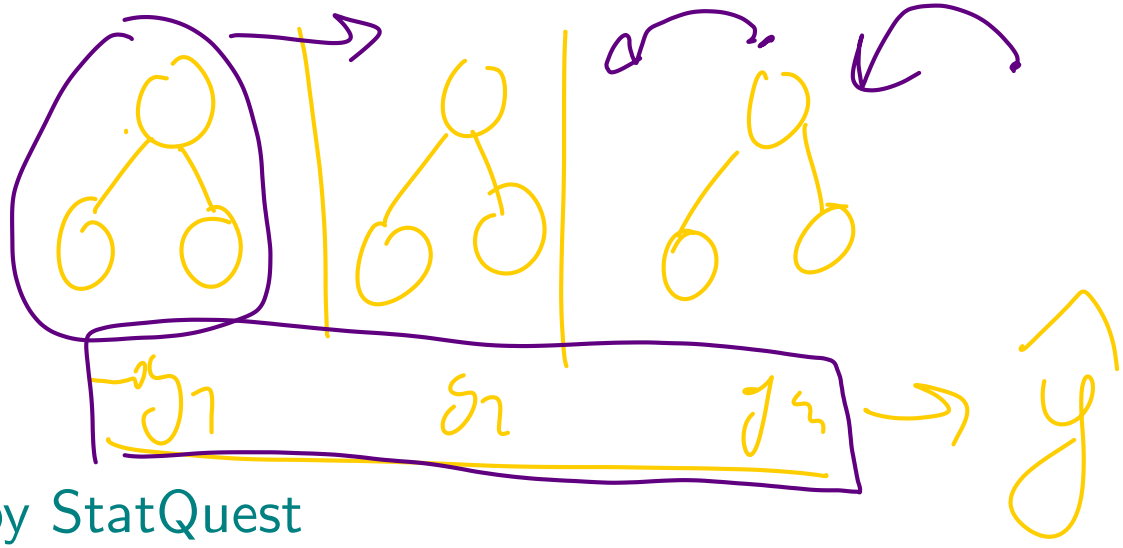
Resources

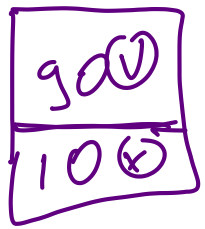


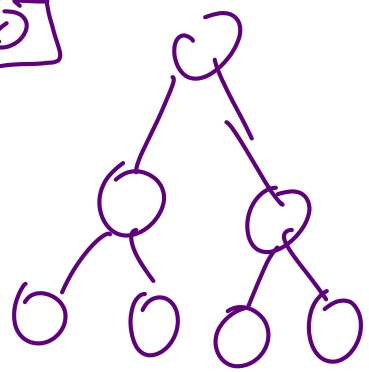
Great videos

- AdaBoost explained by StatQuest
- Gradient Boosting explained by StatQuest
- Chapter on Boosting by LMU SLDS

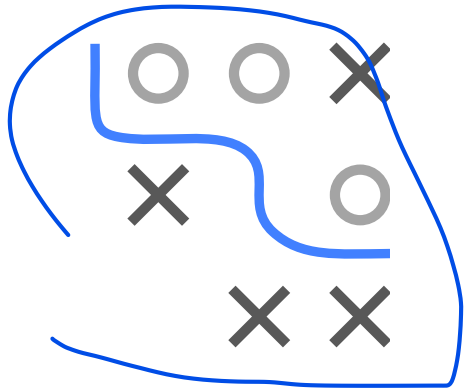
Great models

- LightGBM
- XGBoost
- CatBoost



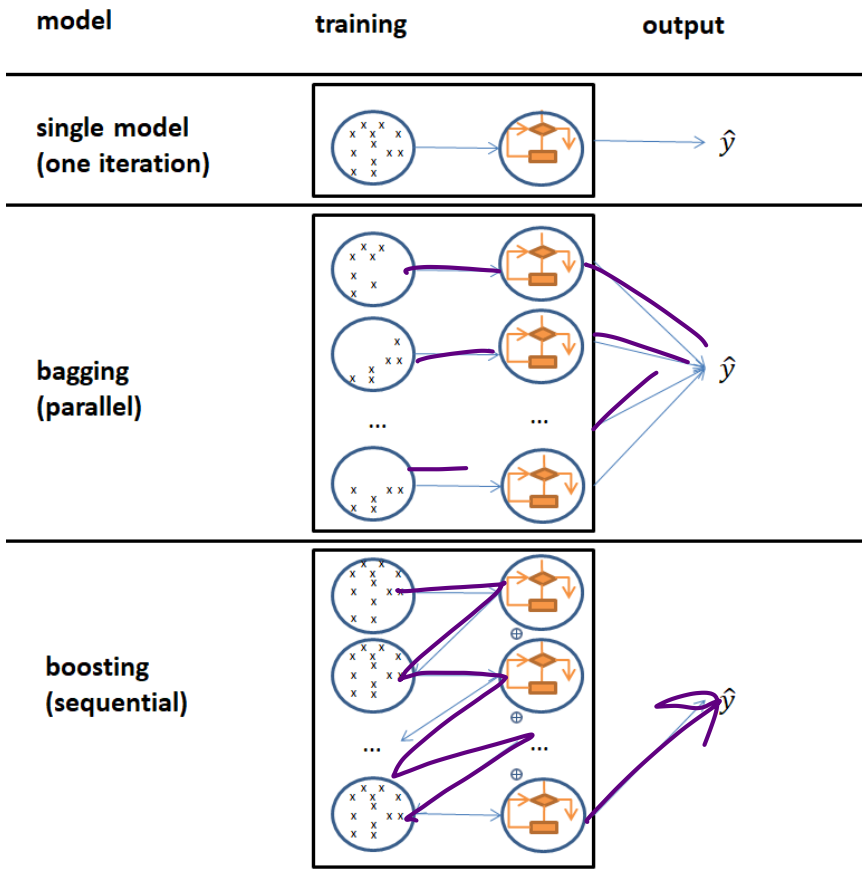


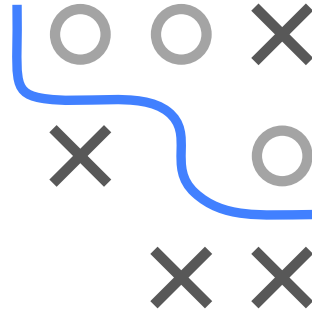
INTRODUCTION TO BOOSTING



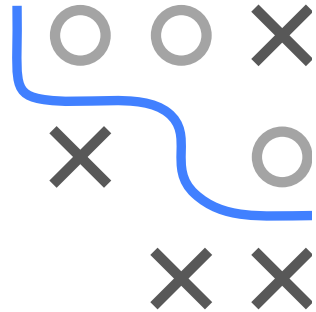
- Boosting is considered to be one of the most powerful learning ideas within the last twenty years.
- Originally designed for classification, (especially gradient) boosting handles regression (and many other supervised tasks) naturally nowadays.
- Homogeneous ensemble method (like bagging), but fundamentally 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





THE BOOSTING QUESTION



The first boosting algorithm ever was in fact no algorithm for practical purposes, but the solution for a theoretical problem:

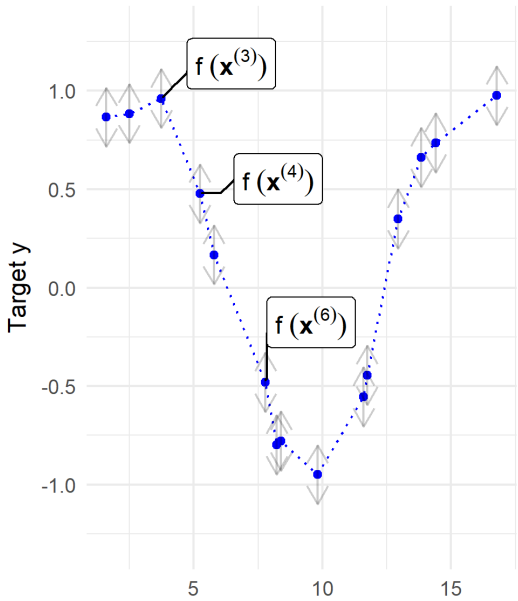
"Does the existence of a weak learner for a certain problem imply the existence of a strong learner?" ▸ Kearns, 1988

- **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.

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

Introduction to Machine Learning

Gradient Boosting: Concept

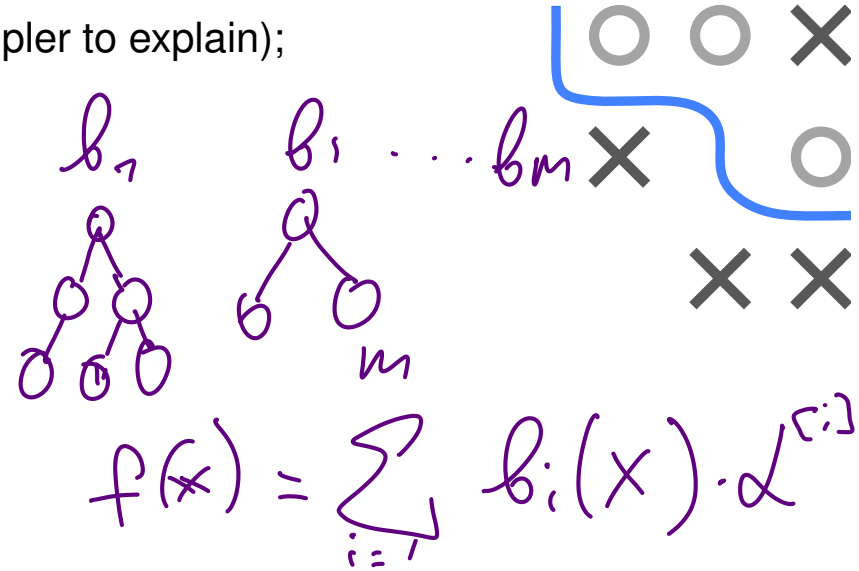


Learning goals

- 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:

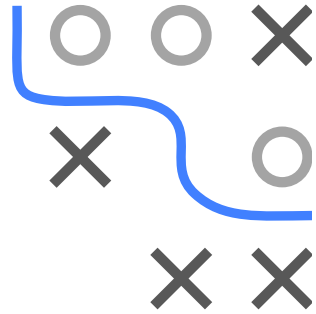
$$f(\mathbf{x}) = \sum_{m=1}^{M} \alpha^{[m]} b(\mathbf{x}, \boldsymbol{\theta}^{[m]}).$$

Hence, we minimize the empirical risk:

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

FORWARD STAGEWISE ADDITIVE MODELING

Why is gradient boosting a good choice for this problem?

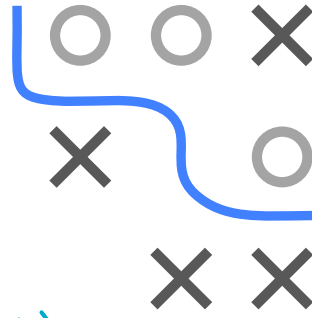


- Because of the additive structure it is difficult to jointly minimize $\mathcal{R}_{\text{emp}}(f)$ w.r.t. $\left(\left(\alpha^{[1]}, \boldsymbol{\theta}^{[1]}\right), \ldots, \left(\alpha^{[M]}, \boldsymbol{\theta}^{[M]}\right)\right)$, which is a very high-dimensional parameter space (though this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

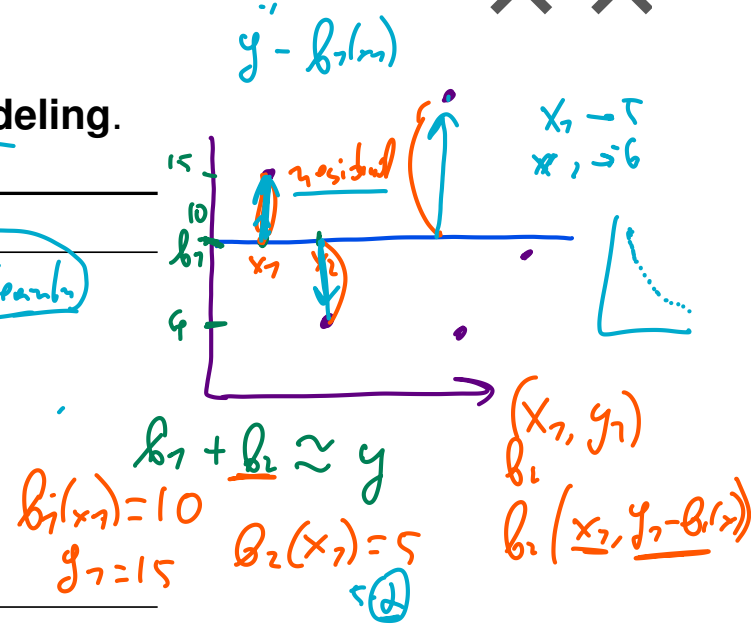
$$\min_{\alpha,\boldsymbol{\theta}} \sum_{i=1}^{n} L\left(y^{(i)}, \hat{f}^{[m-1]}\left(\mathbf{x}^{(i)}\right) + \alpha b\left(\mathbf{x}^{(i)}, \boldsymbol{\theta}\right)\right)$$



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

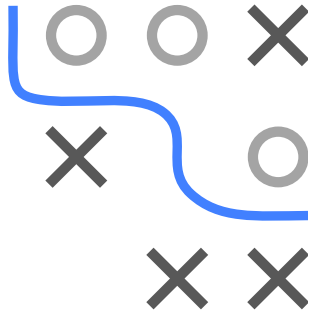
Algorithm Forward Stagewise Additive Modeling.

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



GRADIENT BOOSTING

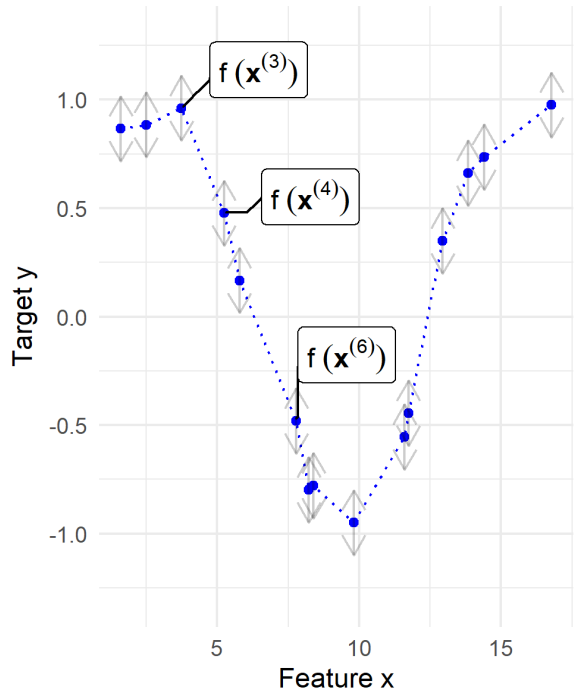
The algorithm we just introduced is not really an algorithm, but rather an abstract principle. We need to find the new additive component $b\left(\mathbf{x}, \boldsymbol{\theta}^{[m]}\right)$ and its weight coefficient $\alpha^{[m]}$ in each iteration m. This can be done by gradient descent, but in function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



Thought experiment: Consider a completely non-parametric model f whose predictions we can arbitrarily define on every point of the training data $\mathbf{x}^{(i)}$. So we basically specify f as a discrete, 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...).
Furthermore, we assume our loss function $L(\cdot)$ to be differentiable.

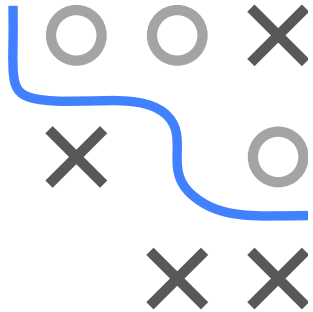


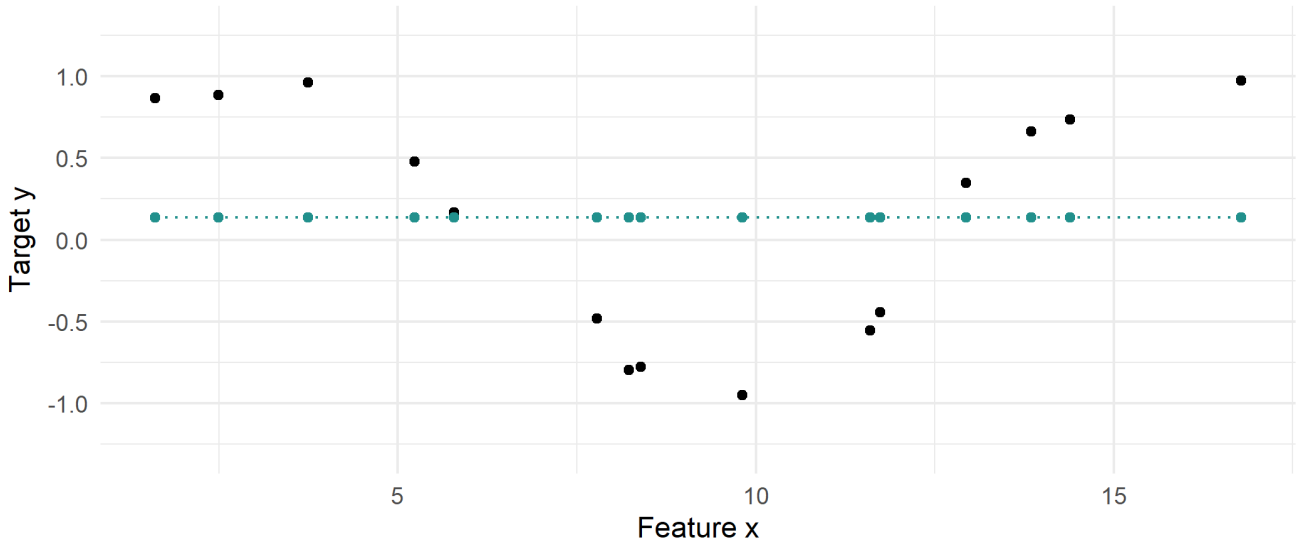
GRADIENT BOOSTING

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

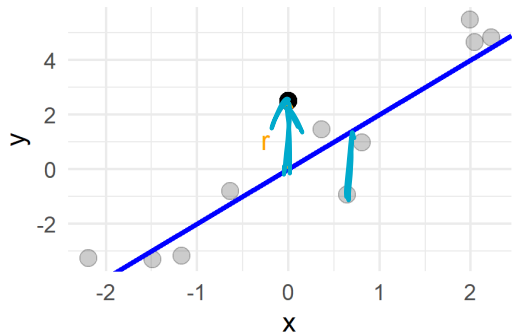
How do we have to distort this function to move it towards the observations and drive loss down?

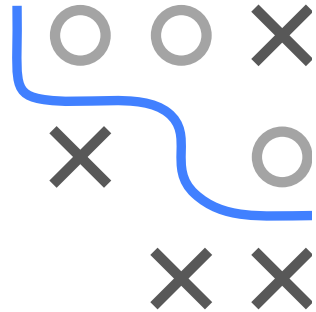
We minimize the risk of such a model with gradient descent (yes, this makes no sense, suspend all doubts for a few seconds).
So, we calculate the gradient at a point of the parameter space, that is, the derivative w.r.t. each component of the parameter vector (which is 0 for all terms with $i \neq j$):

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

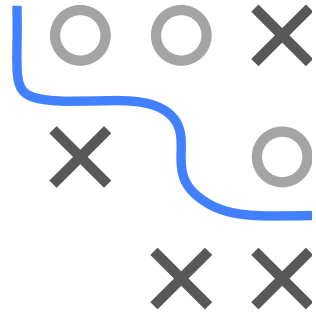
Reminder: The pseudo-residuals $\tilde{r}(f)$ match the usual residuals for the squared loss:

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





BOOSTING AS GRADIENT DESCENT



Combining this with the iterative additive procedure of "forward stagewise modeling", we are at the spot $f^{[m-1]}$ during minimization. At this point, we now calculate the direction of the negative gradient or also called pseudo-residuals $\tilde{r}^{[m](i)}$:

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

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

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

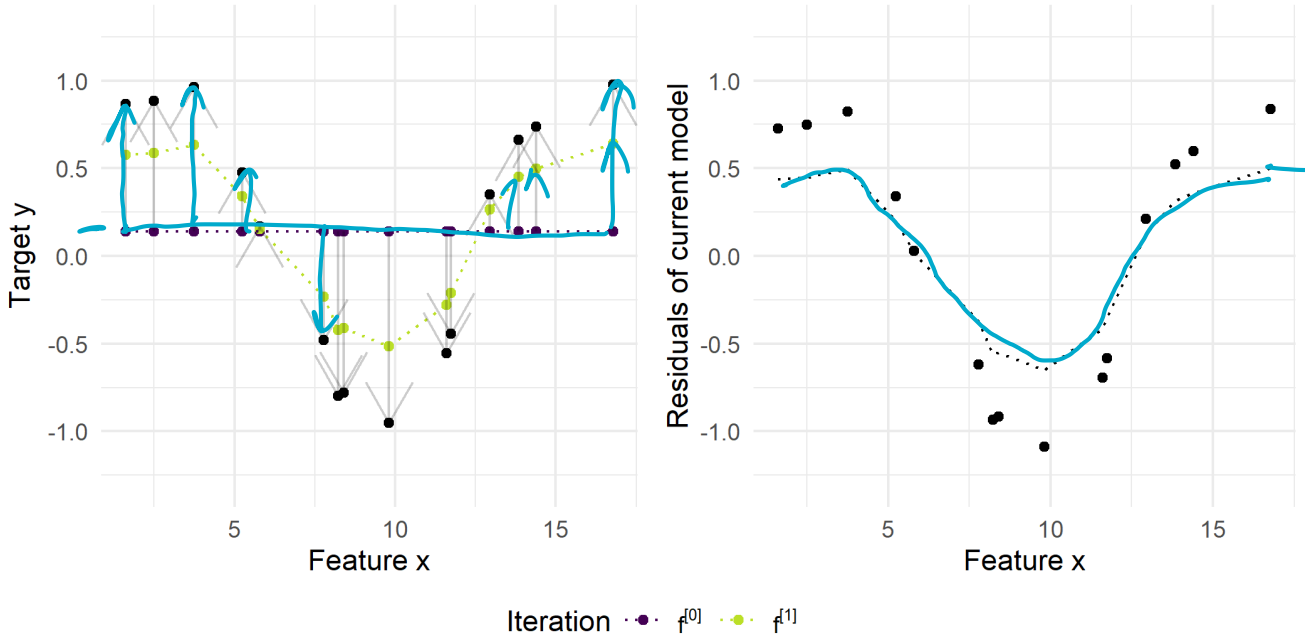
This tells us how we could "nudge" our whole function f in the direction of the data to reduce its empirical risk.

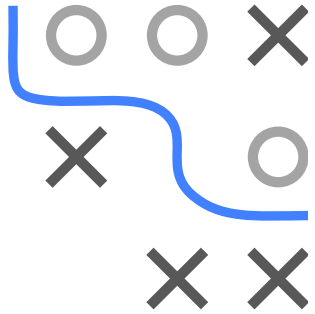
GRADIENT BOOSTING

Iteration 1:

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

$\alpha = 0.6$





$f_1 = f_0 + \alpha b_1 \ldots$

residual $= y - f$

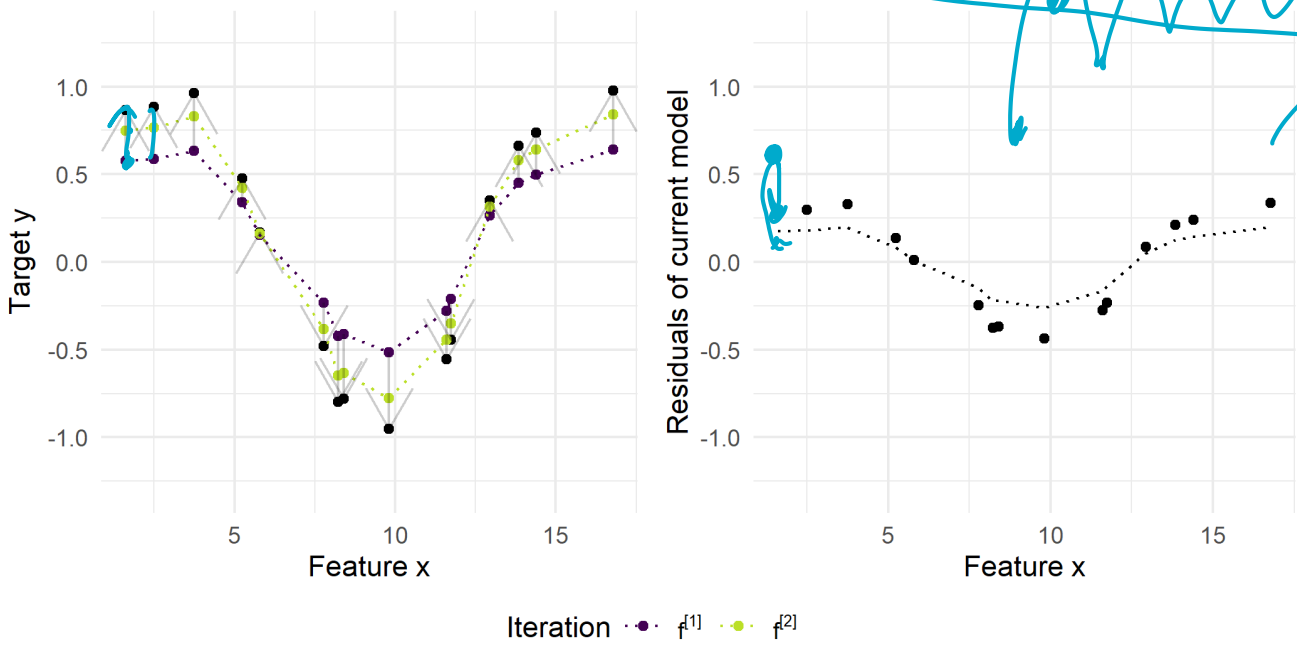
no obs value

$f_1 =$

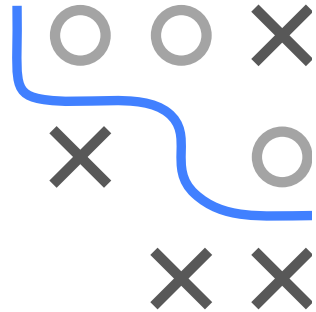
GRADIENT BOOSTING

Iteration 2:

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



GRADIENT BOOSTING

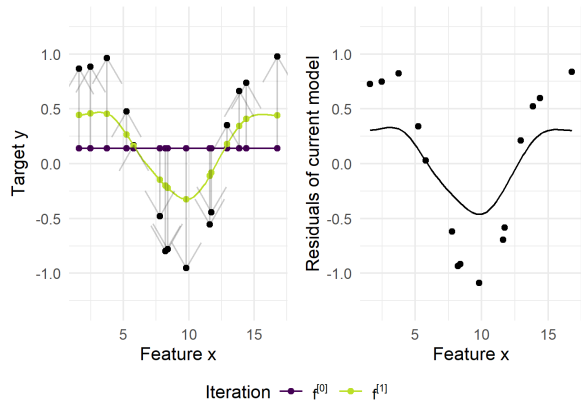


To parameterize a model in this way is pointless, as it just memorizes the instances of the training data.

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:

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

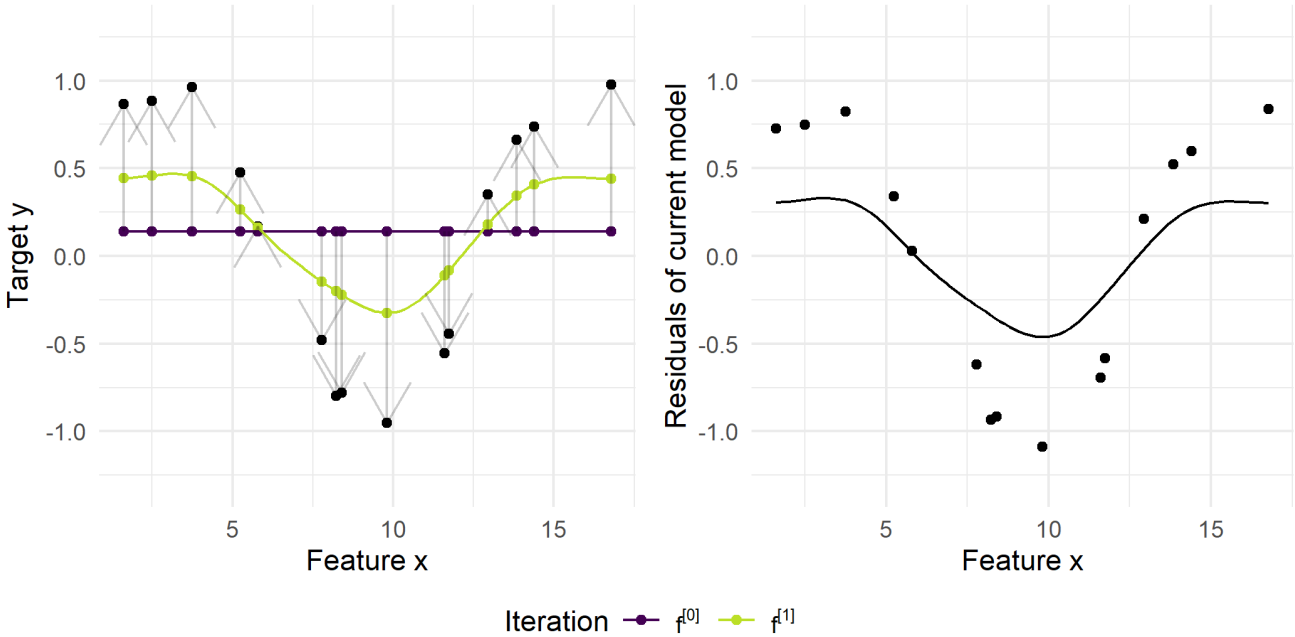
So, evaluated on the training data, our $b(\mathbf{x}, \boldsymbol{\theta}^{[m]})$ corresponds as closely as possible to the negative loss function gradient and generalizes over the whole space.



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.

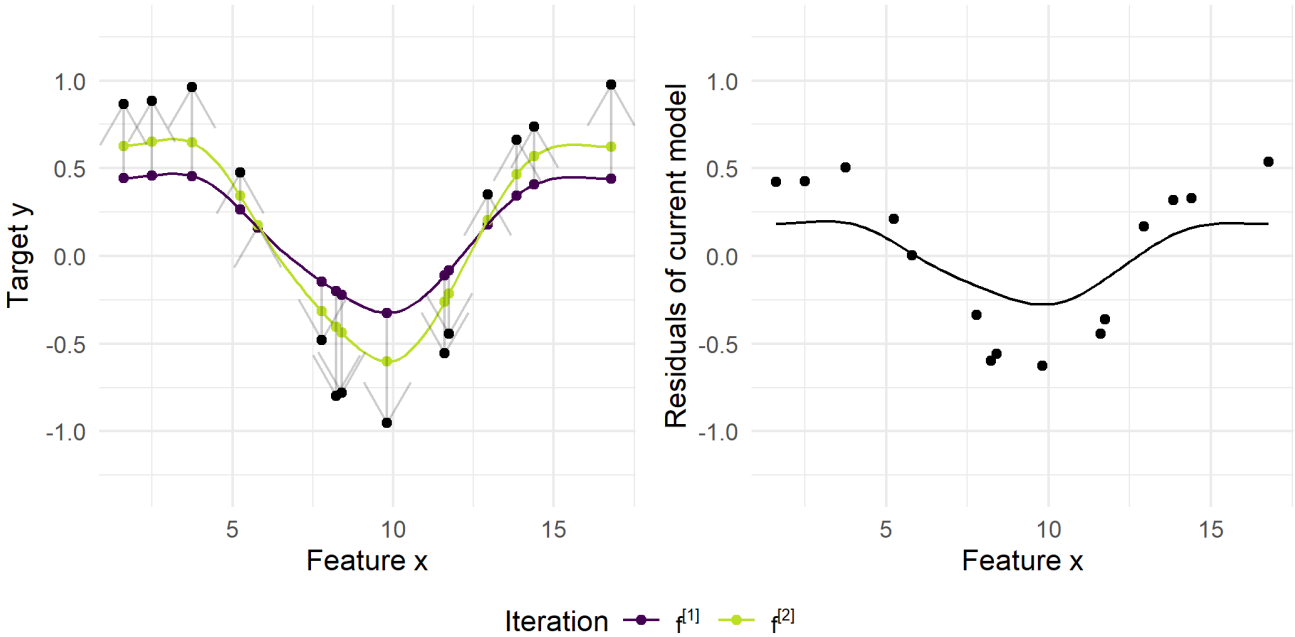
Iteration 1:



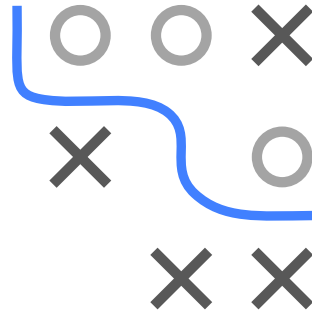
GRADIENT BOOSTING

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:

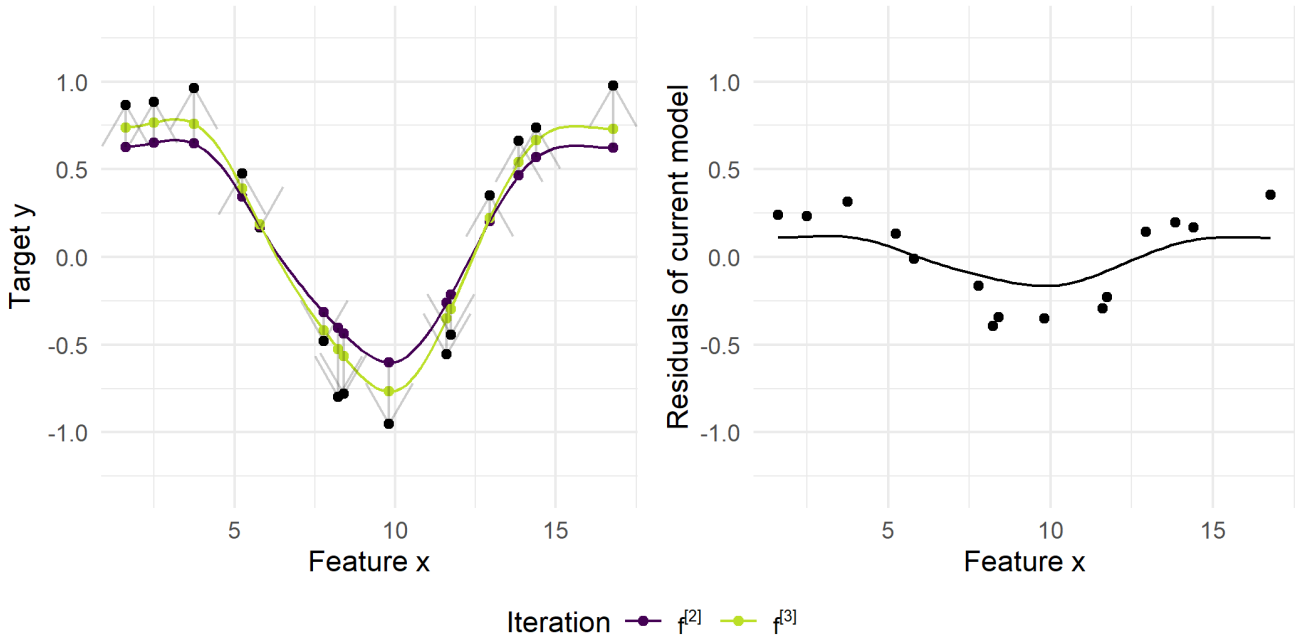


GRADIENT BOOSTING



This base learner is then added to the current state of the ensemble weighted by the learning rate (here: $\alpha = 0.4$) and for the next iteration again the pseudo-residuals of the adapted ensemble are calculated and a base learner is fitted to them.

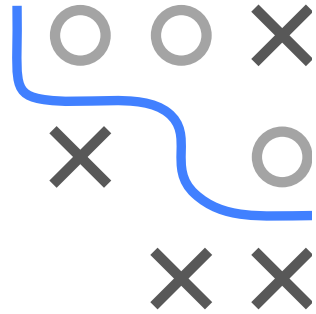
Iteration 3:



GRADIENT BOOSTING ALGORITHM

Algorithm Gradient Boosting Algorithm.

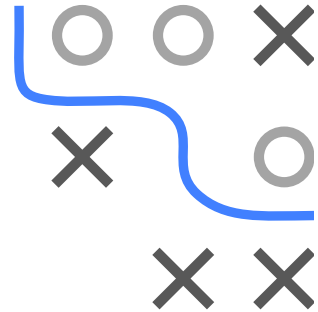
1: Initialize $\hat{f}^{[0]}(\mathbf{x}) = \arg\min_{\theta_0 \in \mathbb{R}} \sum_{i=1}^{n} L(y^{(i)}, \theta_0)$
2: **for** $m = 1 \to M$ **do**
3: For all i: $\tilde{r}^{[m](i)} = -\left[\frac{\partial L(y,f)}{\partial f}\right]_{f=\hat{f}^{[m-1]}(\mathbf{x}^{(i)}), y=y^{(i)}}$
4: Fit a regression base learner to the vector of pseudo-residuals $\tilde{r}^{[m]}$:
5: $\hat{\theta}^{[m]} = \arg\min_{\theta} \sum_{i=1}^{n} (\tilde{r}^{[m](i)} - b(\mathbf{x}^{(i)}, \theta))^2$
6: Set $\alpha^{[m]}$ to α being a small constant value or via line search
7: Update $\hat{f}^{[m]}(\mathbf{x}) = \hat{f}^{[m-1]}(\mathbf{x}) + \alpha^{[m]} b(\mathbf{x}, \hat{\theta}^{[m]})$
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.

LINE SEARCH

The learning rate in gradient boosting influences how fast the algorithm converges. Although a small constant learning rate is commonly used in practice, it can also be replaced by a line search.



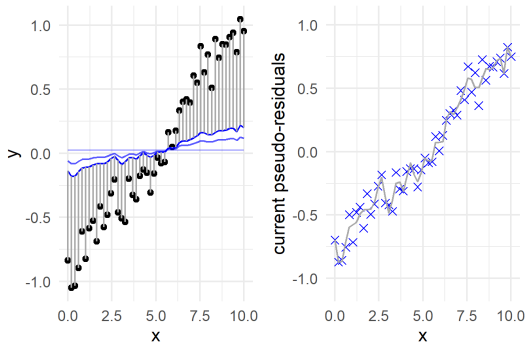
Line search is an iterative approach to find a local minimum. In the case of setting the learning rate, the following one-dimensional optimization problem has to be solved:

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

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

Introduction to Machine Learning

Gradient Boosting: Illustration

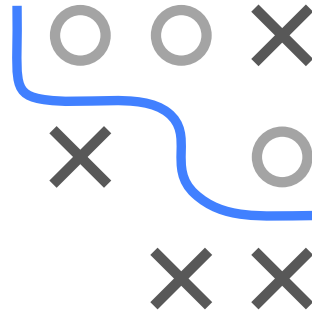


Learning goals

- See simple visualizations of boosting in regression
- Understand impact of different losses and base learners

GRADIENT BOOSTING ILLUSTRATION - GAM

GAM / Splines as BL and compare $L2$ vs. $L1$ loss.

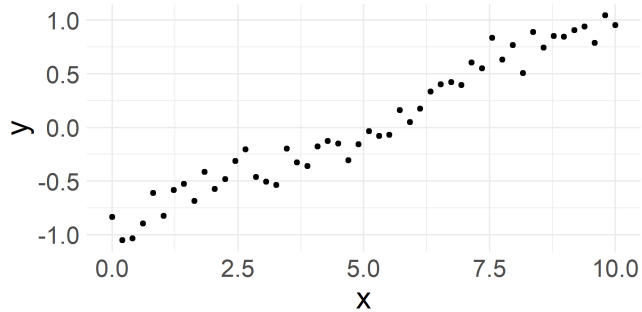


- L2: Init = optimal constant = mean(y); for L1 it's median(y)
- BLs are cubic B-splines with 40 knots.
- PRs $L2$: $\tilde{r}(f) = r(f) = y - f(\mathbf{x})$
- PRs $L1$: $\tilde{r}(f) = sign(y - f(\mathbf{x}))$
- Constant learning rate 0.2

Univariate toy data:

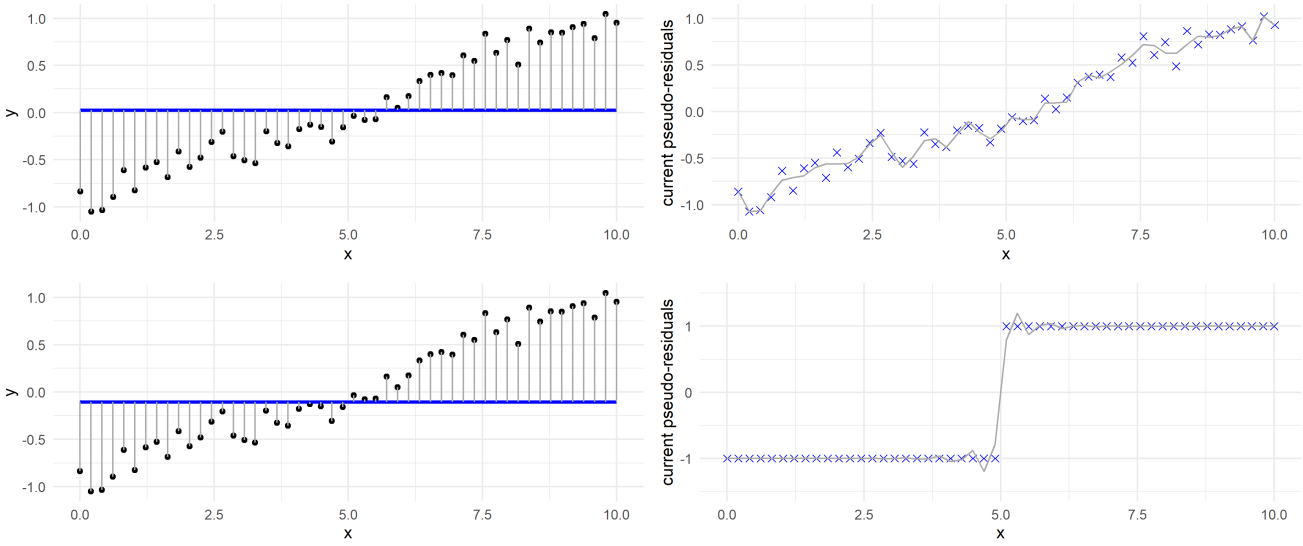
$$y^{(i)} = -1 + 0.2 \cdot x^{(i)} + 0.1 \cdot sin(x^{(i)}) + \epsilon^{(i)}$$

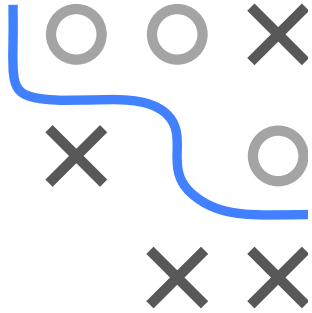
$$n = 50 \; ; \epsilon^{(i)} \sim \mathcal{N}(0, 0.1)$$



GAM WITH *L2* VS *L1* LOSS

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



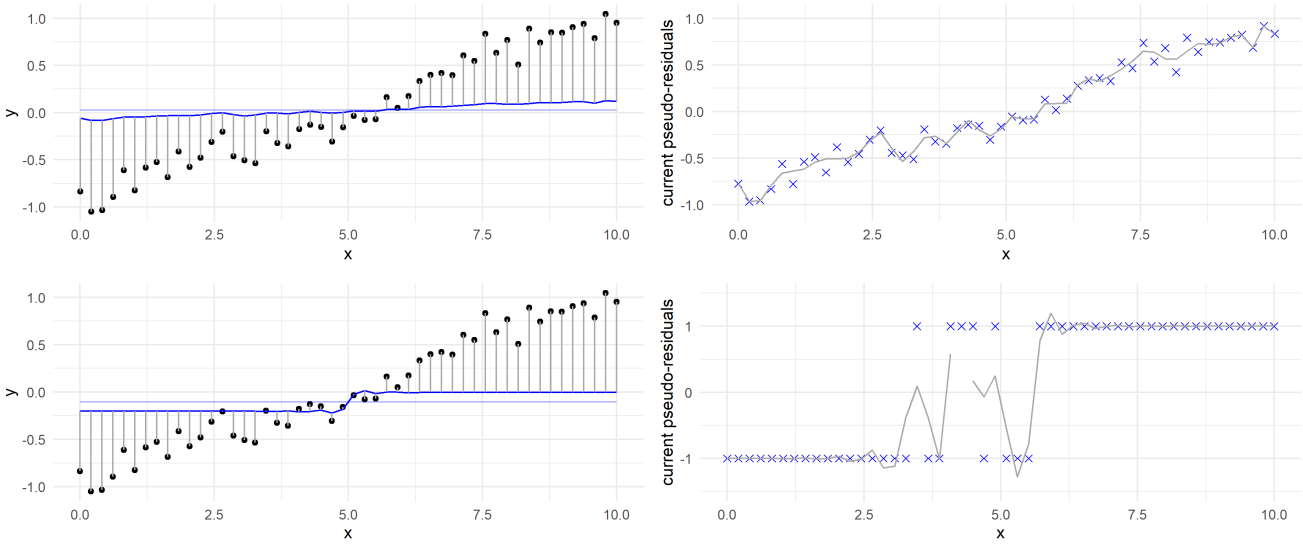


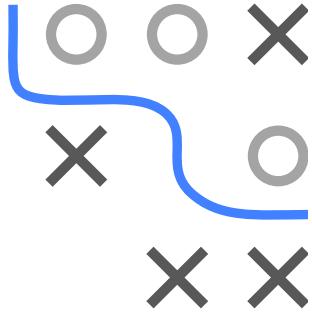
Iteration 1

Shape of PRs affects gradual model fit: *L1* only sees resids' sign, BLs are not affected size of values as in *L2* and hence lead to more moderate changes.

GAM WITH *L2* VS *L1* LOSS

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



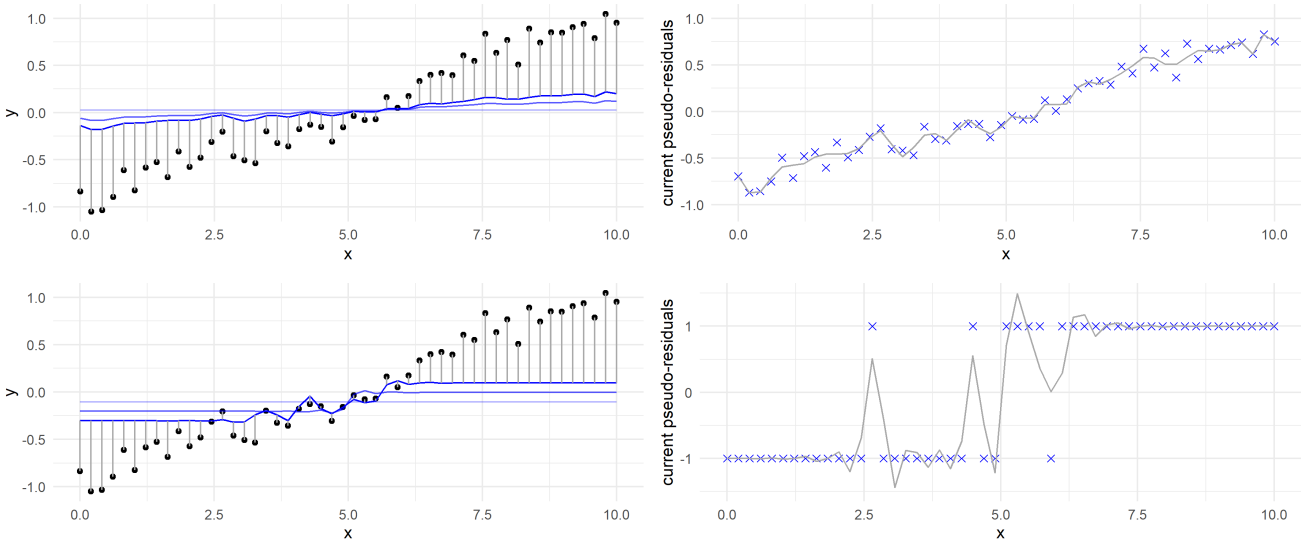


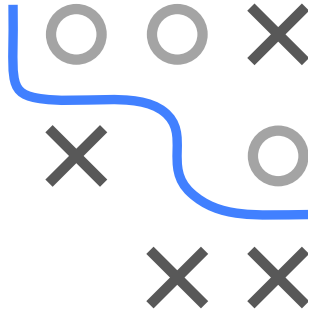
Iteration 2

Shape of PRs affects gradual model fit: *L1* only sees resids' sign, BLs are not affected size of values as in *L2* and hence lead to more moderate changes.

GAM WITH *L2* VS *L1* LOSS

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



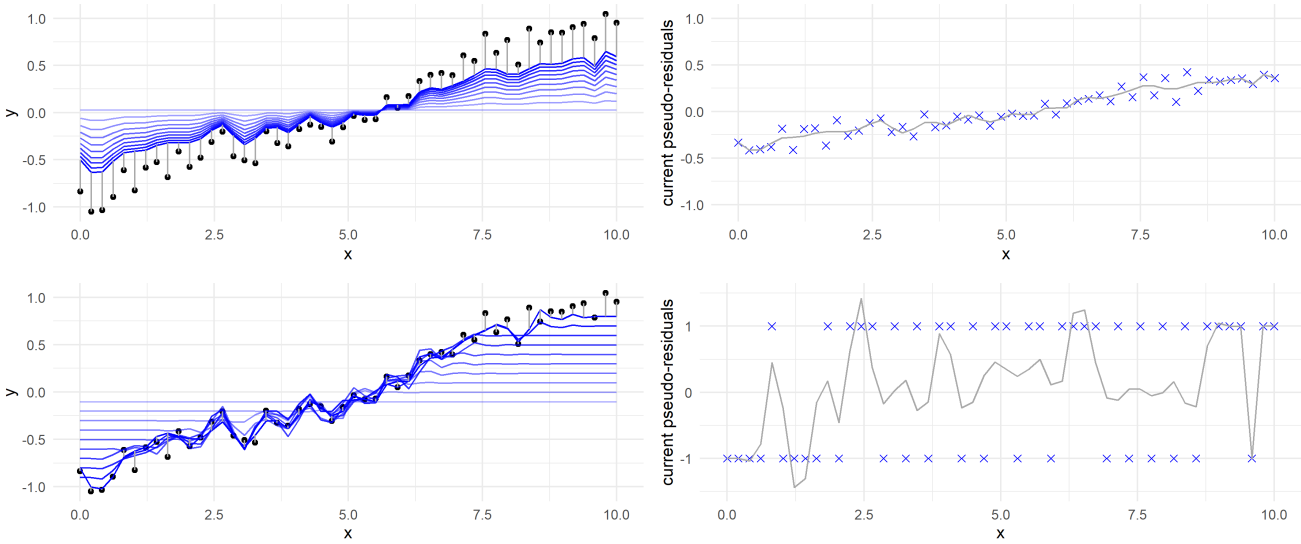


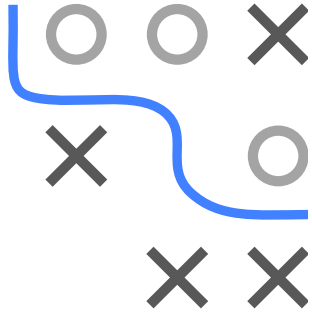
Iteration 3

Shape of PRs affects gradual model fit: *L1* only sees resids' sign, BLs are not affected size of values as in *L2* and hence lead to more moderate changes.

GAM WITH *L2* VS *L1* LOSS

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



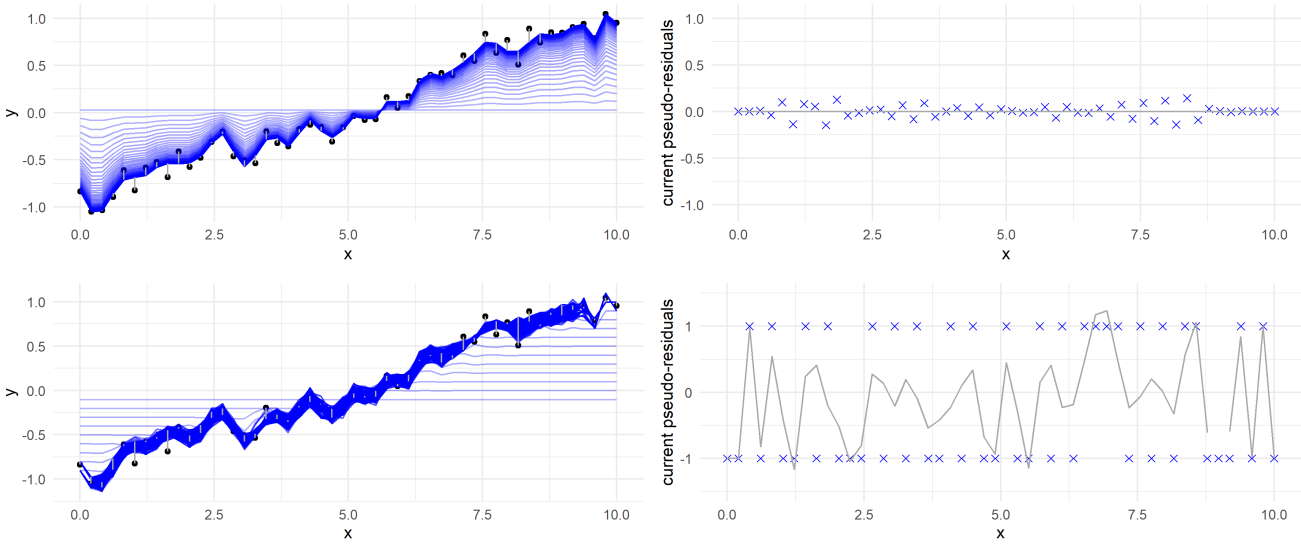


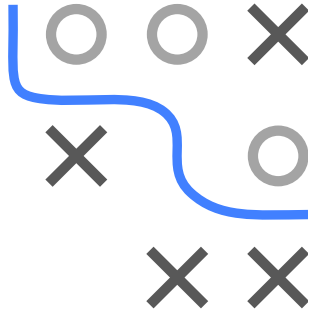
Iteration 10

Shape of PRs affects gradual model fit: *L1* only sees resids' sign, BLs are not affected size of values as in *L2* and hence lead to more moderate changes.

GAM WITH *L2* VS *L1* LOSS

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



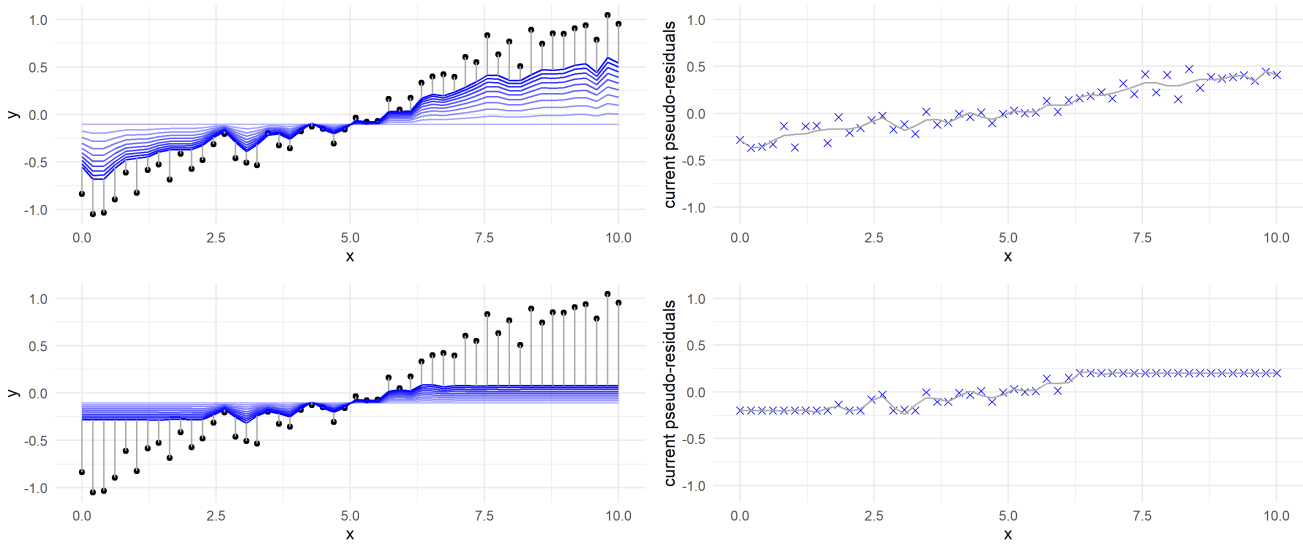


Iteration 100

Shape of PRs affects gradual model fit: *L1* only sees resids' sign, BLs are not affected size of values as in *L2* and hence lead to more moderate changes.

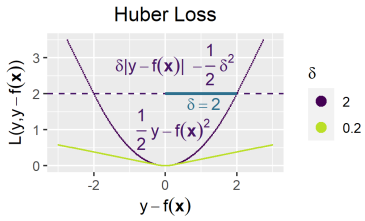
GAM WITH HUBER LOSS

Top: $\delta = 2$, bottom: $\delta = 0.2$.



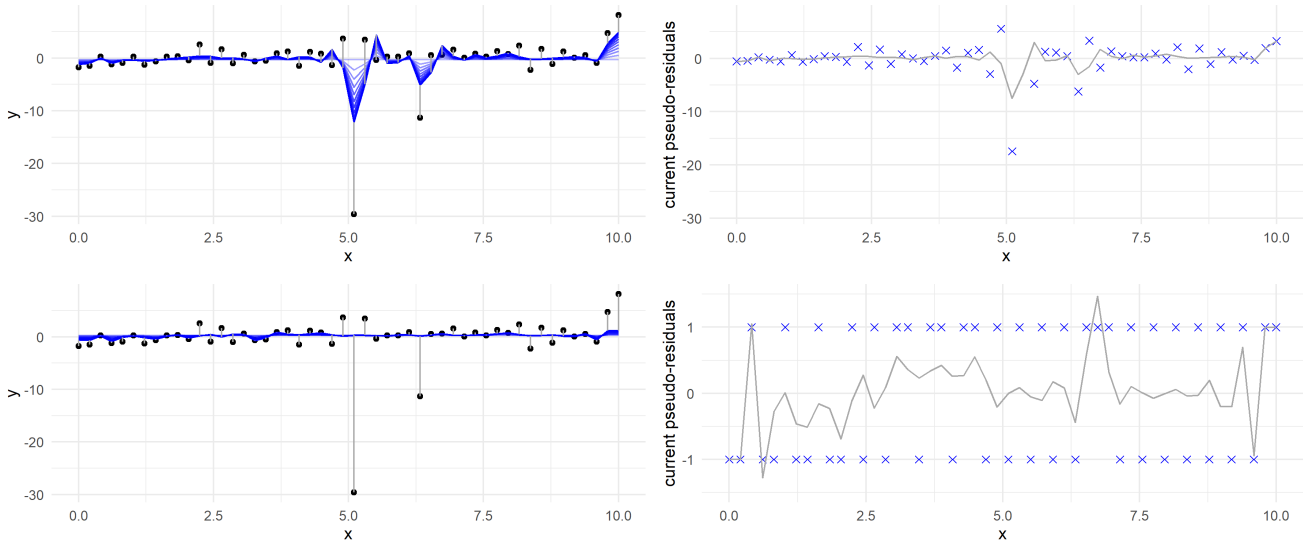
Iteration 10

For small δ, PRs are often bounded, resulting in $L1$-like behavior, while the upper plot more closely resembles $L2$ loss.



GAM WITH OUTLIERS

Instead of Gaussian noise, let's use t-distrib, that leads to outliers in y.
Top: $L2$, bottom: $L1$.

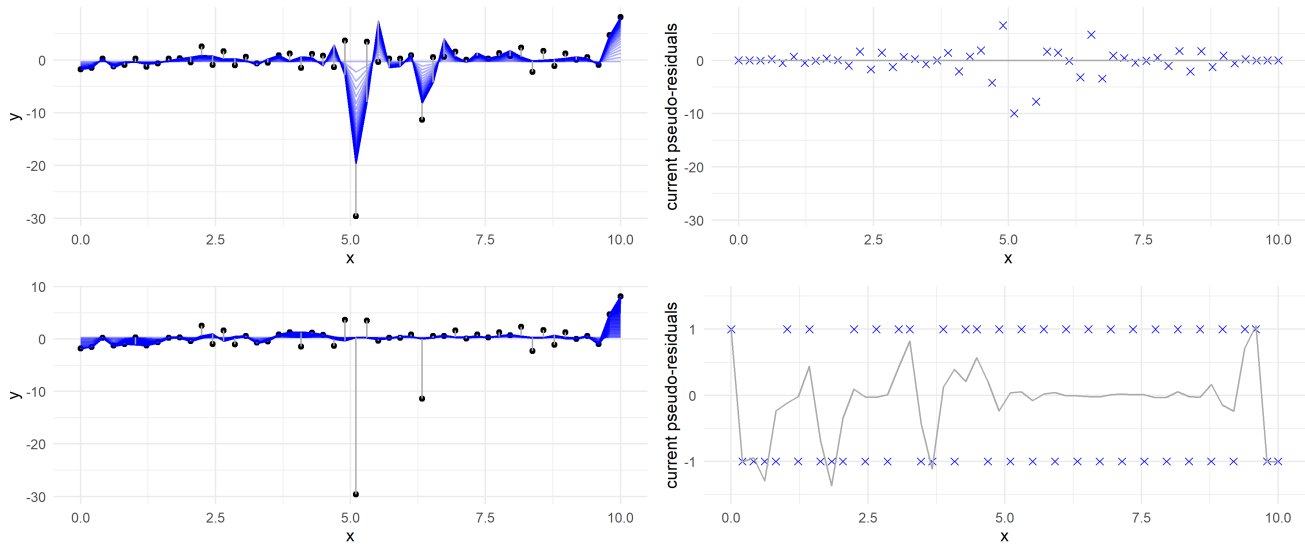


Iteration 10

$L2$ 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 Gaussian noise, let's use t-distrib, that leads to outliers in y.
Top: $L2$, bottom: $L1$.

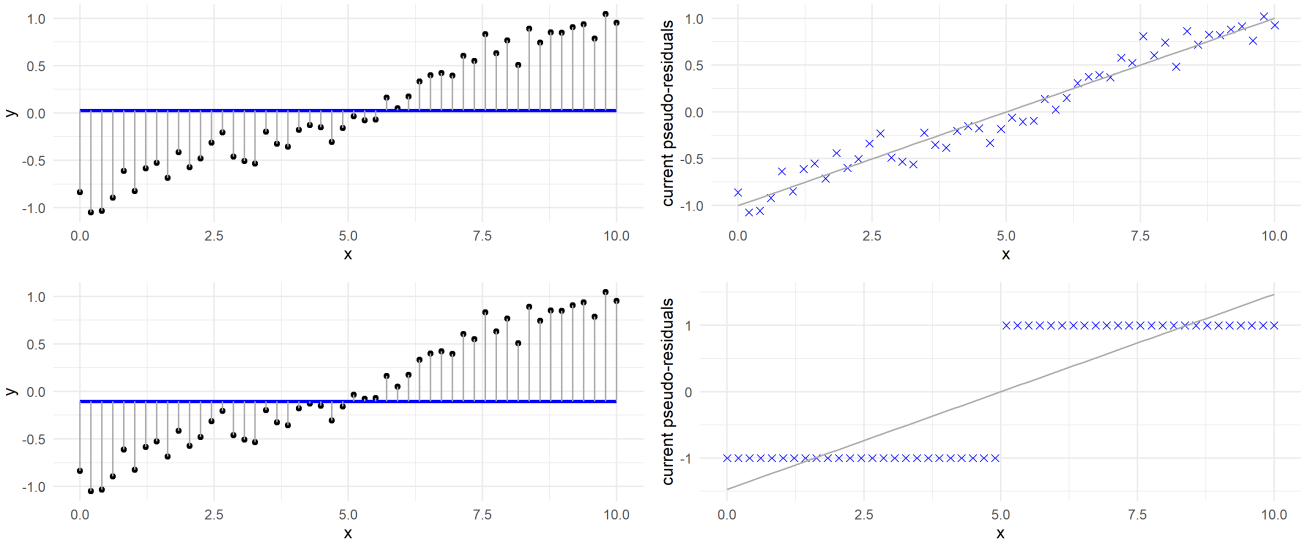


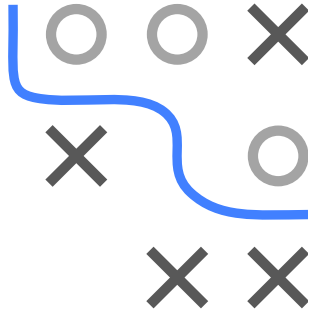
Iteration 100

$L2$ 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

Top: *L2*, bottom: *L1*.



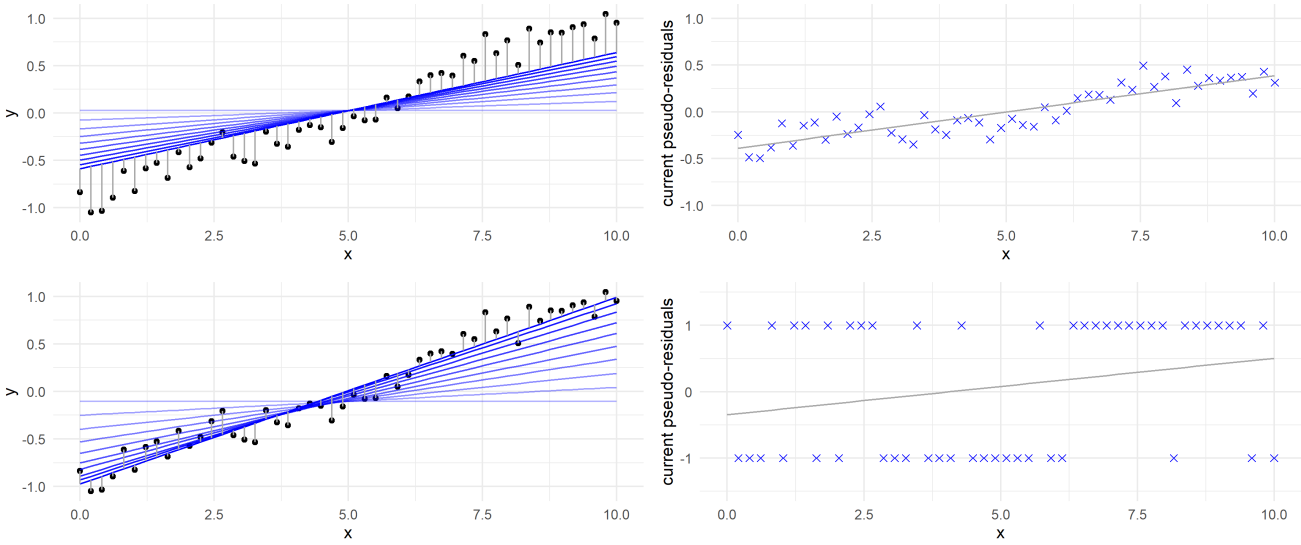


Iteration 1

L2: as $\tilde{r}(f) = r(f)$, BL of 1st iter already optimal; but learn rate slows us down.

LM WITH *L2* VS *L1* LOSS

Top: *L2*, bottom: *L1*.

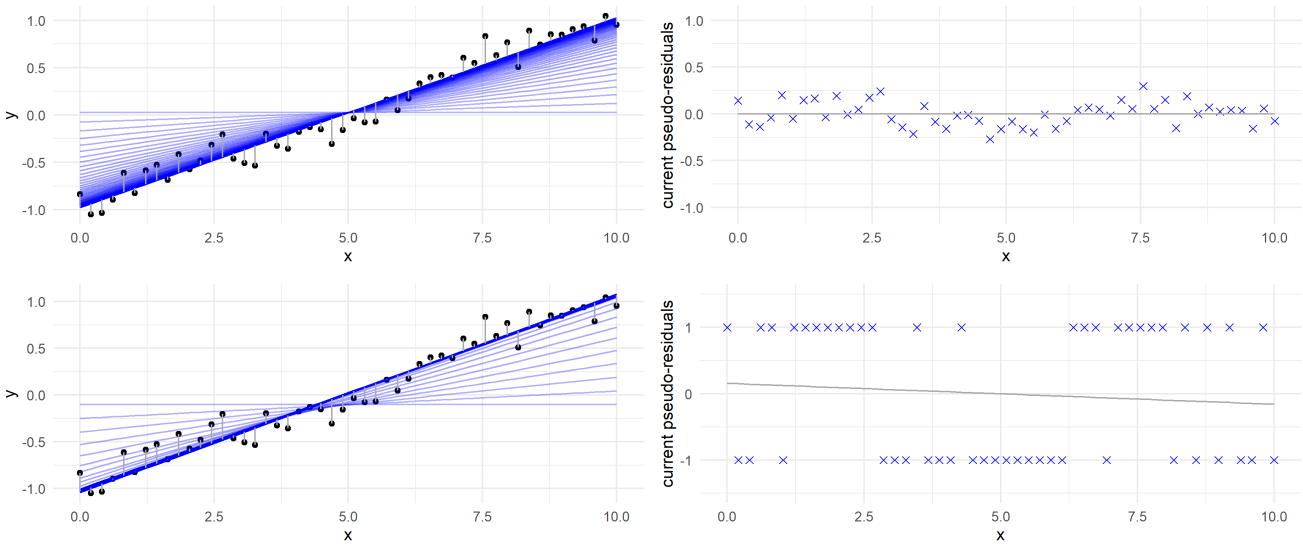


Iteration 10

L2: as $\tilde{r}(f) = r(f)$, BL of 1st iter already optimal; but learn rate slows us down.

LM WITH *L2* VS *L1* LOSS

Top: *L2*, bottom: *L1*.

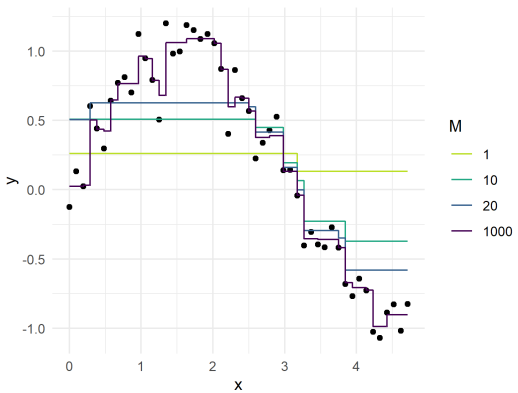


Iteration 100

L2: as $\tilde{r}(f) = r(f)$, BL of 1st iter already optimal; but learn rate slows us down.

Introduction to Machine Learning

Gradient Boosting: Regularization



Learning goals

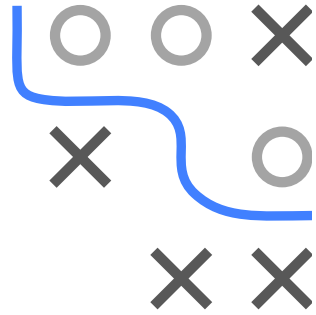
- Learn about three main regularization options: number of iterations, tree depth and shrinkage
- Understand how regularization influences model fit

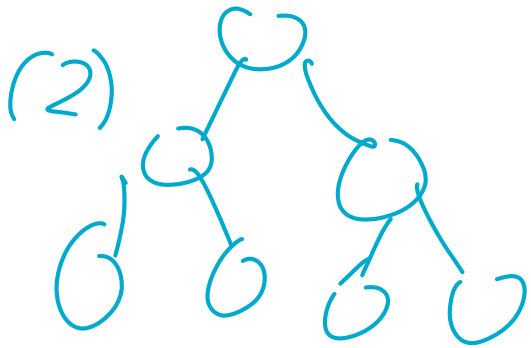
ITERS, TREE DEPTH, LEARN RATE

GB can overfit easily, due to its aggressive loss minimization.

Options for regularization:

- Limit nr of iters M, i.e., additive components ("early stopping"),
- Limit depth of trees. Can also be interpreted as choosing the order of interaction (see later).
- Use a small learn rate α for only mild model updates. α a.k.a. shrinkage.



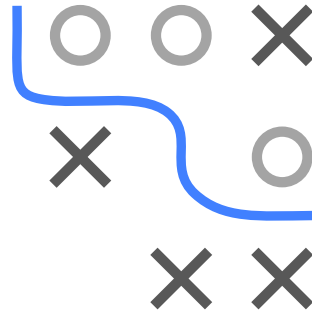


Practical hints:

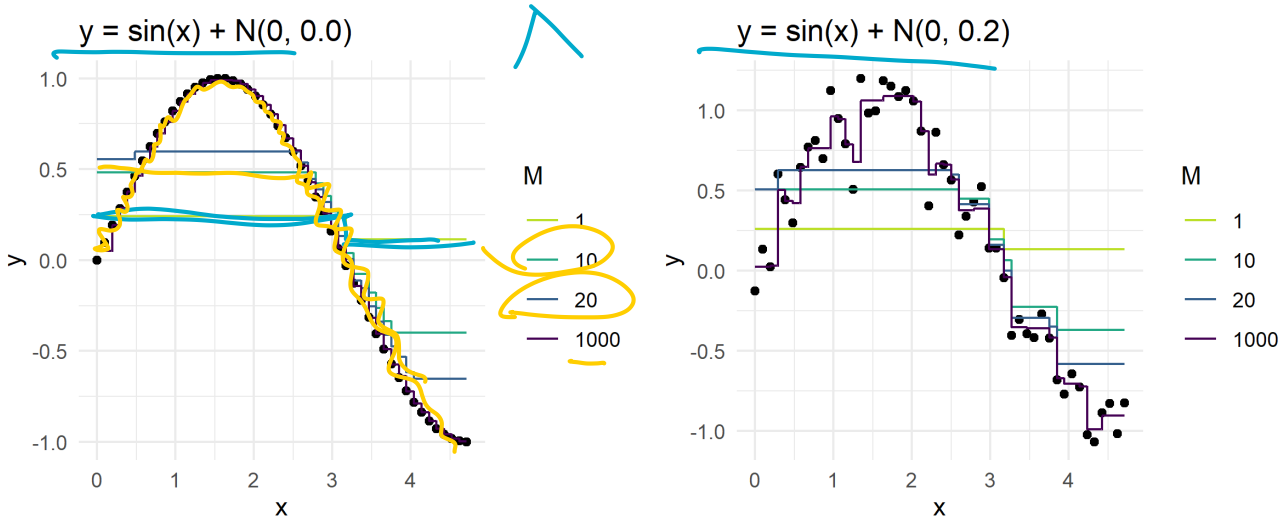
- Optimal values for M and α strongly depend on each other: by increasing M one can use a smaller value for α and vice versa.
- Fast option = Make α small and choose M by CV.
- Probably best to tune all 3 hyperpars jointly via, e.g., CV.

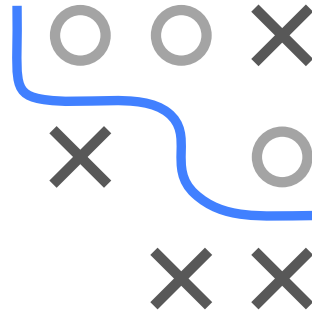
STOCHASTIC GRADIENT BOOSTING

- Minor modification to incorporate the advantages of bagging
- In each iter, we only fit on a random subsample of the train data
- Especially for small train sets, this often leads helps
- Size of random sets = new hyperpar



EXAMPLE: SINUSOIDAL WITH TREE STUMPS

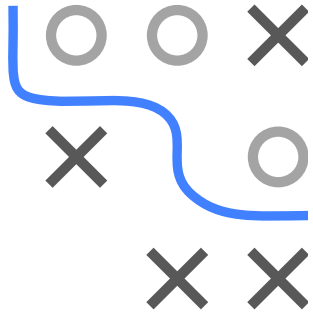


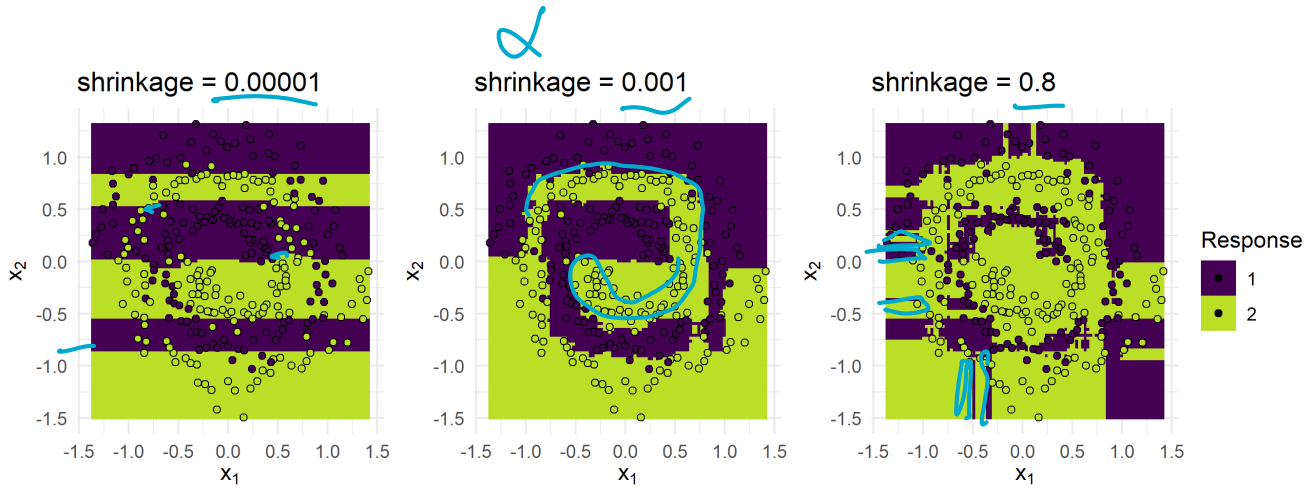


Works quite nicely without noise, but overfits on the RHS.

EXAMPLE: SPIRALS DATA

We examine effect of learn rate, with fixed nr of trees and fixed depth.

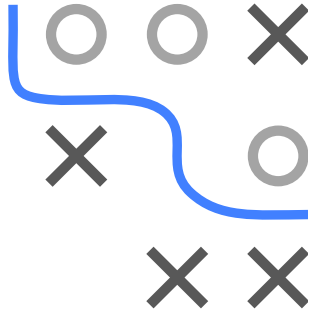




We observe an oversmoothing effect in the left scenario with strong regularization (i.e., very small learning rate) and overfitting when regularization is too weak (right). $\alpha = 0.001$ yields a pretty good fit.

EXAMPLE: SPAM DETECTION WITH TREES

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



Use 3-CV in grid search; optimal config in red: