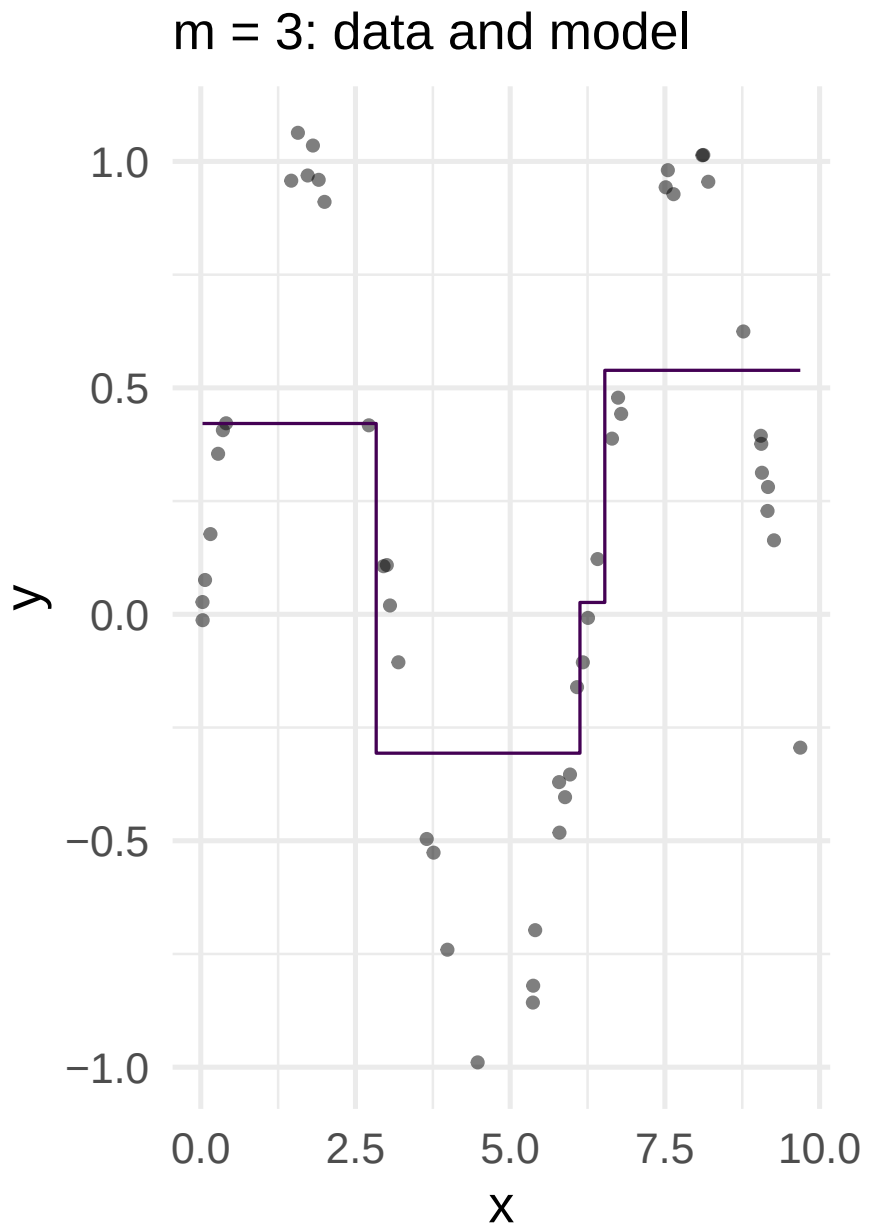


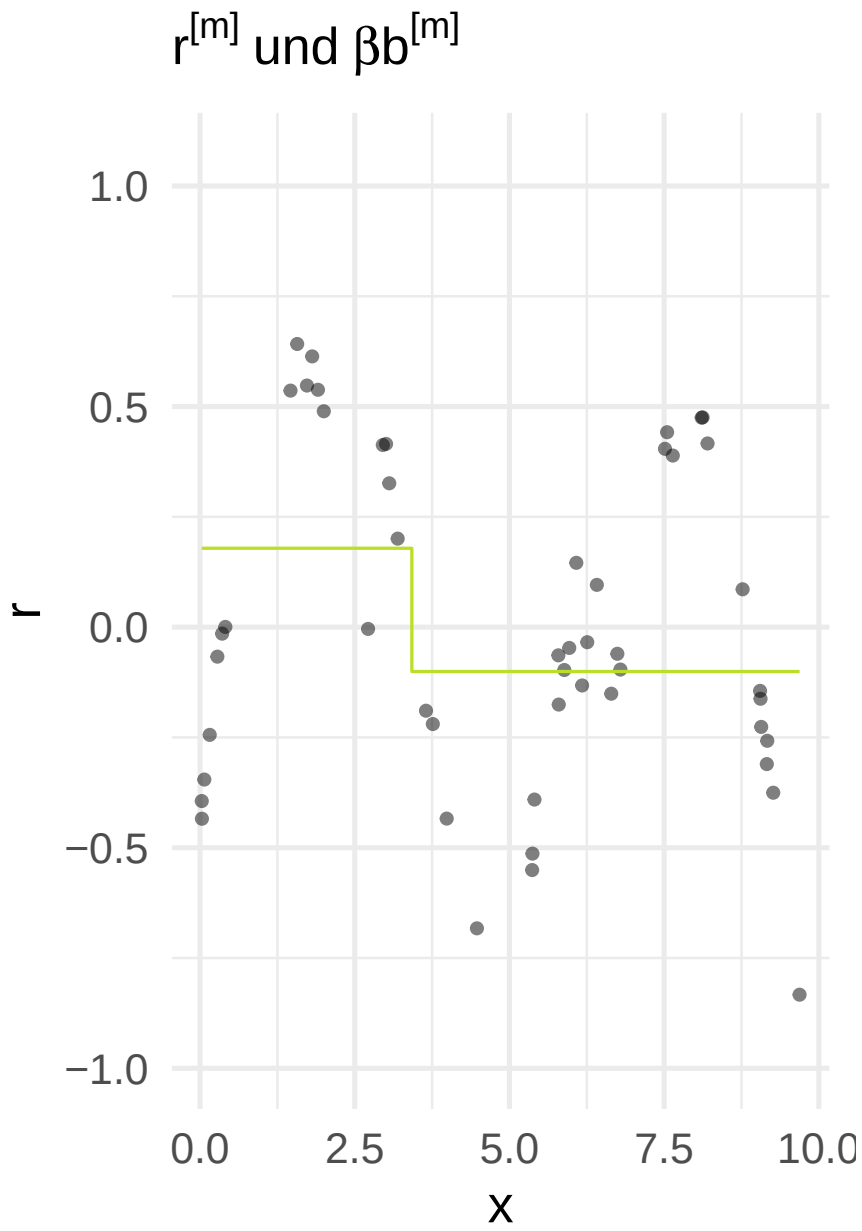
GRADIENT BOOSTING ILLUSTRATION - 2



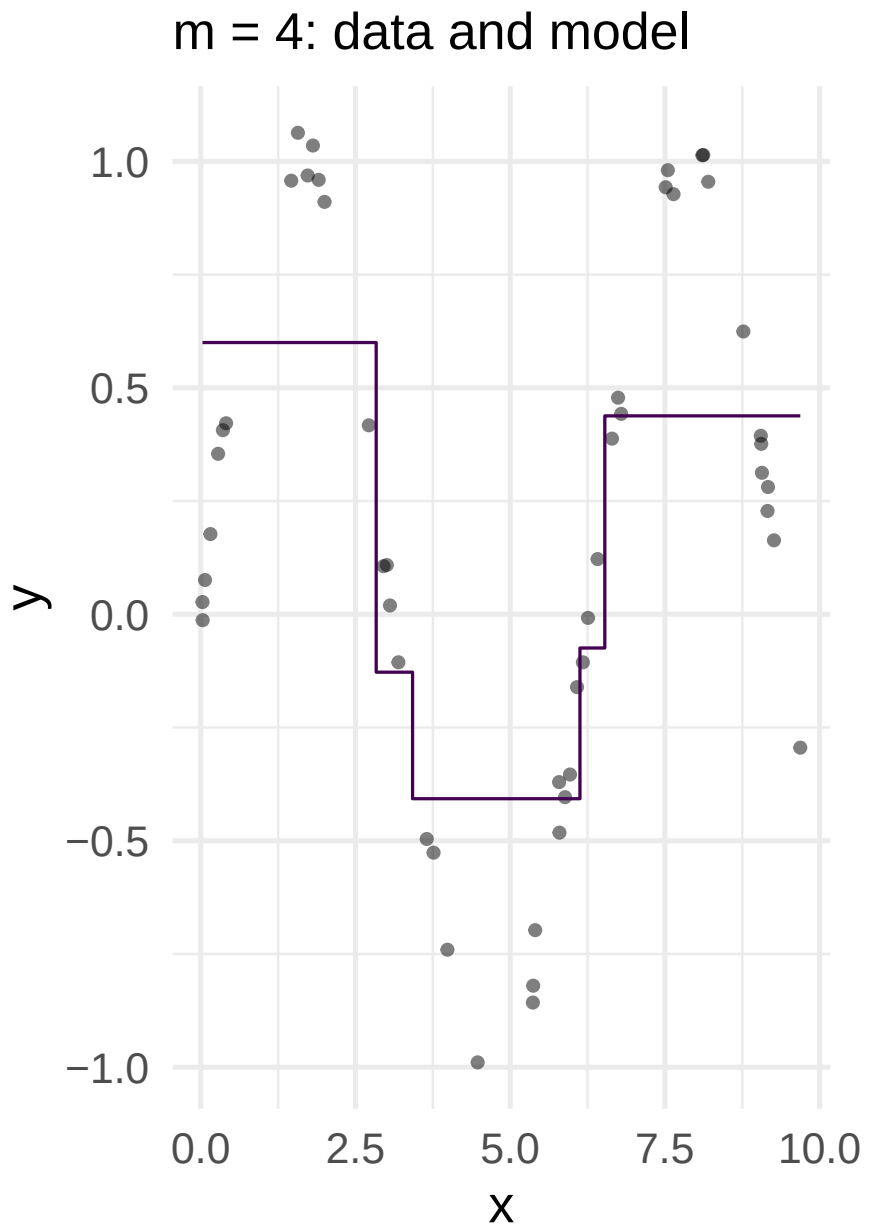


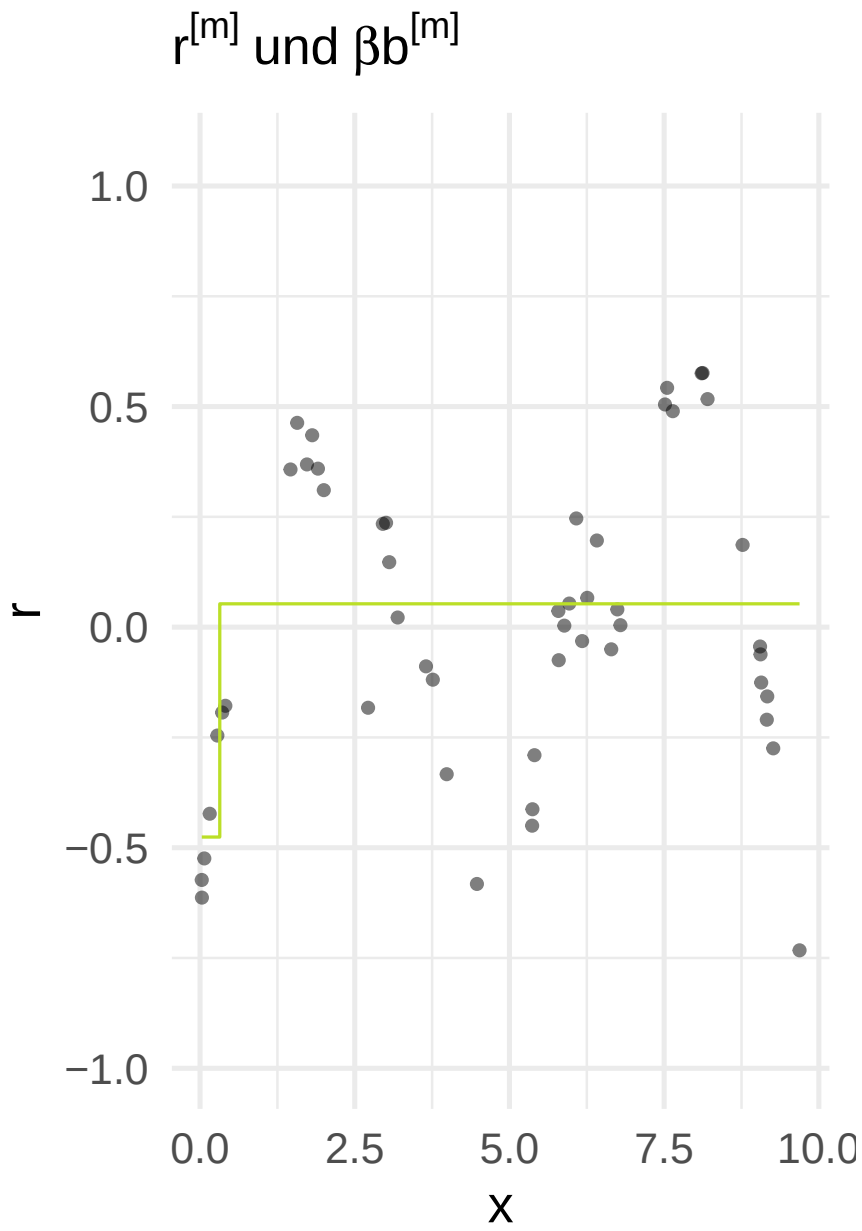
GRADIENT BOOSTING ILLUSTRATION - 2



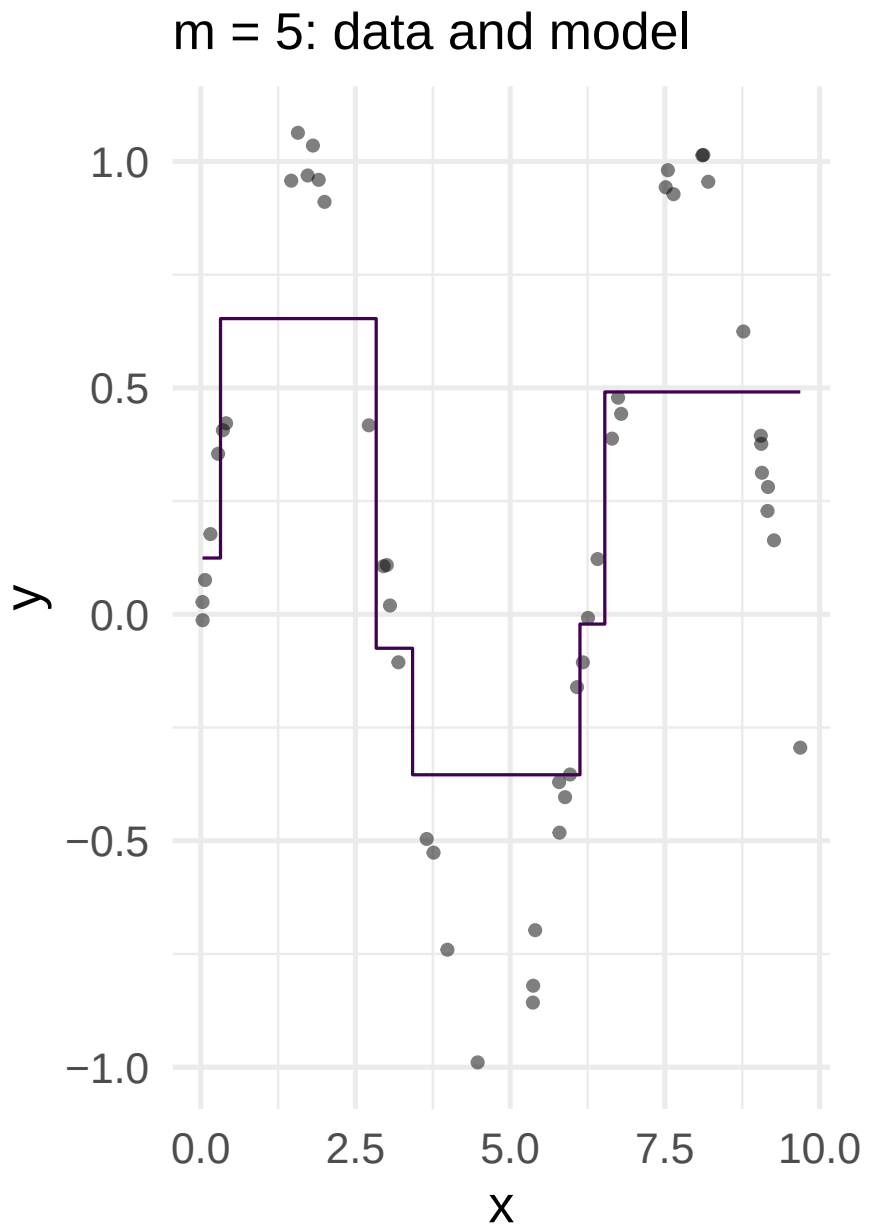


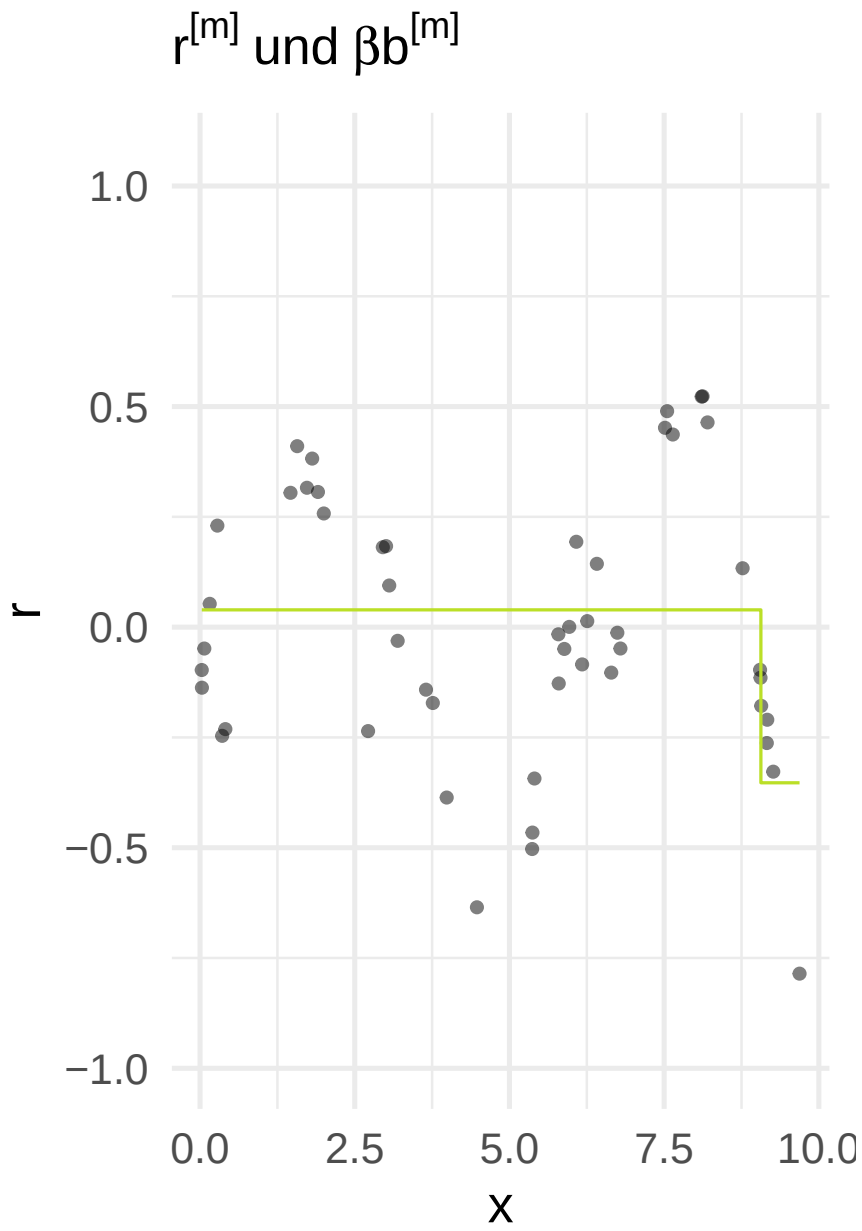
GRADIENT BOOSTING ILLUSTRATION - 2



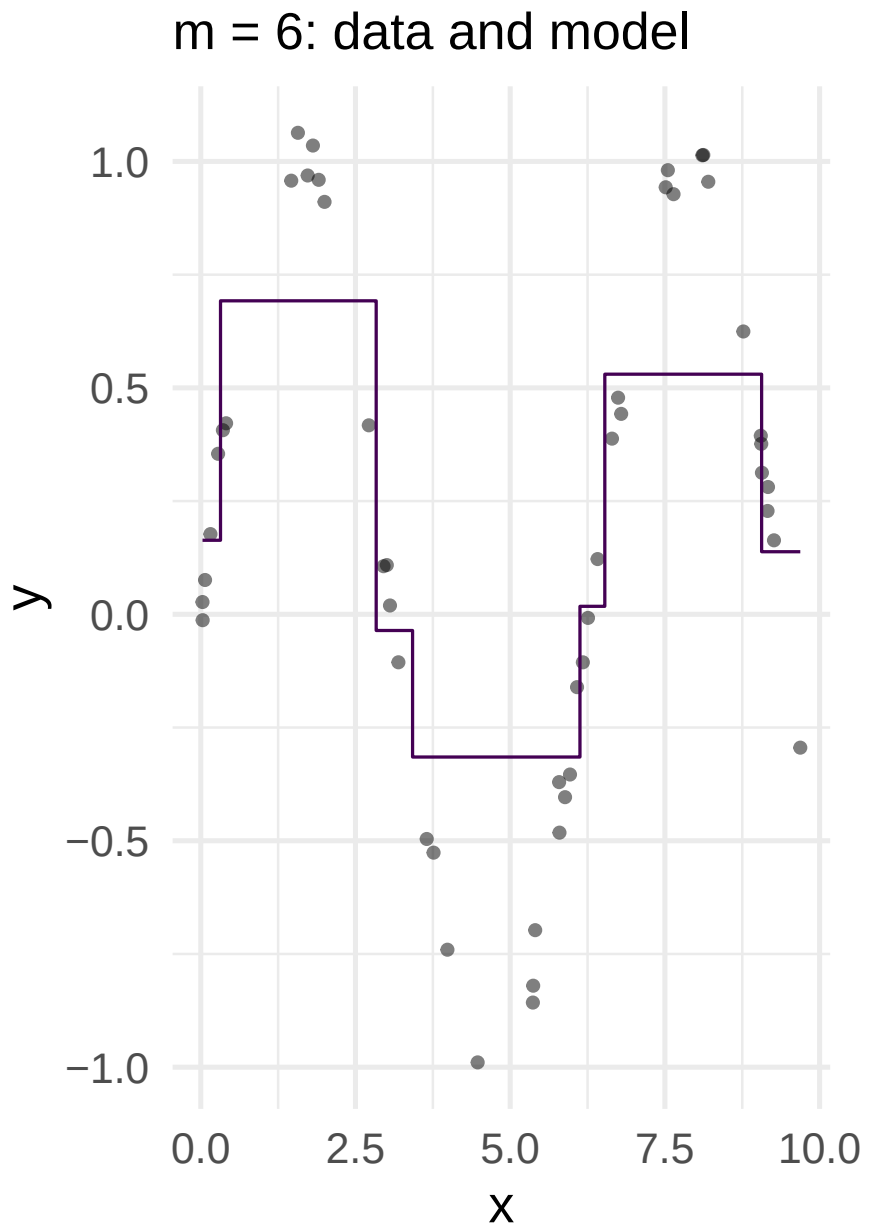


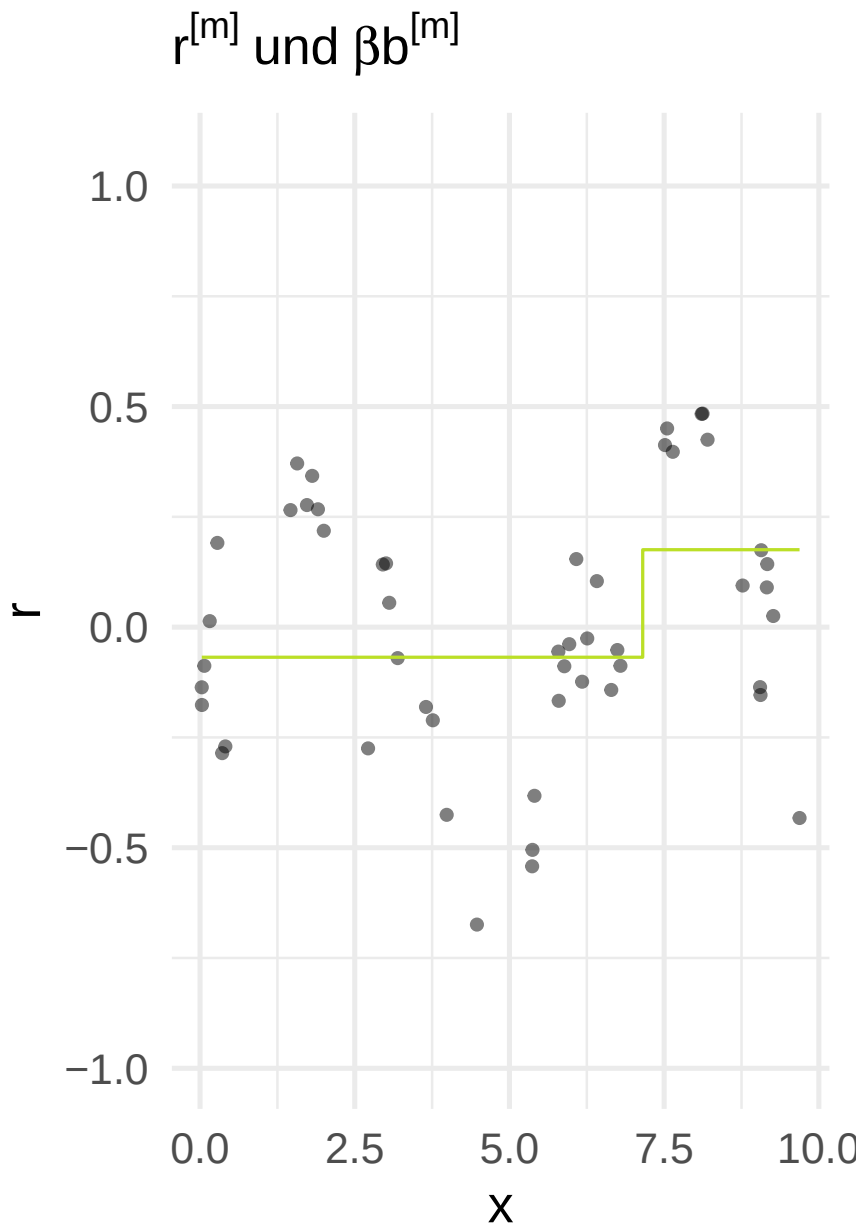
GRADIENT BOOSTING ILLUSTRATION - 2



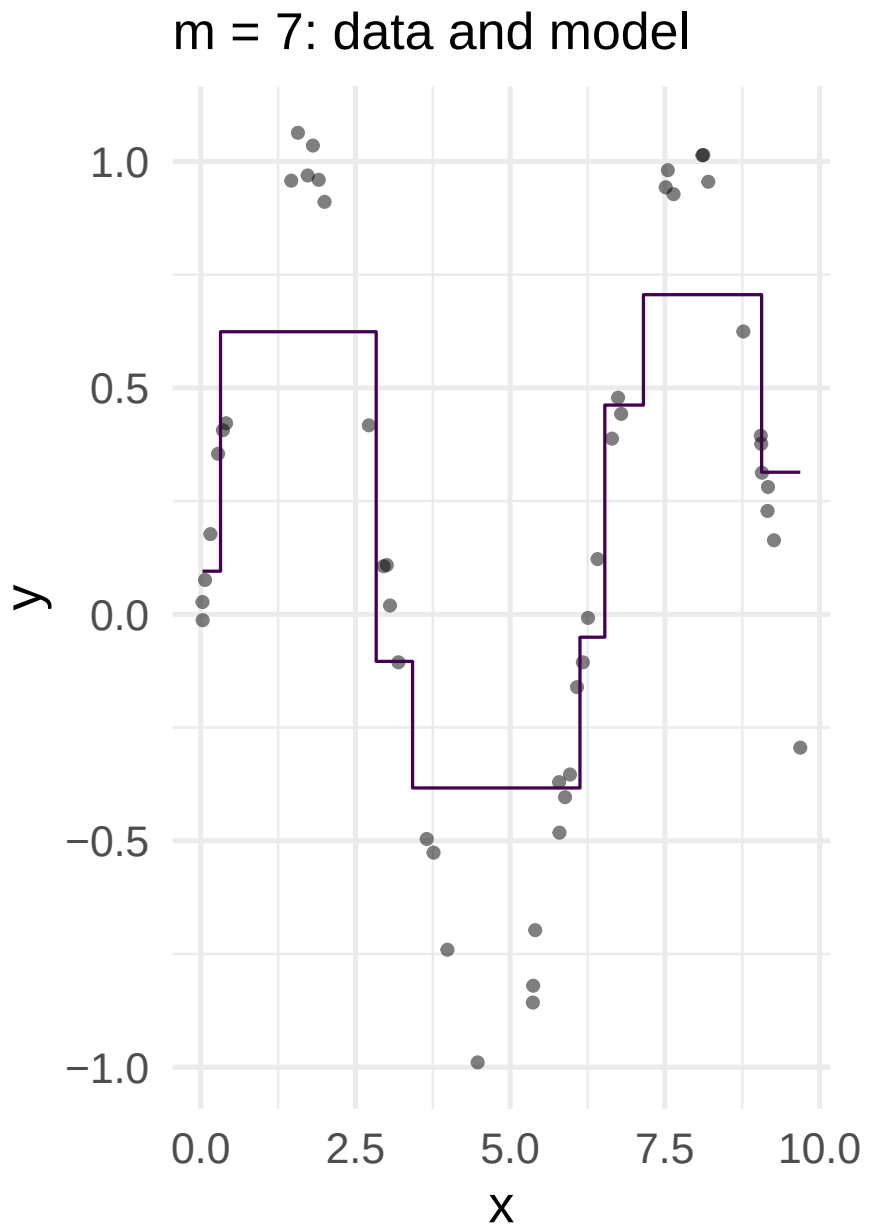


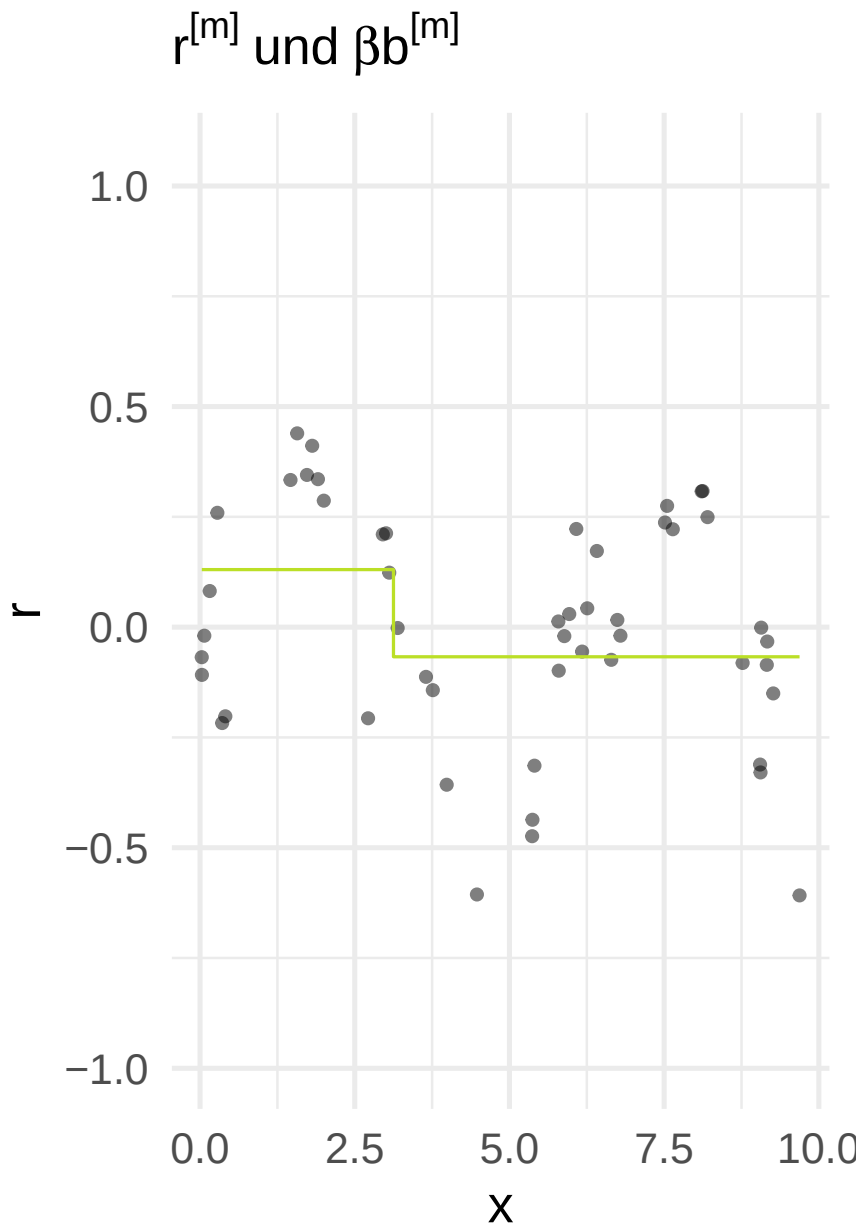
GRADIENT BOOSTING ILLUSTRATION - 2



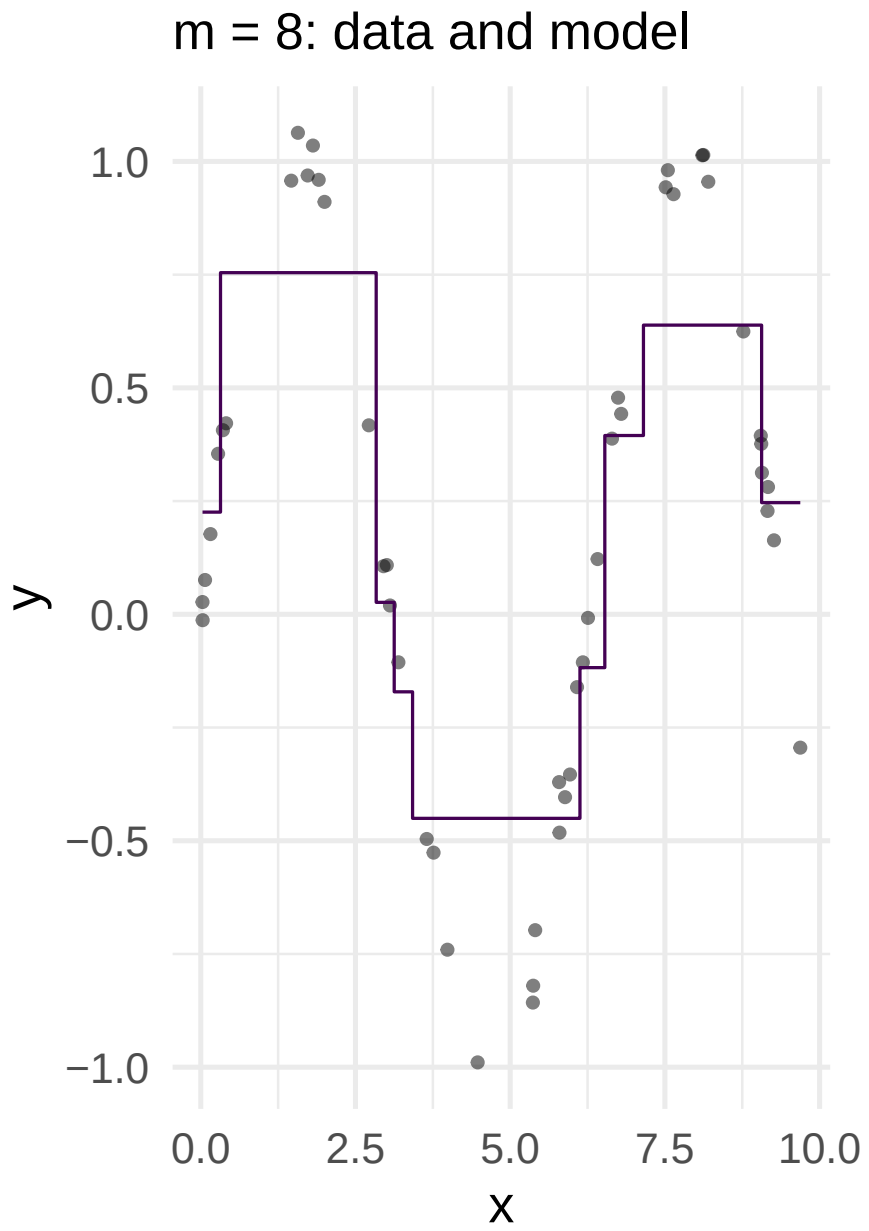


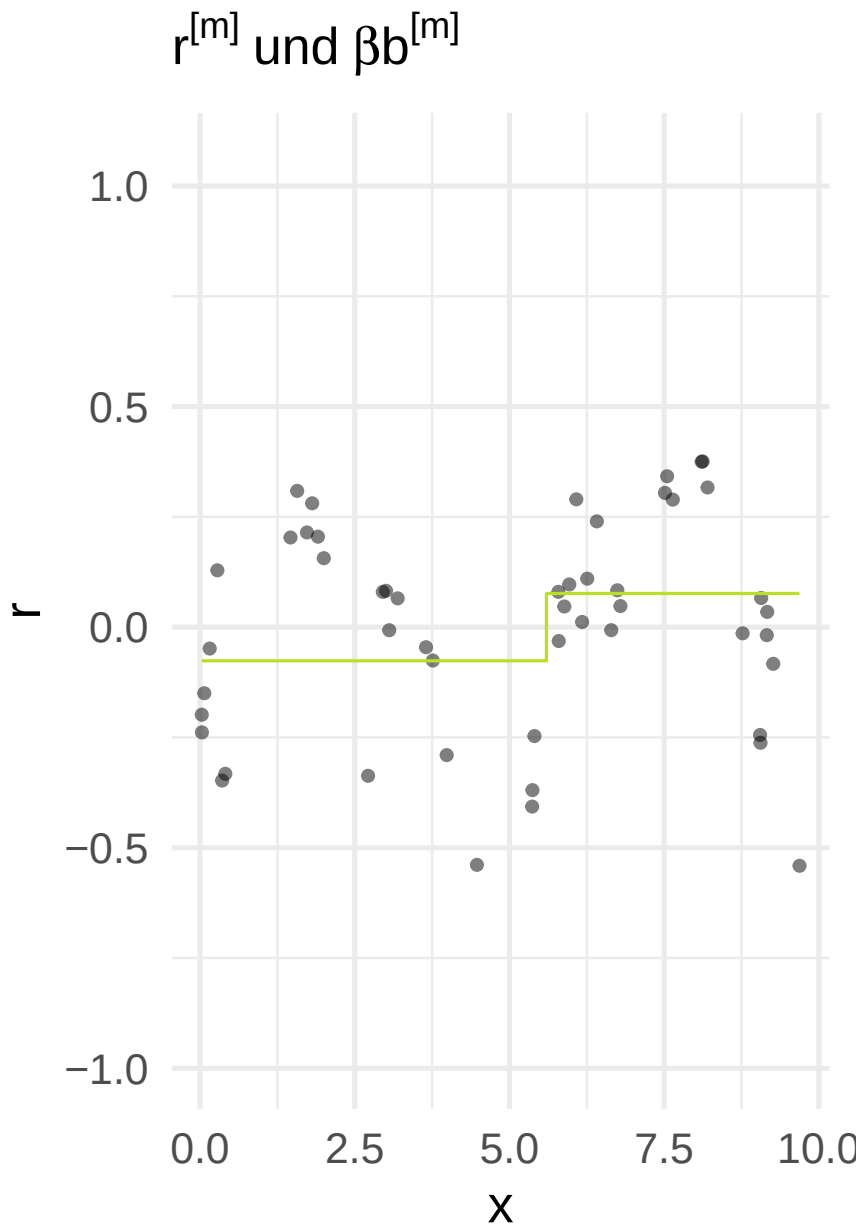
GRADIENT BOOSTING ILLUSTRATION - 2



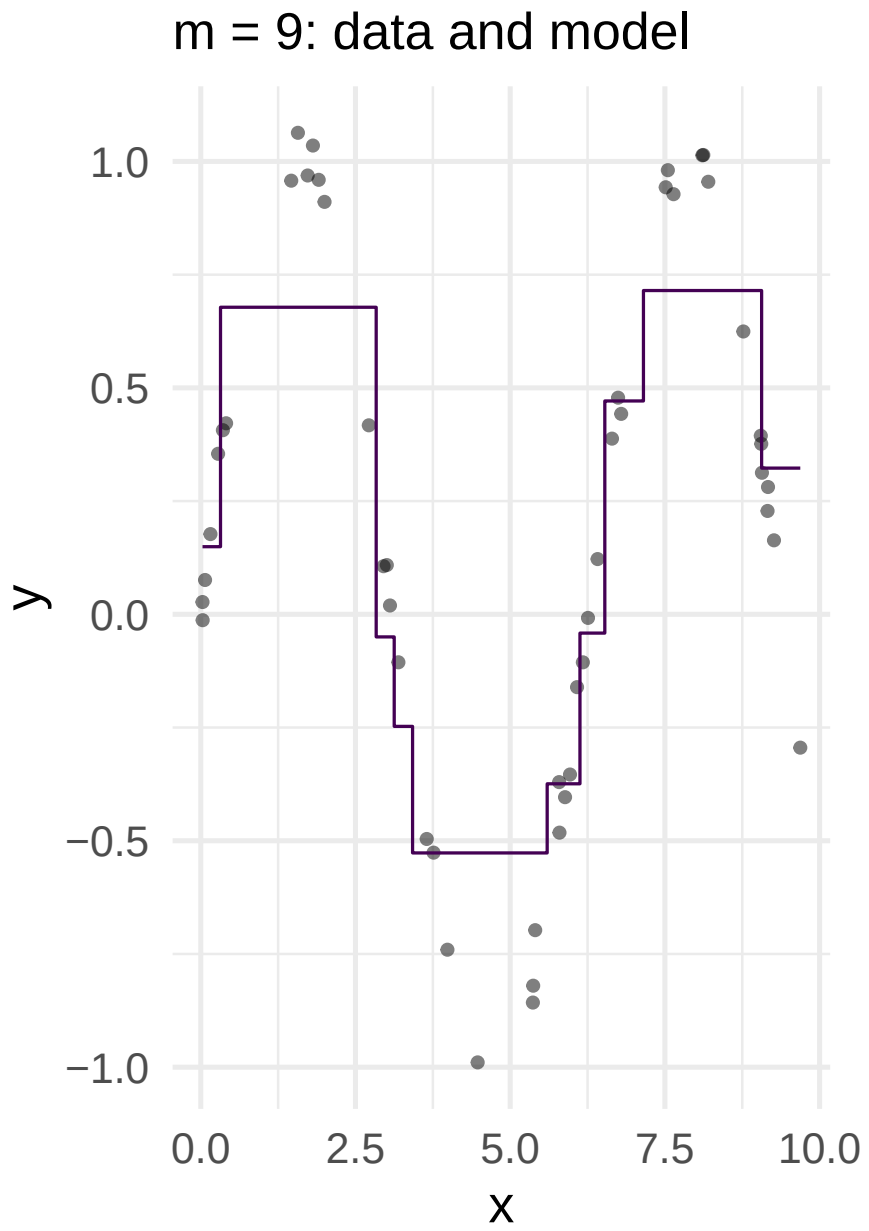


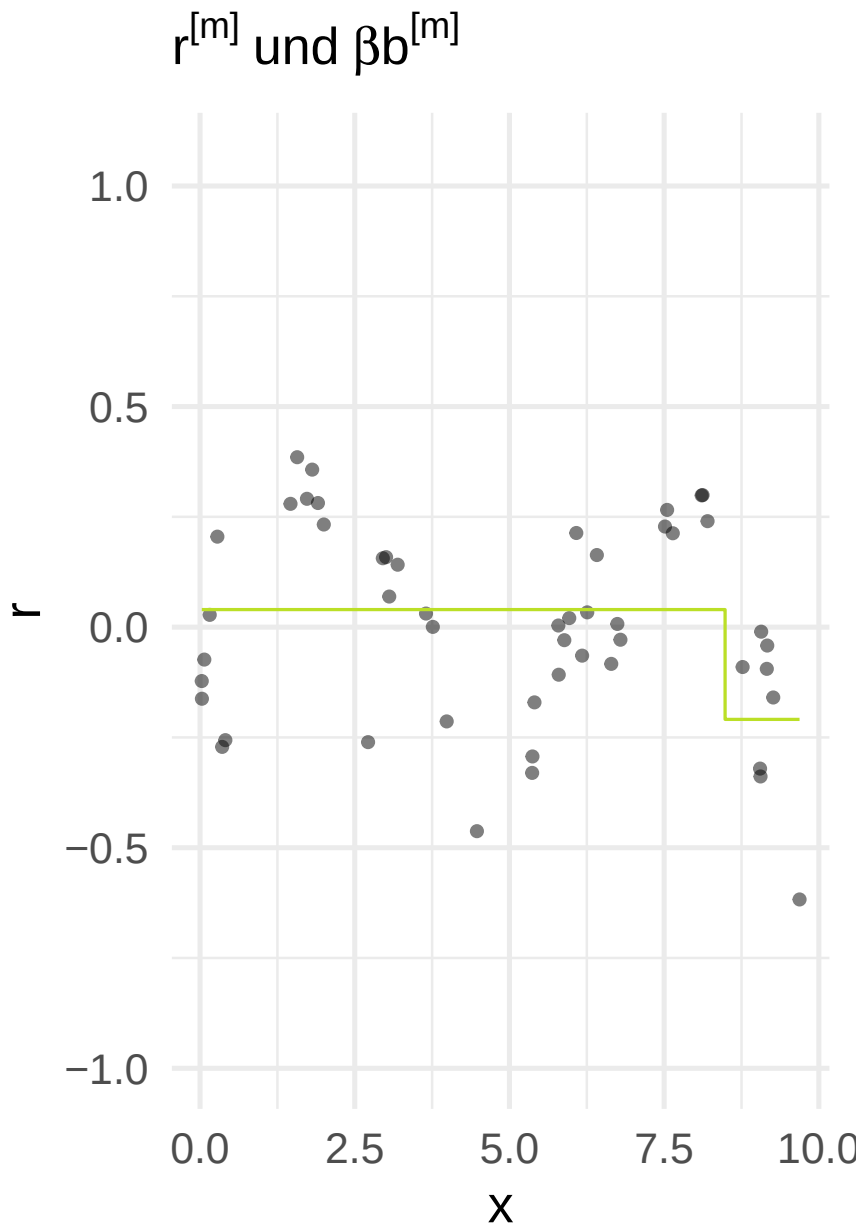
GRADIENT BOOSTING ILLUSTRATION - 2



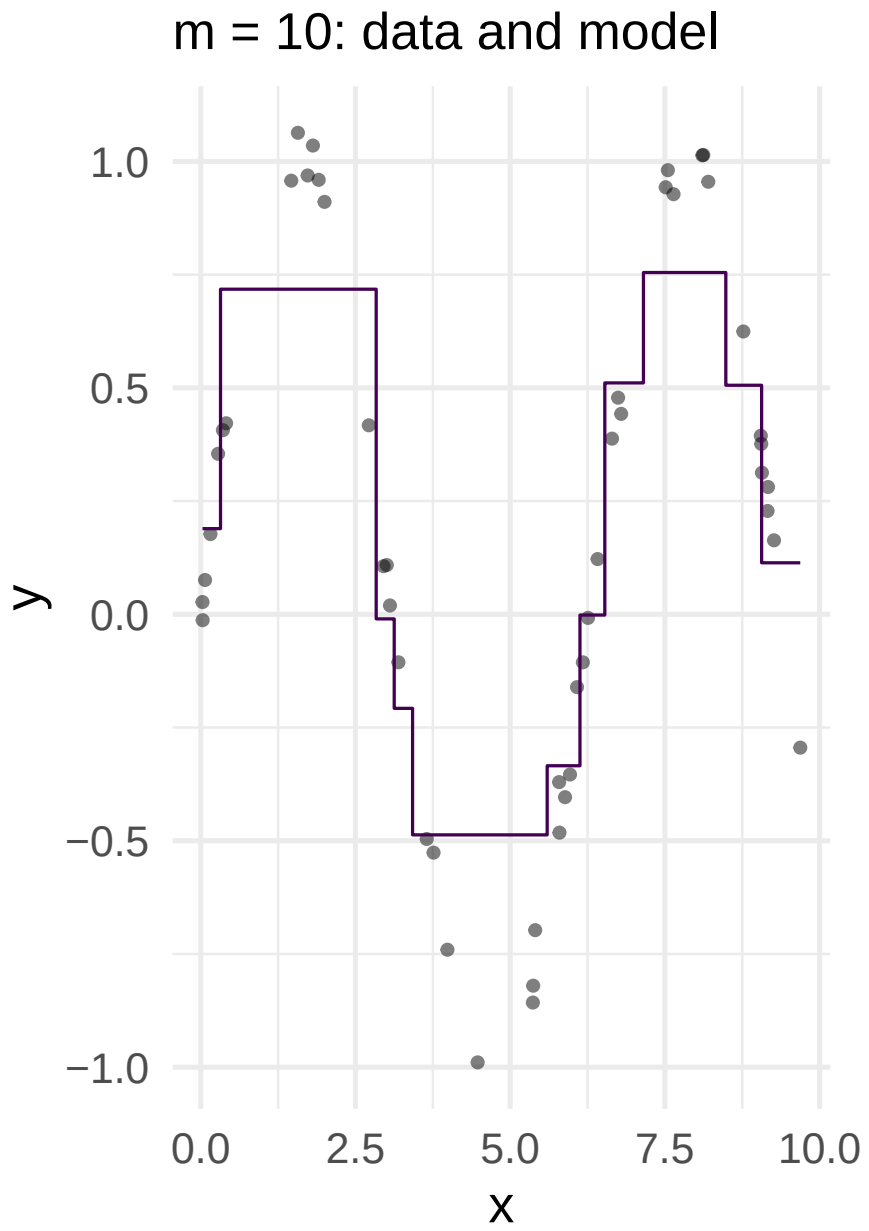


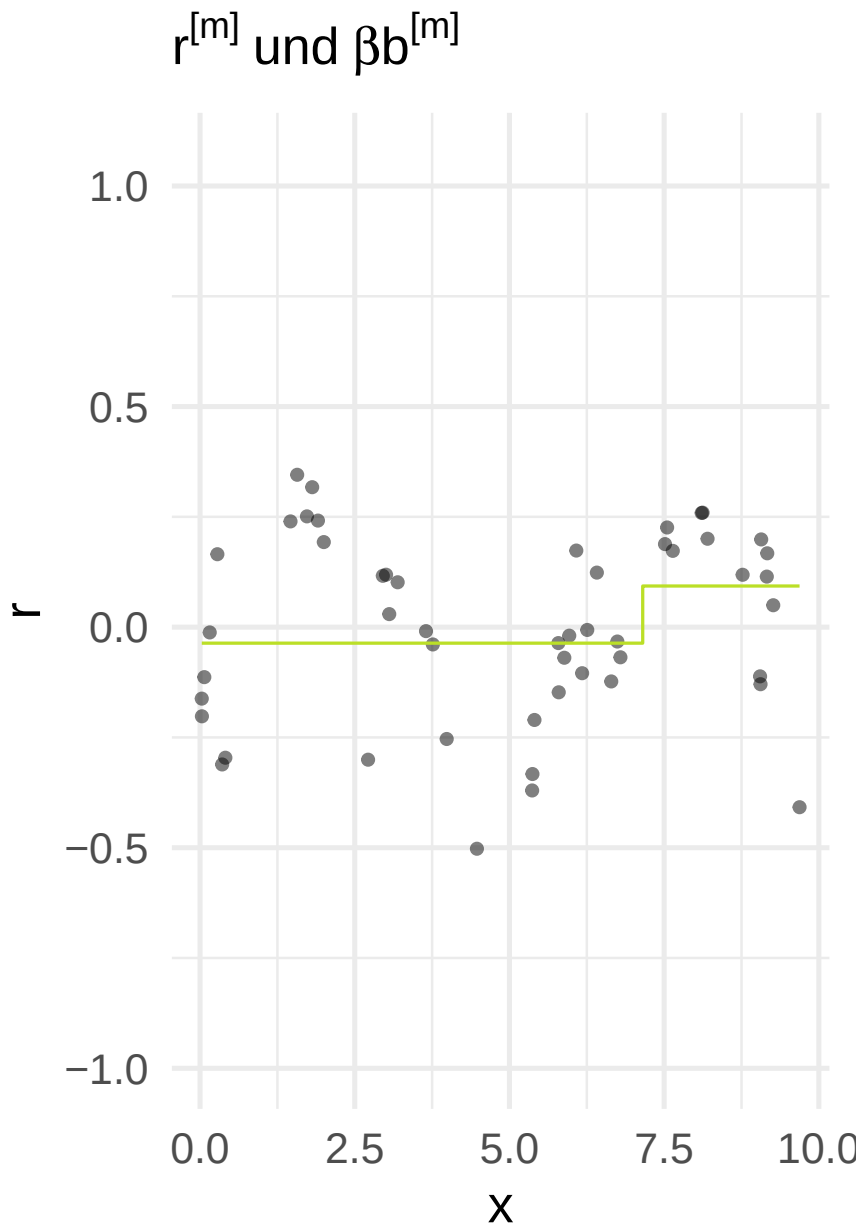
GRADIENT BOOSTING ILLUSTRATION - 2



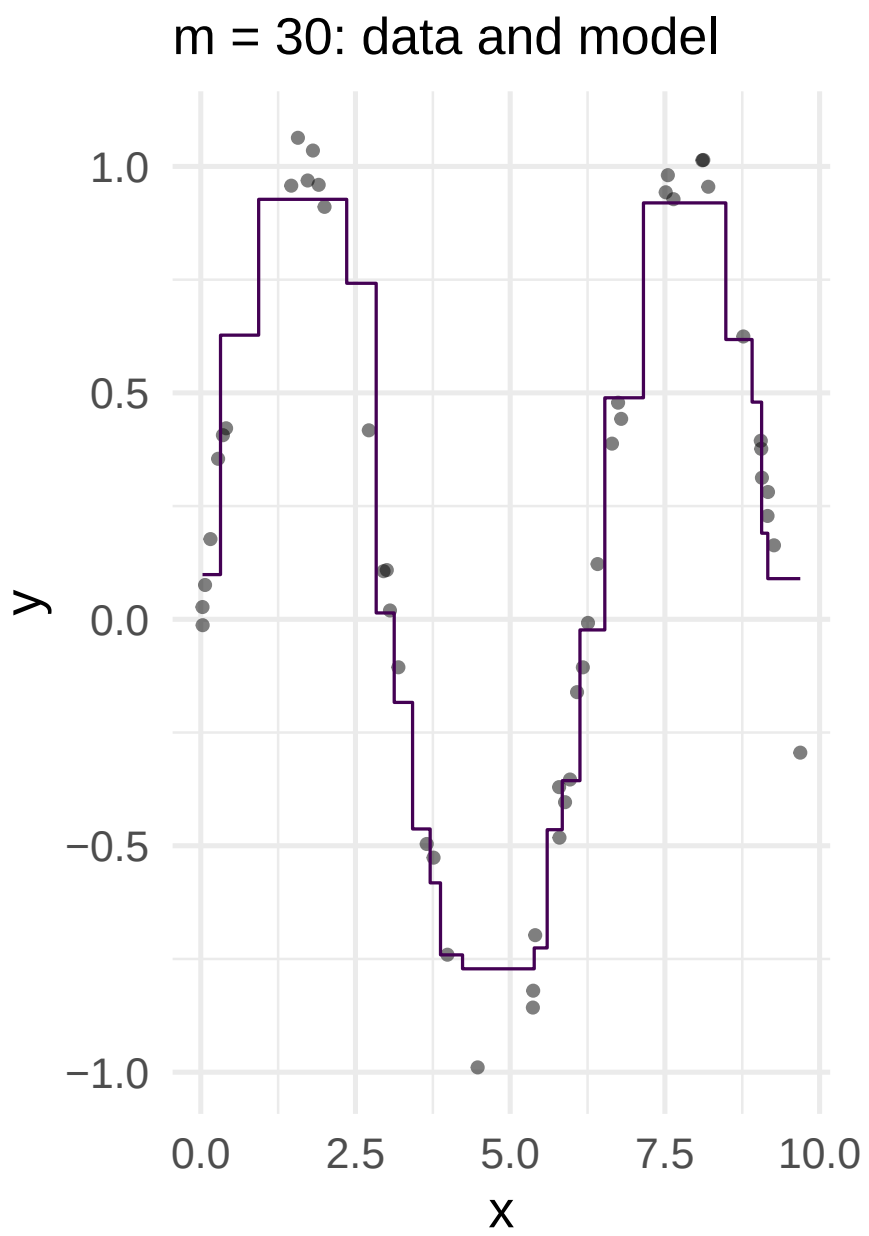


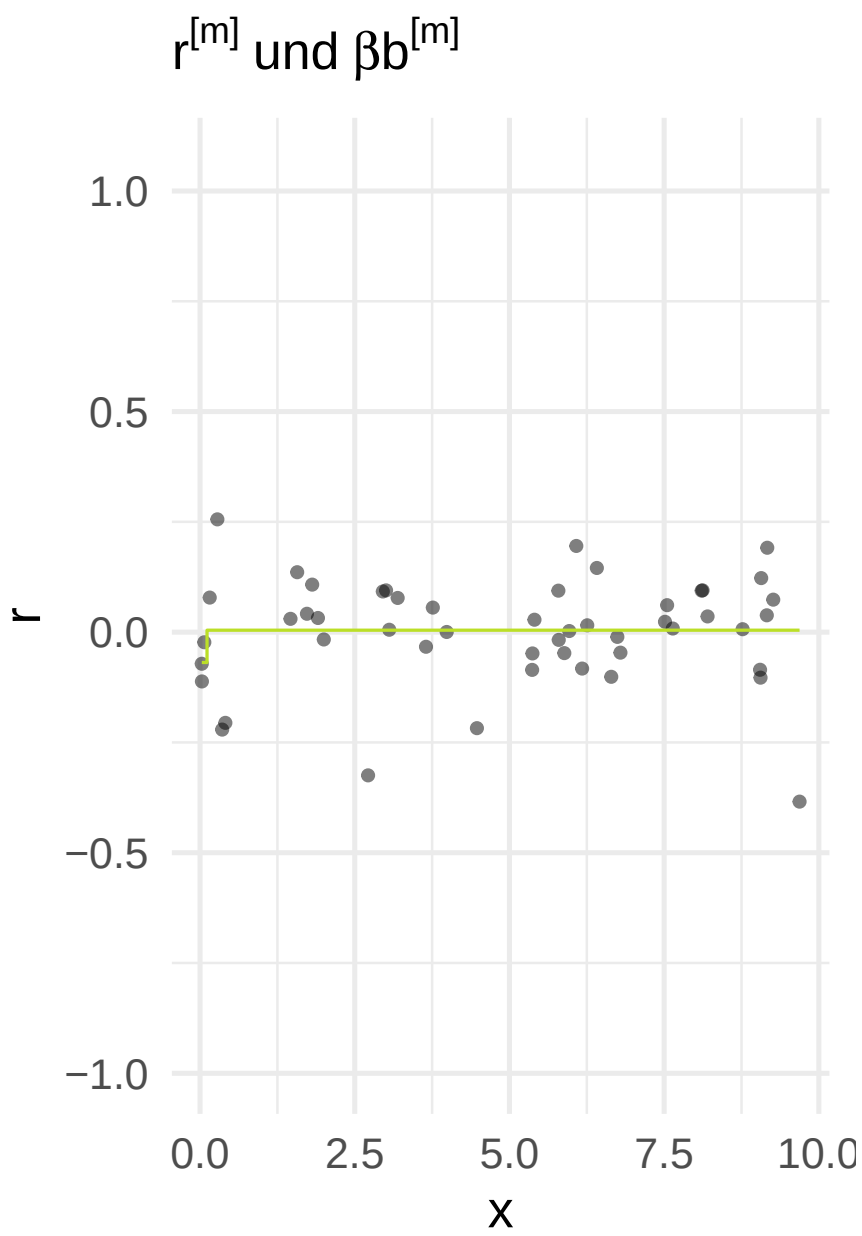
GRADIENT BOOSTING ILLUSTRATION - 2



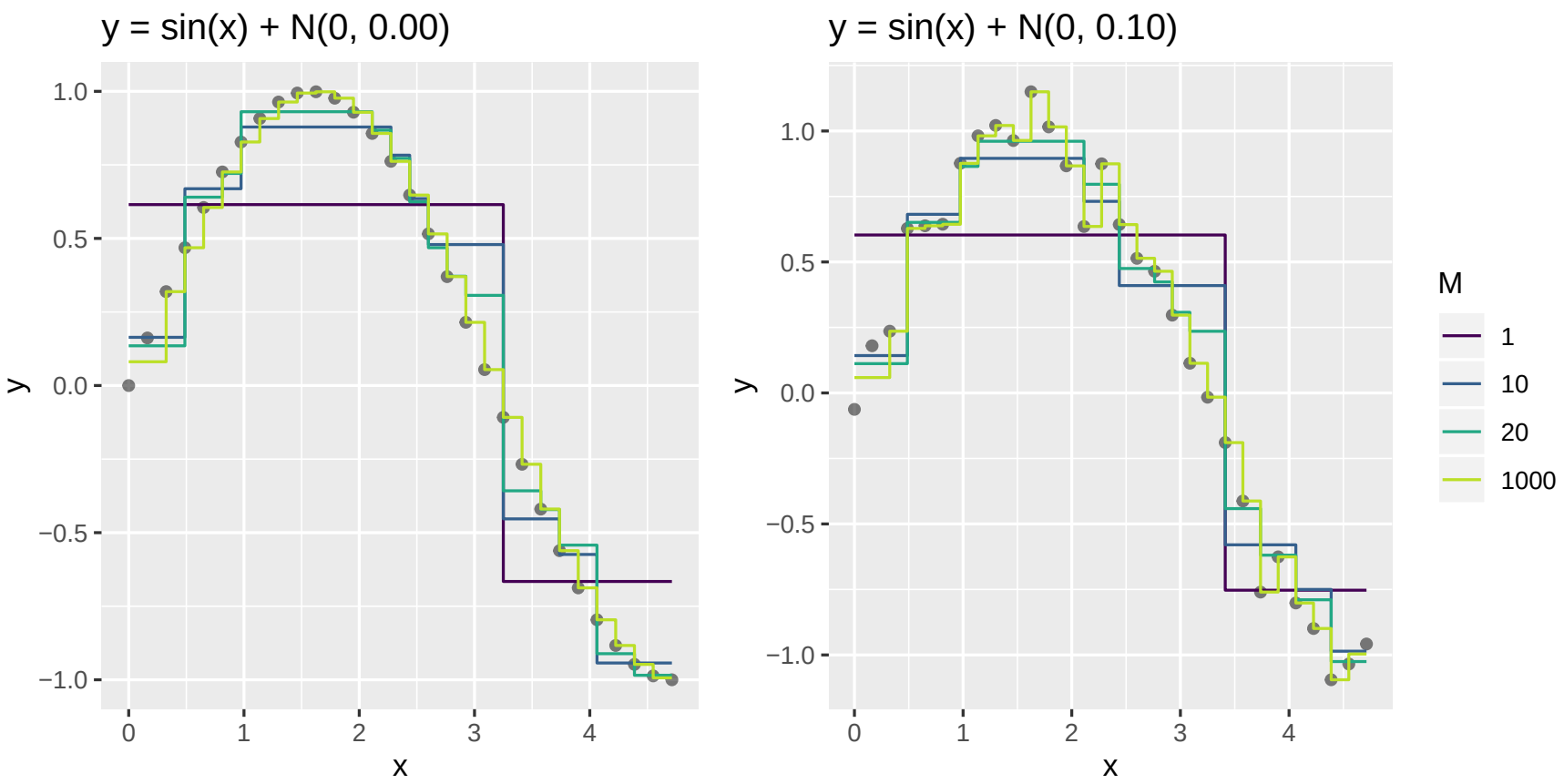


GRADIENT BOOSTING ILLUSTRATION - 2





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.