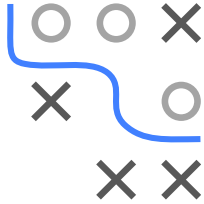
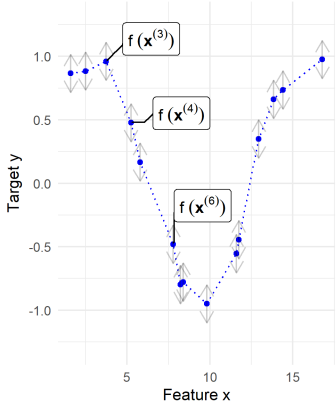
Introduction to Machine Learning



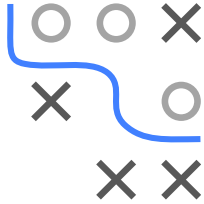
Boosting
Gradient Boosting: Concept



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 $\mathcal{B}$.

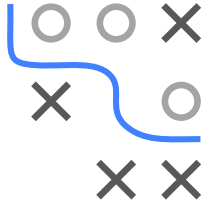
We want to learn an additive model:

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

Hence, we minimize the empirical risk:

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

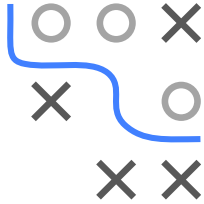
FORWARD STAGEWISE ADDITIVE MODELING / 2



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 / 3



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

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

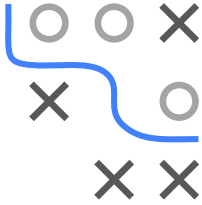
Doing this iteratively is called **forward stagewise additive modeling**.

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

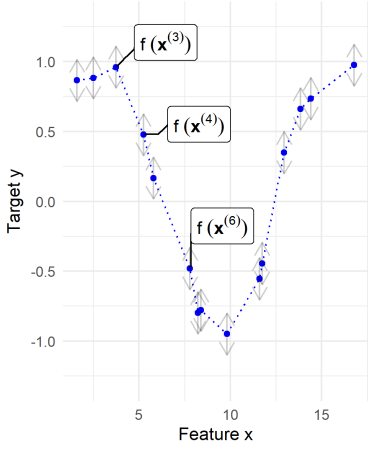
The algorithm we just introduced is not really an algorithm, but rather an abstract principle. We need to find the new additive component $b\left(\mathbf{x}, \boldsymbol{\theta}^{[m]}\right)$ and its weight coefficient $\alpha^{[m]}$ in each iteration m. This can be done by gradient descent, but in function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

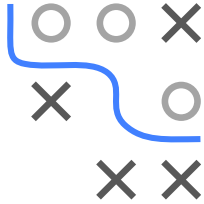


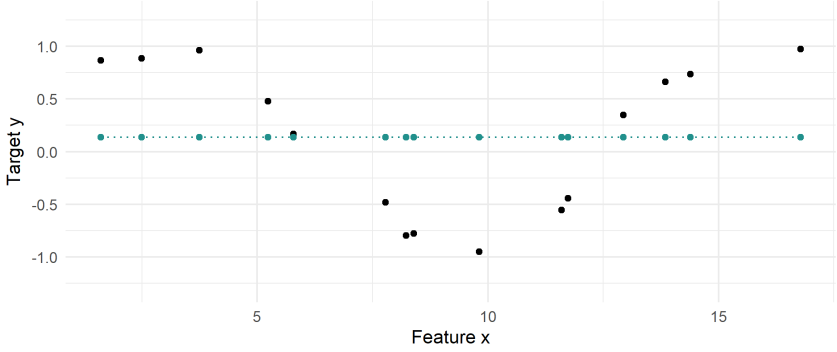
GRADIENT BOOSTING

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.

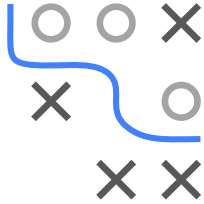
Initialization: Set all parameters to the optimal constant value (e.g., the mean of y for $L2$).





PSEUDO RESIDUALS

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

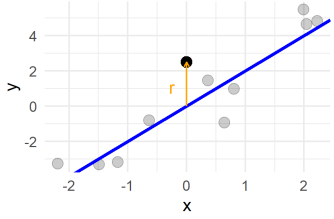


We minimize the risk of such a model with gradient descent (yes, this makes no sense, suspend all doubts for a few seconds).
So, we calculate the gradient at a point of the parameter space, that is, the derivative w.r.t. each component of the parameter vector (which is 0 for all terms with $i \neq j$):

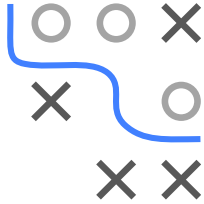
$$\tilde{r}^{(i)} = -\frac{\partial \mathcal{R}_{\text{emp}}}{\partial f\left(\mathbf{x}^{(i)}\right)} = -\frac{\partial \sum_j L(y^{(j)}, f(\mathbf{x}^{(j)}))}{\partial f\left(\mathbf{x}^{(i)}\right)} = -\frac{\partial L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)}{\partial f\left(\mathbf{x}^{(i)}\right)}.$$

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:

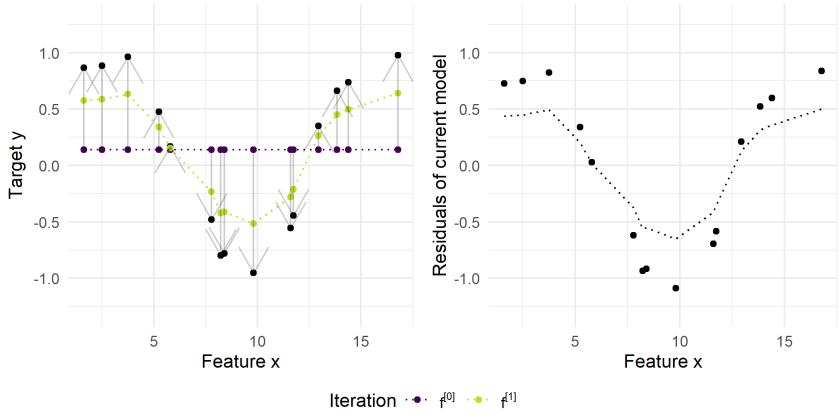
$$f^{[m]}(\mathbf{x}^{(i)}) = f^{[m-1]}(\mathbf{x}^{(i)}) - \alpha \frac{\partial L \left(y^{(i)}, f \left(\mathbf{x}^{(i)} \right) \right)}{\partial f^{[m-1]}(\mathbf{x}^{(i)})}.$$

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:

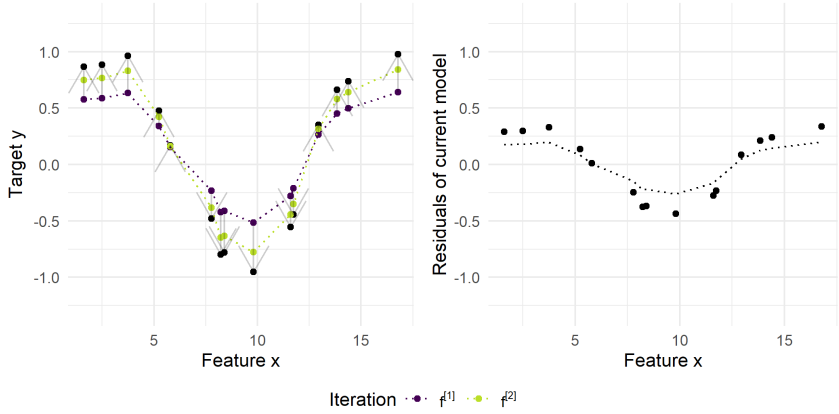
Let's move our function $f\left(\mathbf{x}^{(i)}\right)$ a fraction towards the pseudo-residuals with a learning rate of $\alpha = 0.6$.



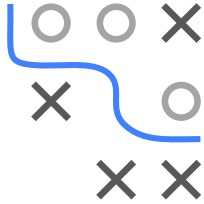
GRADIENT BOOSTING / 2

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 / 3

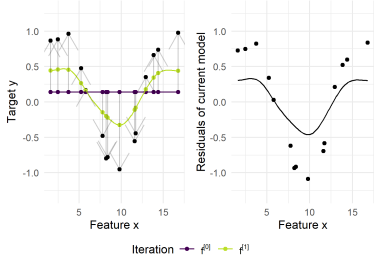


To parameterize a model in this way is pointless, as it just memorizes the instances of the training data.

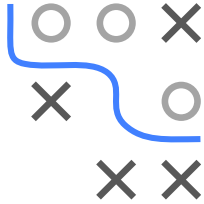
So, we restrict our additive components to $b\left(\mathbf{x}, \boldsymbol{\theta}^{[m]}\right) \in \mathcal{B}$.
The pseudo-residuals are calculated exactly as stated above, then we fit a simple model $b(\mathbf{x}, \boldsymbol{\theta}^{[m]})$ to them:

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

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.

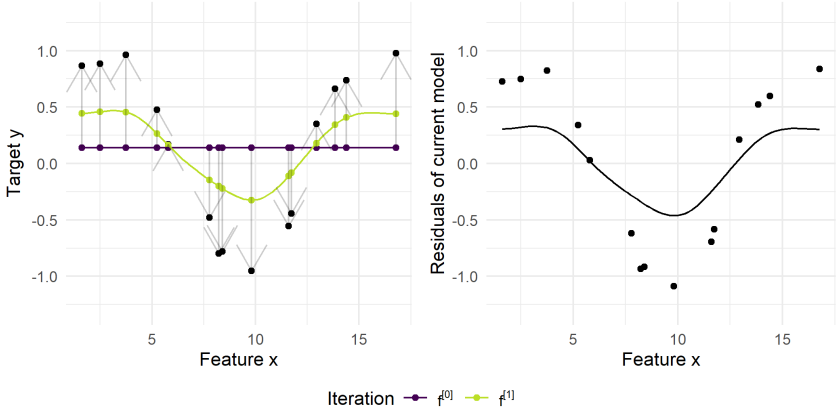


GRADIENT BOOSTING / 4



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.

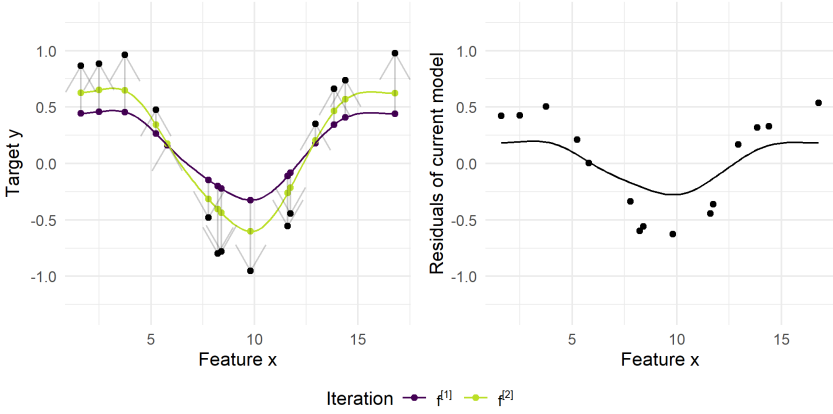
Iteration 1:



GRADIENT BOOSTING / 5

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).

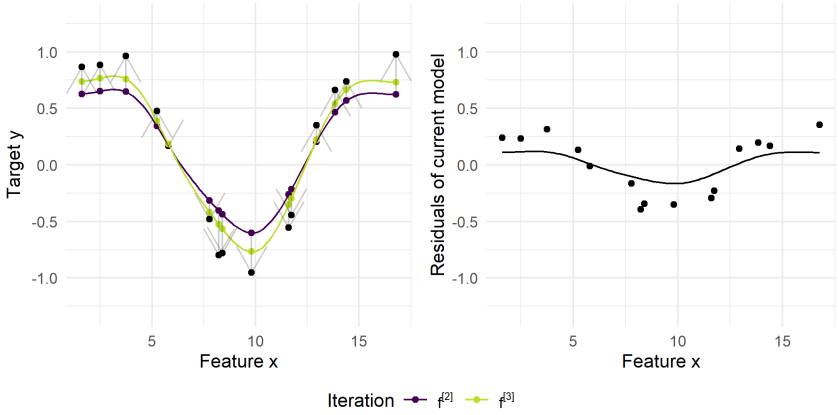
Iteration 2:



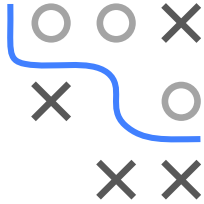
GRADIENT BOOSTING / 6

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

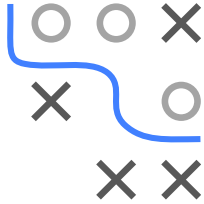


Algorithm Gradient Boosting Algorithm.

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** $m = 1 \rightarrow M$ **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: $\hat{\boldsymbol{\theta}}^{[m]} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} (\tilde{r}^{[m](i)} - b(\mathbf{x}^{(i)}, \boldsymbol{\theta}))^2$
6: Set $\alpha^{[m]}$ to α being a small constant value or via line search
7: Update $\hat{f}^{[m]}(\mathbf{x}) = \hat{f}^{[m-1]}(\mathbf{x}) + \alpha^{[m]} b(\mathbf{x}, \hat{\boldsymbol{\theta}}^{[m]})$
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 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:

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

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