(Gradient Tree) Boosting

History, adaptive complexity, sparsity, and some asymptotics

Berent Ånund Strømnes Lunde[1]

[1]Department of Mathematics
University of Bergen

MA8701 - Advanced statistical methods in inference and learning
NTNU, Norway
15th February 2021

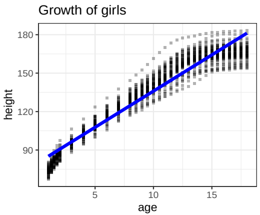
Outline

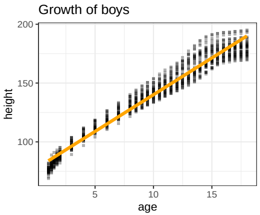
1. Background and development
2. Gradient Tree Boosting
3. Automatic GTB

Background outline

- Motivate the boosting technique.
- Layout of boosting-timeline.
- AdaBoost: The first boosting algorithm.
- 1'st order gradient boosting.
- Connection with boosting and sparsity.

Question 1: Linear regression

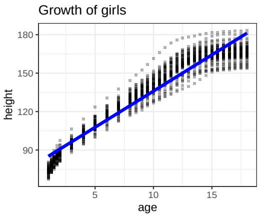


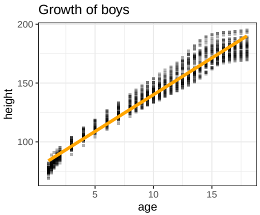


Researcher asks...

How can I model the `height` of children given their `age` and `sex`? And I need a model fast! [Berkeley growth curve dataset]

Question 1: Linear regression





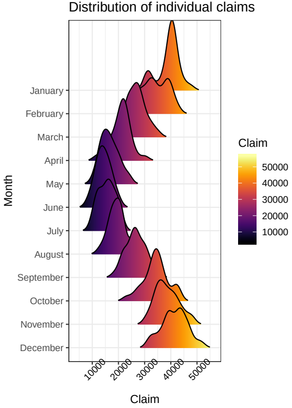
Researcher asks...

How can I model the height of children given their age and sex? And I need a model fast! [Berkeley growth curve dataset]

The statistician responds...

Easy! Just try a linear regression:
height $\approx \beta_0 + \beta_1$age $+ \beta_2$sex. Estimate parameters $\beta = \{\beta_0, \beta_1, \beta_2\}$ by minimizing the mean squared error (MSE):
$\hat{\beta} = \arg\min_\beta \sum_i \left(y_i - f(\text{age}_i, \text{sex}_i; \beta)\right)^2$.

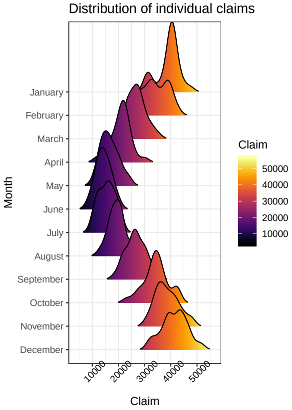
Question 2: Generalized linear models



Researcher asks...

Is there an efficient way to model the `risk` of customers of insurance given some history of `claims` and information about the customers? The model needs to be production friendly!

Question 2: Generalized linear models



Researcher asks...

Is there an efficient way to model the risk of customers of insurance given some history of claims and information about the customers? The model needs to be production friendly!

The actuary responds...

Easy! Divide and conquer: split the claims into size and frequency and model them using a gamma and a Poisson generalized linear model, respectively. The `glm()`-function in R is your friend.

Supervised learning

- The above problems may be framed as supervised learning:

Supervised learning

Find the best (in expectation, relative to loss l) predictive function:

$$\hat{f} = \arg\min_{f} E_{(\mathbf{x}^0, y^0, \hat{\theta})} \left[l(y^0, f(\mathbf{x}^0; \hat{\theta})) \right]$$

The loss often corresponds to a NLL. $(\mathbf{x}^0, y^0)$ independent of $\hat{\theta}$.

Supervised learning

- The above problems may be framed as supervised learning:

Supervised learning

Find the best (in expectation, relative to loss l) predictive function:

$$\hat{f} = \arg\min_{f} E_{(\mathbf{x}^0, y^0, \hat{\theta})} \left[l(y^0, f(\mathbf{x}^0; \hat{\theta})) \right]$$

The loss often corresponds to a NLL. $(\mathbf{x}^0, y^0)$ independent of $\hat{\theta}$.

User restricted f, is it...

- Non-linear?
- Continuous?
- Which features should it use?
- Do we have enough data to fit θ?

Supervised learning

- The above problems may be framed as supervised learning:

Supervised learning

Find the best (in expectation, relative to loss l) predictive function:

$$\hat{f} = \arg\min_{f} E_{(\mathbf{x}^0, y^0, \hat{\theta})} \left[l(y^0, f(\mathbf{x}^0; \hat{\theta})) \right]$$

The loss often corresponds to a NLL. $(\mathbf{x}^0, y^0)$ independent of $\hat{\theta}$.

User restricted f, is it...

- Non-linear?
- Continuous?
- Which features should it use?
- Do we have enough data to fit θ?

Gradient boosting

- Targets selection of $\hat{f}$ directly.
- Iterative procedure: Approximating gradient descent in function space.

Question 3: Gradient boosting

Researcher asks...

I want to do well in this ML-competition, but...

Question 3: Gradient boosting

Researcher asks...

I want to do well in this ML-competition, but...

- My data has missing values
- Both n and p are very large (design matrix with some billion elements)
- Relationships are non-linear and possibly discontinuous
- I don't care about explainability, just give me predictive power!

Question 3: Gradient boosting

Researcher asks...

I want to do well in this ML-competition, but...

- My data has missing values
- Both n and p are very large (design matrix with some billion elements)
- Relationships are non-linear and possibly discontinuous
- I don't care about explainability, just give me predictive power!

The data scientist/Kaggle master responds...

Try gradient boosting?

- State-of-the-art gradient boosting libraries: XGBoost, LightGBM and CatBoost.

Boosting timeline

- 1990, Schapire: The strength of weak learnability
- 1995, Freund & Schapire: AdaBoost
- 1997, Breiman: Arcing the Edge (why adaboost works)
- 1999, Friedman (1): Greedy function approximation and stochastic gradient boosting
- 1999, Mason et al.: Boosting algorithms as gradient descent in function space
- 2000, Friedman et al.: Additive Logistic Regression: A Statistical View of Boosting
- 2014 xgboost introduced in HIGGS Kaggle competition
- 2016 Chen & Guestring, xgboost article
- LightGBM, catboost, ngboost

Boosting timeline

- 1990, Schapire: The strength of weak learnability
- 1995, Freund & Schapire: AdaBoost
- 1997, Breiman: Arcing the Edge (why adaboost works)
- 1999, Friedman (1): Greedy function approximation and stochastic gradient boosting
- 1999, Mason et al.: Boosting algorithms as gradient descent in function space
- 2000, Friedman et al.: Additive Logistic Regression: A Statistical View of Boosting
- 2014 xgboost introduced in HIGGS Kaggle competition
- 2016 Chen & Guestring, xgboost article
- LightGBM, catboost, ngboost

Boosting timeline

- 1990, Schapire: The strength of weak learnability
- 1995, Freund & Schapire: AdaBoost
- 1997, Breiman: Arcing the Edge (why adaboost works)
- 1999, Friedman (1): Greedy function approximation and stochastic gradient boosting
- 1999, Mason et al.: Boosting algorithms as gradient descent in function space
- 2000, Friedman et al.: Additive Logistic Regression: A Statistical View of Boosting
- 2014 xgboost introduced in HIGGS Kaggle competition
- 2016 Chen & Guestring, xgboost article
- LightGBM, catboost, ngboost

Boosting timeline

- 1990, Schapire: The strength of weak learnability
- 1995, Freund & Schapire: AdaBoost
- 1997, Breiman: Arcing the Edge (why adaboost works)
- 1999, Friedman (1): Greedy function approximation and stochastic gradient boosting
- 1999, Mason et al.: Boosting algorithms as gradient descent in function space
- 2000, Friedman et al.: Additive Logistic Regression: A Statistical View of Boosting
- 2014 xgboost introduced in HIGGS Kaggle competition
- 2016 Chen & Guestring, xgboost article
- LightGBM, catboost, ngboost

Boosting timeline

- 1990, Schapire: The strength of weak learnability
- 1995, Freund & Schapire: AdaBoost
- 1997, Breiman: Arcing the Edge (why adaboost works)
- 1999, Friedman (1): Greedy function approximation and stochastic gradient boosting
- 1999, Mason et al.: Boosting algorithms as gradient descent in function space
- 2000, Friedman et al.: Additive Logistic Regression: A Statistical View of Boosting
- 2014 xgboost introduced in HIGGS Kaggle competition
- 2016 Chen & Guestring, xgboost article
- LightGBM, catboost, ngboost

Boosting timeline

- 1990, Schapire: The strength of weak learnability
- 1995, Freund & Schapire: AdaBoost
- 1997, Breiman: Arcing the Edge (why adaboost works)
- 1999, Friedman (1): Greedy function approximation and stochastic gradient boosting
- 1999, Mason et al.: Boosting algorithms as gradient descent in function space
- 2000, Friedman et al.: Additive Logistic Regression: A Statistical View of Boosting
- 2014 xgboost introduced in HIGGS Kaggle competition
- 2016 Chen & Guestring, xgboost article
- LightGBM, catboost, ngboost

Boosting timeline

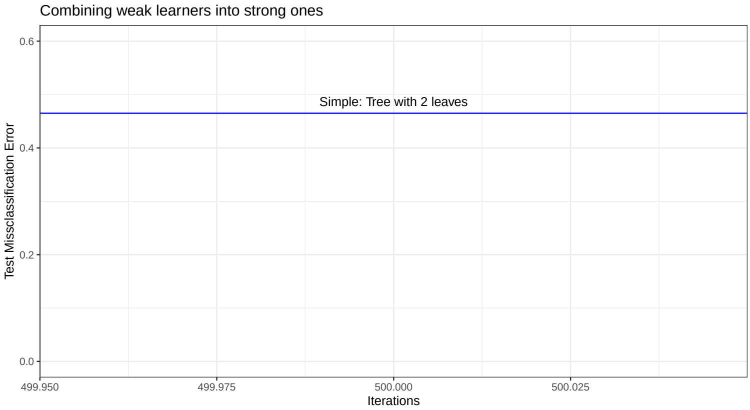
- 1990, Schapire: The strength of weak learnability
- 1995, Freund & Schapire: AdaBoost
- 1997, Breiman: Arcing the Edge (why adaboost works)
- 1999, Friedman (1): Greedy function approximation and stochastic gradient boosting
- 1999, Mason et al.: Boosting algorithms as gradient descent in function space
- 2000, Friedman et al.: Additive Logistic Regression: A Statistical View of Boosting
- 2014 xgboost introduced in HIGGS Kaggle competition
- 2016 Chen & Guestring, xgboost article
- LightGBM, catboost, ngboost

Boosting timeline

- 1990, Schapire: The strength of weak learnability
- 1995, Freund & Schapire: AdaBoost
- 1997, Breiman: Arcing the Edge (why adaboost works)
- 1999, Friedman (1): Greedy function approximation and stochastic gradient boosting
- 1999, Mason et al.: Boosting algorithms as gradient descent in function space
- 2000, Friedman et al.: Additive Logistic Regression: A Statistical View of Boosting
- 2014 xgboost introduced in HIGGS Kaggle competition
- 2016 Chen & Guestring, xgboost article
- LightGBM, catboost, ngboost

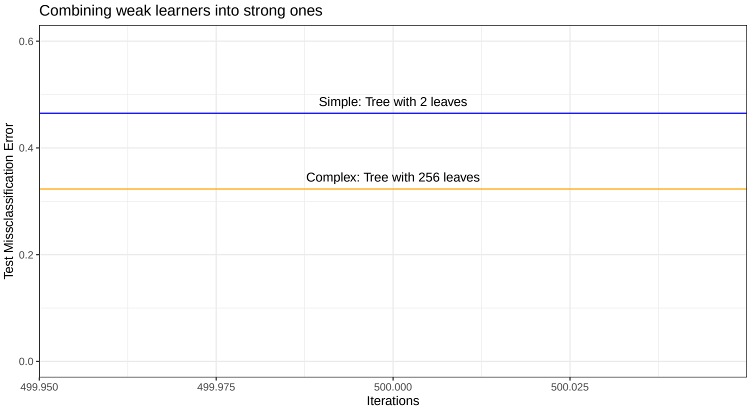
The boosting principle

- What did Schapire figure out in 1990?



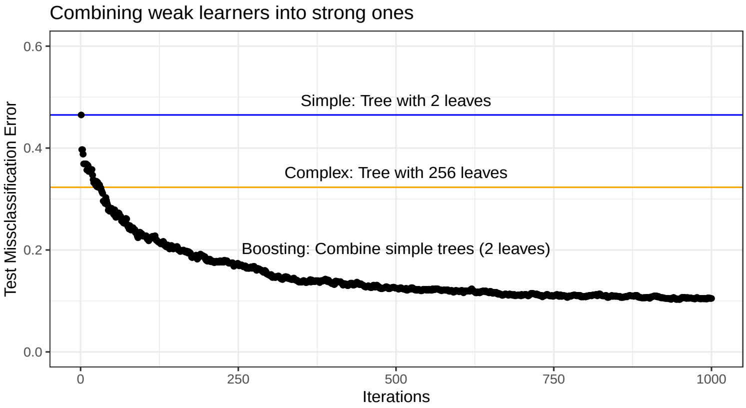
The boosting principle

- What did Schapire figure out in 1990?



The boosting principle

- What did Schapire figure out in 1990?



AdaBoost

The first algorithm to employ the modular boosting principle:

- Final model is an additive combination of simpler models: $\hat{y} = \text{sign}\left[f^{(K)}(\mathbf{x})\right]$.
- The "weak learners" are learned in an iterative manner, with objective

$$\{f_k, \alpha_k\} = \arg\min_{f,\alpha} \sum_{i=1}^{n} l(y_i, f^{(k-1)}(\mathbf{x}) + \alpha f(\mathbf{x}_i))$$

  at iteration k.
- Crucially, the loss is the "exponential loss": $l(y, \hat{y}) = e^{-y\hat{y}}$.
- This gives a solution to the above objective

$$f_k = \arg\min \sum_{i=1}^{n} w_i I(y_i \neq f(\mathbf{x}_i)),\ \alpha_k = \log\left(\frac{1-\text{err}_k}{\text{err}_k}\right),\ \text{err}_k = \frac{\sum_{i=1}^{n} w_i I(y_i \neq f(\mathbf{x}_i))}{\sum_{i=1}^{n} w_i}$$

AdaBoost algorithm

1. Initialize the training weights: $w_i = \frac{1}{n}, \; i = 1, \cdots, n$
2. **for** $k = 1$ to K:

   i) Fit a classifier, $f_k(\mathbf{x})$, to the training data using weights w_i.

   $$f_k = \arg\min \sum_{i=1}^{n} w_i I(y_i \neq f(\mathbf{x}_i))$$

   ii) Compute model-weight

   $$\alpha_k = \log\left(\frac{1-\mathrm{err}_k}{\mathrm{err}_k}\right), \; \mathrm{err}_k = \frac{\sum_{i=1}^{n} w_i I(y_i \neq f(\mathbf{x}_i))}{\sum_{i=1}^{n} w_i}$$

   iii) Recompute training weights:

   $$w_i \leftarrow w_i \exp\left(\alpha_k I(y_i \neq f_k(\mathbf{x}_i))\right), \; i = 1, \cdots, n$$

   end for
3. **Return** $f^{(K)}(\mathbf{x}) = \sum_{k=1}^{K} \alpha_k f_k(\mathbf{x})$ to use with `sign()`.

From AdaBoost to gradient boosting

- Exponential loss has the same population minimizer as binomial NLL.
- But it is less robust. The motivation is computational.
  - Leads to ”nice” result for iterative reweighting:

From AdaBoost to gradient boosting

- Exponential loss has the same population minimizer as binomial NLL.
- But it is less robust. The motivation is computational.
  - Leads to ”nice” result for iterative reweighting:
- Breiman 1997 answers why AdaBoost works.
  - Gives hints towards iterative optimization of *some* objective function.

From AdaBoost to gradient boosting

- Exponential loss has the same population minimizer as binomial NLL.
- But it is less robust. The motivation is computational.
  - Leads to "nice" result for iterative reweighting:
- Breiman 1997 answers why AdaBoost works.
  - Gives hints towards iterative optimization of *some* objective function.
- Question: Possible to construct a more general algorithm?
  - Solve: $\hat{f} = \arg\min_f \sum_i l(y_i, \hat{y}_i + f(\mathbf{x}_i))$ for more general loss functions.

From AdaBoost to gradient boosting

- Exponential loss has the same population minimizer as binomial NLL.
- But it is less robust. The motivation is computational.
  - Leads to ”nice” result for iterative reweighting:
- Breiman 1997 answers why AdaBoost works.
  - Gives hints towards iterative optimization of *some* objective function.
- Question: Possible to construct a more general algorithm?
  - Solve: $\hat{f} = \arg\min_f \sum_i l(y_i, \hat{y}_i + f(\mathbf{x}_i))$ for more general loss functions.
- Answered ”yes” in 1999:
  - Friedman: Greedy function approximation: A gradient boosting machine
  - Mason et al.: Boosting algorithms as gradient descent in function space

Gradient boosting (1'st order)

Input:

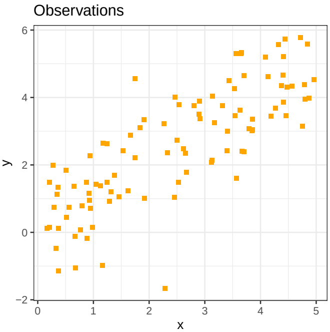
- A training set $\mathcal{D}_n = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$,
- a differentiable loss $l(y, f(\mathbf{x}))$,
- a family of base-learners $\mathcal{H}$,

1. Initialize model with a constant value:
   $f^{(0)} = \underset{\eta}{\arg\min} \sum_{i=1}^n l(y_i, \eta)$.
2. **for** $k = 1$ to K:
   $i)$ Compute derivatives g_i for all $i = 1 : n$.
   $ii)$ Fit a base-learner $f_k(\mathbf{x}) \in \mathcal{H}$ to $\{-g_i, \mathbf{x}\}_{i=1}^n$ using MSE-loss.
   $iii)$ Find an optimized scaling α_k of f_k:
   $\hat{\alpha}_k = \underset{\alpha}{\arg\min} \sum_{i=1}^n l(y_i, f^{(k-1)}(\mathbf{x}_i) + \alpha f_k(\mathbf{x}_i))$.
   $v)$ Update the model with a scaled base-learner (δ "small"): $f^{(k)}(\mathbf{x}) = f^{(k-1)}(\mathbf{x}) + \delta \hat{\alpha}_k f_k(\mathbf{x})$.
   end for
3. **Return** $f^{(K)}(\mathbf{x})$.

Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

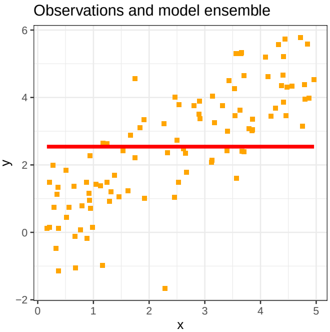
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.



Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

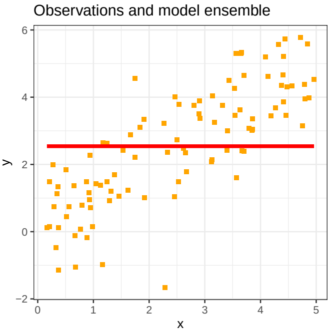
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

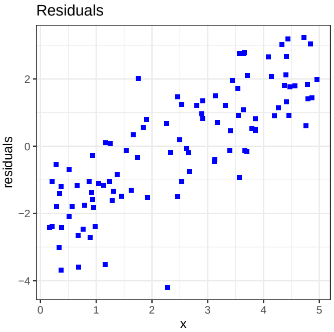


Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

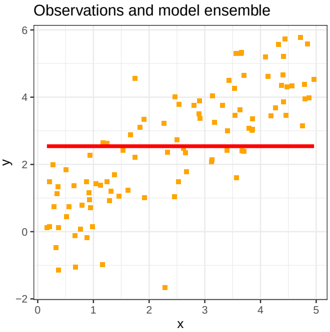


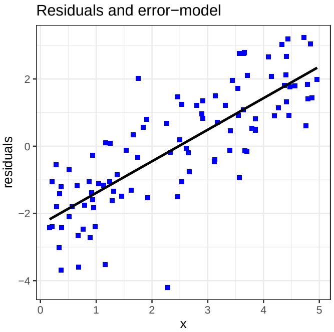


Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

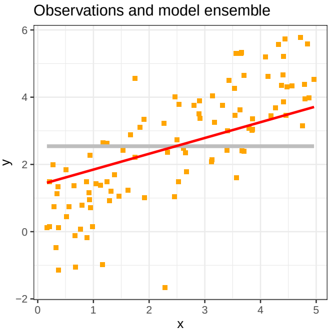


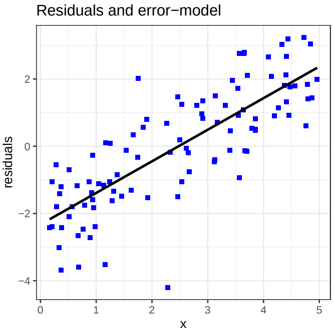


Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

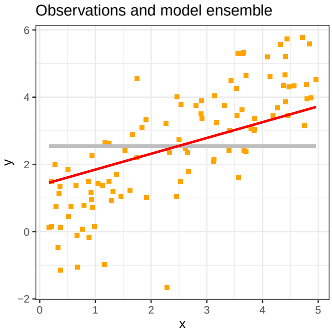


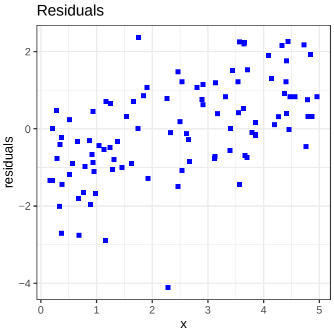


Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

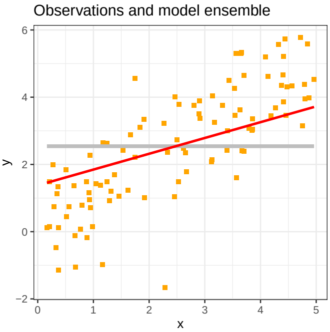


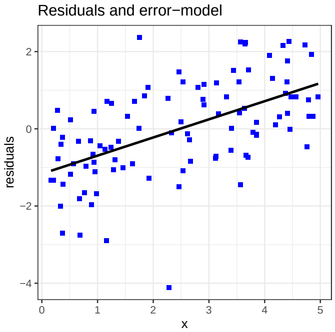


Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

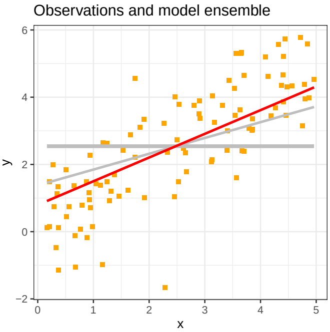


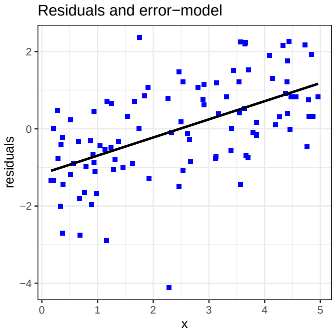


Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

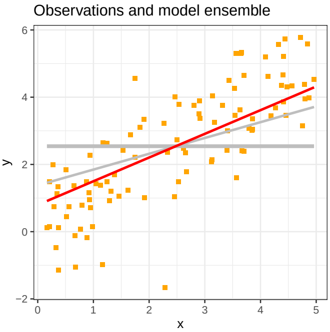


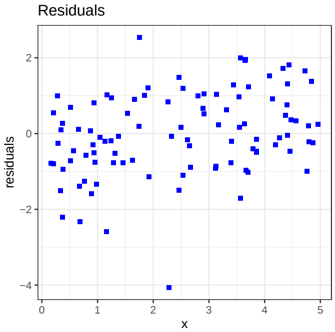


Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

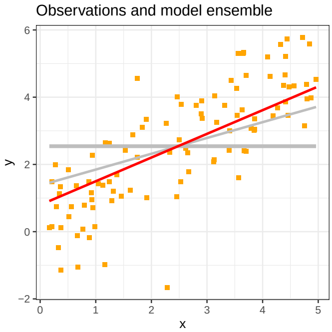


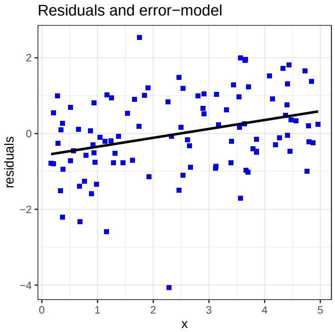


Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

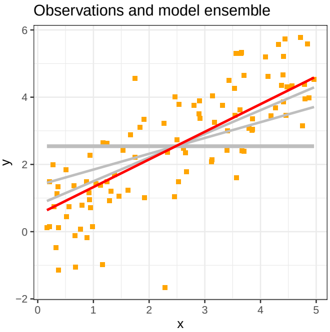


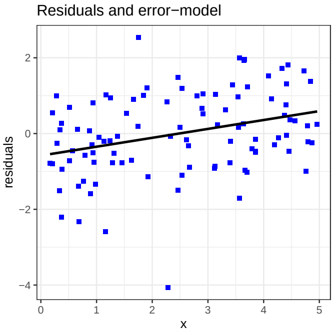


Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

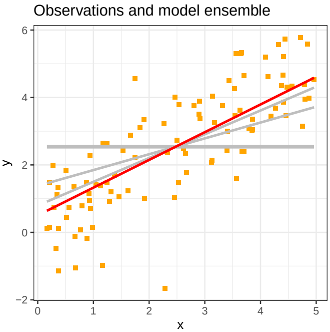


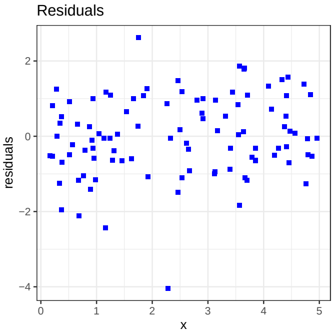


Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

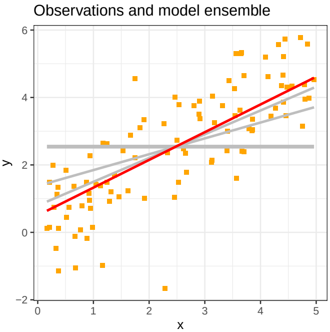


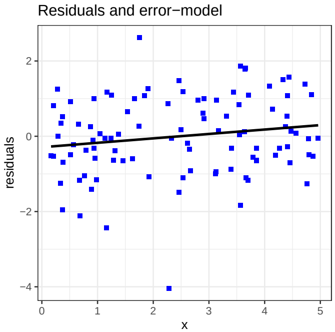


Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

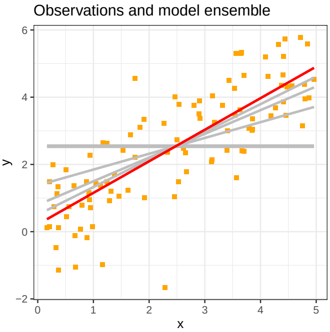


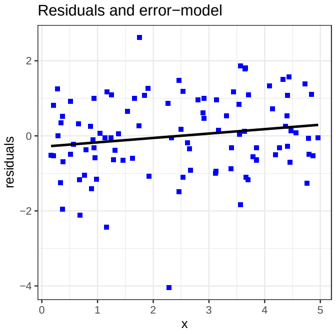


Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.





Why this iterative procedure is a good idea

The procedure...

Why this iterative procedure is a good idea

The procedure...

- Adapts the complexity of the model, f, to the data,
- Only add as much complexity in a certain direction as it deserves
- Builds sparse models: Connection to the LARS algorithm for computing LASSO solution paths.

Why this iterative procedure is a good idea

The procedure...

- Adapts the complexity of the model, f, to the data,
- Only add as much complexity in a certain direction as it deserves
- Builds sparse models: Connection to the LARS algorithm for computing LASSO solution paths.

As we have seen, it goes beyond only residuals:

- Given a differentiable loss function l
- Instead of building a model on the "errors" in the MSE case,
- Compute derivatives from $l(y_i, \hat{y}_i)$ over the data given predictions $\hat{y}_i$ from the current model.
- Build a model on the derivatives.

Boosting as gradient descent in function space

- It is complicated, see Maseon et al. for details.
- At best we are approximating functional gradient descent.
  - Find $f \in \mathcal{H}$ that is closest (measured by MSE) to the true functional gradient, according to a sample from the gradient $\{-g_i, \mathbf{x}_i\}_{i=1}^n$.

Relationship to L1 regularization

Lasso & LARS & ϵ

- LASSO computes Lasso (without deletion step) solution paths.
- The ϵ-algorithm converges to LARS when $\epsilon \to 0$.

Relationship to L1 regularization

Lasso & LARS & ϵ

- LASSO computes Lasso (without deletion step) solution paths.
- The ϵ-algorithm converges to LARS when $\epsilon \to 0$.

The ϵ-algorithm

1. Initialize: $\beta = 0$ and $\epsilon > 0$.
2. Repeat until convergence
   (a) $\mathbf{r} = \mathbf{y} - \mathbf{X}\beta$
   (b) $j = \arg\min_k |\mathbf{r} - X^k \beta_k|_2$
   (c) $\beta_j = \beta_j + \rho\epsilon$.

Relationship to L1 regularization

Lasso & LARS & ϵ

- LASSO computes Lasso (without deletion step) solution paths.
- The ϵ-algorithm converges to LARS when $\epsilon \to 0$.

The ϵ-algorithm

1. Initialize: $\beta = 0$ and $\epsilon > 0$.
2. Repeat until convergence
   (a) $\mathbf{r} = \mathbf{y} - \mathbf{X}\beta$
   (b) $j = \arg\min_k |\mathbf{r} - X^k \beta_k|_2$
   (c) $\beta_j = \beta_j + \rho\epsilon$.

Implications for boosting

- The ϵ-algorithm is obviously boosting.

Relationship to L1 regularization

Lasso & LARS & ϵ

- LASSO computes Lasso (without deletion step) solution paths.
- The ϵ-algorithm converges to LARS when $\epsilon \to 0$.

The ϵ-algorithm

1. Initialize: $\beta = 0$ and $\epsilon > 0$.
2. Repeat until convergence
   (a) $\mathbf{r} = \mathbf{y} - \mathbf{X}\beta$
   (b) $j = \arg\min_k |\mathbf{r} - X^k \beta_k|_2$
   (c) $\beta_j = \beta_j + \rho\epsilon$.

Implications for boosting

- The ϵ-algorithm is obviously boosting.
- But, the direct connection to L1-penalization holds only in the special case above.

Relationship to L1 regularization

Lasso & LARS & ϵ

- LASSO computes Lasso (without deletion step) solution paths.
- The ϵ-algorithm converges to LARS when $\epsilon \to 0$.

The ϵ-algorithm

1. Initialize: $\beta = 0$ and $\epsilon > 0$.
2. Repeat until convergence
   (a) $\mathbf{r} = \mathbf{y} - \mathbf{X}\beta$
   (b) $j = \arg\min_k |\mathbf{r} - X^k \beta_k|_2$
   (c) $\beta_j = \beta_j + \rho\epsilon$.

Implications for boosting

- The ϵ-algorithm is obviously boosting.
- But, the direct connection to L1-penalization holds only in the special case above.
- The general case for boosting is unknown.

Relationship to L1 regularization

Lasso & LARS & ϵ

- LASSO computes Lasso (without deletion step) solution paths.
- The ϵ-algorithm converges to LARS when $\epsilon \to 0$.

The ϵ-algorithm

1. Initialize: $\beta = 0$ and $\epsilon > 0$.
2. Repeat until convergence
   (a) $\mathbf{r} = \mathbf{y} - \mathbf{X}\beta$
   (b) $j = \arg\min_k |\mathbf{r} - X^k \beta_k|_2$
   (c) $\beta_j = \beta_j + \rho\epsilon$.

Implications for boosting

- The ϵ-algorithm is obviously boosting.
- But, the direct connection to L1-penalization holds only in the special case above.
- The general case for boosting is unknown.
- Gives a hint to type of base-learner: Learn one "direction" of parameter space at a time.

Techniques for improvement

- The "learning rate" or "shrinkage", $0 < \delta \leq 1$, is crucial.
- We have an additive combination of functions: $\hat{y} = f^(K)(\mathbf{x}) = f_0 + \sum_{i=1}^{K} f_k(\mathbf{x})$
  - Intuitively, the ensemble should improve if f_k elements are decorrelated.
  - Subsampling of both rows and columns usually give significant improvements.
- L1/L2-type regularization (priors on $f_k(\mathbf{x};\theta)$ parameters).
- Weak-learner dependent techniques.
  - Trees as the most popular weak-learner typically have multiple tuning parameters.

Recap boosting concept and development

- Boosting: Make weak-learners strong, iteratively.
- Many different boosting algorithms exist.
  - AdaBoost (for classification) was first. Relies on some ”exponential loss”, that leads to nice results for computation.
  - Will soon mention xgboost, lightgbm, catboost, and ngboost.
- Gradient boosting for general differentiable loss functions.
  - Approximate functional gradient descent.
- Connection to L1 regularization and sparsity.
  - Explicit connection to LARS for linear regression.
  - The general case is unknown.

Question

- Intuitively, what is a good base-learner?
  - Consider trees, linear functions, smoothing splines, even Neural Nets?

Gradient tree boosting outline

- Why does trees work.
- The XGBoost flavour (2'nd order GTB).
- The loss vs complexity tradeoff in GTB (what is complexity?)
- Hyperparameter tuning

Trees: where boosting gets interesting

Previously we used a linear model for "base learners" f_k:

- The linear combination of linear functions is still a linear model...

Trees: where boosting gets interesting

Previously we used a linear model for "base learners" f_k:

- The linear combination of linear functions is still a linear model...

- More interesting with non-linear learning procedures for f_k
- But needs to retain the possibility of a simple (sparse) model.
- We need something that can be non-linear but adapts this to data!

Trees: where boosting gets interesting

Previously we used a linear model for ”base learners” f_k:

- The linear combination of linear functions is still a linear model...

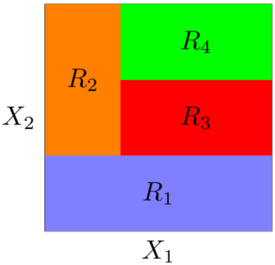
- More interesting with non-linear learning procedures for f_k
- But needs to retain the possibility of a simple (sparse) model.
- We need something that can be non-linear but adapts this to data!
- Trees: complexity from the simple mean or ”tree-stumps” to potentially a complete fit to training data.

The tree-learning procedure: Recursive binary splitting

Trees are constant predictions in T regions, R_t, of feature space:

$$\hat{y} = \sum_{t=1}^{T} w_t I(\mathbf{x} \in R_t)$$

But how do we choose the regions R_t?

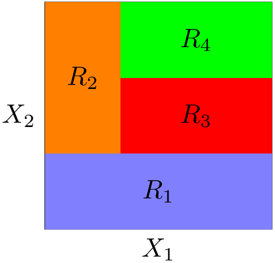


The tree-learning procedure: Recursive binary splitting

Trees are constant predictions in T regions, R_t, of feature space:

$$\hat{y} = \sum_{t=1}^{T} w_t I(\mathbf{x} \in R_t)$$

But how do we choose the regions R_t?



Recursive binary splitting

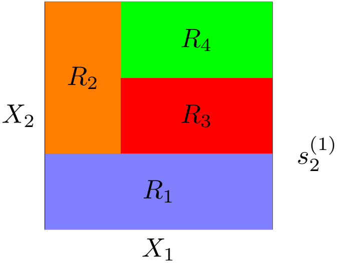
1. Start with a constant prediction for all of feature space
2. Split a leaf-node into two regions (on feature j and split-point s_j chosen by some criteria).
3. Continue step 2 recursively on all leaves.

The tree-learning procedure: Recursive binary splitting

Trees are constant predictions in T regions, R_t, of feature space:

$$\hat{y} = \sum_{t=1}^{T} w_t I(\mathbf{x} \in R_t)$$

But how do we choose the regions R_t?



Recursive binary splitting

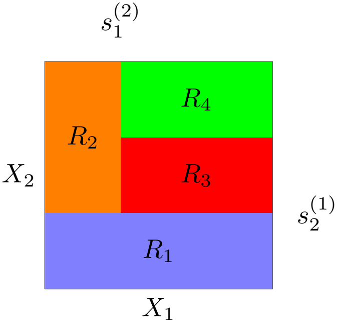
1. Start with a constant prediction for all of feature space
2. Split a leaf-node into two regions (on feature j and split-point s_j chosen by some criteria).
3. Continue step 2 recursively on all leaves.

The tree-learning procedure: Recursive binary splitting

Trees are constant predictions in T regions, R_t, of feature space:

$$\hat{y} = \sum_{t=1}^{T} w_t I(\mathbf{x} \in R_t)$$

But how do we choose the regions R_t?



Recursive binary splitting

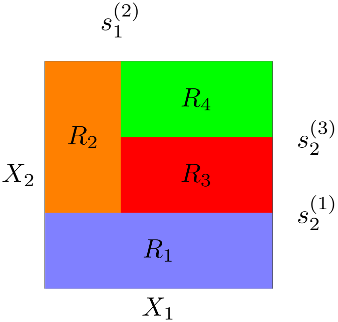
1. Start with a constant prediction for all of feature space
2. Split a leaf-node into two regions (on feature j and split-point s_j chosen by some criteria).
3. Continue step 2 recursively on all leaves.

The tree-learning procedure: Recursive binary splitting

Trees are constant predictions in T regions, R_t, of feature space:

$$\hat{y} = \sum_{t=1}^{T} w_t I(\mathbf{x} \in R_t)$$

But how do we choose the regions R_t?



Recursive binary splitting

1. Start with a constant prediction for all of feature space
2. Split a leaf-node into two regions (on feature j and split-point s_j chosen by some criteria).
3. Continue step 2 recursively on all leaves.

Explicit 2'nd order GTB: Objective

- Given an initial function $f^{(k-1)}(\mathbf{x})$, one ideally seeks a function $f_k(\mathbf{x})$ minimizing

$$\hat{f}_k(\mathbf{x}^0) = \arg\min_{f_k} E\left[l\left(y^0, f^{(k-1)}(\mathbf{x}^0) + f_k(\mathbf{x}^0)\right)\right]. \tag{1}$$

Explicit 2'nd order GTB: Objective

- Given an initial function $f^{(k-1)}(\mathbf{x})$, one ideally seeks a function $f_k(\mathbf{x})$ minimizing

$$\hat{f}_k(\mathbf{x}^0) = \arg\min_{f_k} E\left[l\left(y^0, f^{(k-1)}(\mathbf{x}^0) + f_k(\mathbf{x}^0)\right)\right]. \tag{1}$$

  - Approximate by the functional derivative?
  - The distribution (and also the expectation) of $(\mathbf{x}, y)$ is unknown.

Explicit 2'nd order GTB: Objective

- Given an initial function $f^{(k-1)}(\mathbf{x})$, one ideally seeks a function $f_k(\mathbf{x})$ minimizing

$$\hat{f}_k(\mathbf{x}^0) = \arg\min_{f_k} E\left[l\left(y^0, f^{(k-1)}(\mathbf{x}^0) + f_k(\mathbf{x}^0) \right) \right]. \tag{1}$$

  - Approximate by the functional derivative?
  - The distribution (and also the expectation) of $(\mathbf{x}, y)$ is unknown.

- Approximate the expectation and the objective by averaging the training set.

$$\hat{f}_k(\mathbf{x}) = \arg\min_{f_k} \frac{1}{n} \sum_{i=1}^{n} l\left(y_i, f^{(k-1)}(\mathbf{x}_i) + f_k(\mathbf{x}_i) \right), \tag{2}$$

  - Still a hard problem for arbitrary l and f_k.

Explicit 2'nd order GTB: Approximation

- Still a hard problem for arbitrary l and f_k

$$\hat{f}_k(\mathbf{x}) = \arg\min_{f_k} \frac{1}{n} \sum_{i=1}^{n} l\left(y_i, f^{(k-1)}(\mathbf{x}_i) + f_k(\mathbf{x}_i)\right). \tag{3}$$

Explicit 2'nd order GTB: Approximation

- Still a hard problem for arbitrary l and f_k

$$\hat{f}_k(\mathbf{x}) = \arg\min_{f_k} \frac{1}{n} \sum_{i=1}^{n} l\left(y_i, f^{(k-1)}(\mathbf{x}_i) + f_k(\mathbf{x}_i)\right). \tag{3}$$

- Do two things:
  1. Let $\hat{y}^{(k-1)} = f^{(k-1)}(\mathbf{x}_i)$ and $g_{i,k} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i^{(k-1)})$, $h_{i,k} = \frac{\partial^2}{\partial \hat{y}_i{}^2} l(y_i, \hat{y}_i^{(k-1)})$, then

$$\begin{aligned}\hat{f}_k(\mathbf{x}) &= \arg\min_{f_k} \frac{1}{n} \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(k-1)}) + g_{i,k} f_k(\mathbf{x}_i) + \frac{1}{2} h_{i,k} f_k(\mathbf{x}_i)^2 \\ &= \arg\min_{f_k} \frac{1}{n} \sum_{i=1}^{n} g_{i,k} f_k(\mathbf{x}_i) + \frac{1}{2} h_{i,k} f_k(\mathbf{x}_i)^2 \end{aligned} \tag{4}$$

  2. Search for f_k in the space of classification and regression trees (CART).

Explicit 2'nd order GTB: Split-enumeration and leaf-weights

- Let $q_k : \mathbb{R}^m \to \mathcal{L}_k$ be a given tree structure for the k'th tree, mapping a point in feature space to a leaf-index, and $I_{tk} = \{i : q_k(\mathbf{x}_i) = t\}$ then

$$\frac{1}{n}\sum_{i=1}^{n} g_{i,k} f_k(\mathbf{x}_i) + \frac{1}{2} h_{i,k} f_k(\mathbf{x}_i)^2 = \frac{1}{n}\sum_{t\in\mathcal{L}_k}\sum_{i\in I_{t,k}} g_{i,k} f_k(\mathbf{x}_i) + \frac{1}{2} h_{i,k} f_k(\mathbf{x}_i)^2$$

- From here, it is easy to see (differentiate to zero) that a CART tree f_k with structure q_k, minimizing the 2'nd order loss-approximation has leaf-weights $w_{t,k}$

$$\hat{w}_{tk} = -\frac{G_{tk}}{H_{tk}}, \; G_{tk} = \sum_{i\in I_{tk}} g_{ik}, \; H_{tk} = \sum_{i\in I_{tk}} h_{ik}.$$

- Why is all of this important? Due to fast enumeration of split-candidates!

GTB: Greedy recursive binary splitting

Input:

- A training set with derivatives and features $\{\mathbf{x}_i, g_{i,k}, h_{i,k}\}_{i=1}^n$

Do:

1. Initialize the tree with a constant value $\hat{w}$ in a root node:

$$\hat{w} = -\frac{\sum_{i=1}^n g_{i,k}}{\sum_{i=1}^n h_{i,k}}$$

2. Choose a leaf node t and let I_{tk} be the index set of observations falling into node t

For each feature j, compute the reduction in training loss

$$\mathcal{R}_t(j, s_j) = \frac{1}{2n}\left[\frac{\left(\sum_{i\in I_L(j,s:j)} g_{ik}\right)^2}{\sum_{i\in I_L(j,s_j)} h_{ik}} + \frac{\left(\sum_{i\in I_R(j,s_j)} g_{ik}\right)^2}{\sum_{i\in I_R(j,s_j)} h_{ik}} - \frac{\left(\sum_{i\in I_{tk}} g_{ik}\right)^2}{\sum_{i\in I_{tk}} h_{ik}}\right]$$

for different split-points s_j, and where

$I_L(j, s_j) = \{i \in I_{tk} : x_{i,j} \leq s_j\}$ and $I_R(j, s_j) = \{i \in I_{tk} : x_{i,j} > s_j\}$

The values of j and s_j maximizing $\mathcal{R}_t(j, s_j)$ are chosen as the next split, creating two new leaves from the old leaf t.

3. Continue step 2 iteratively, until some threshold on tree-complexity is reached.

Algorithm: 2'nd order GTB

Input:
- A training set $\mathcal{D}_n = \{(x_i, y_i)\}_{i=1}^n$,
- a differentiable loss $l(y, f(x))$,
- a learning rate δ,
- boosting iterations K,
- one or more tree-complexity regularization criteria.

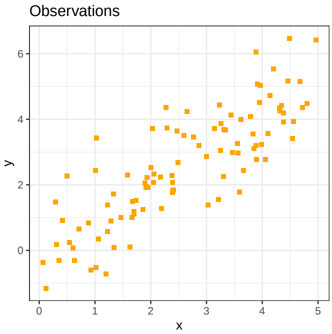
Do:

1. Initialize model with a constant value: $f^{(0)}(\mathbf{x}) = \arg\min_\eta \sum_{i=1}^n l(y_i, \eta)$.
2. **for** $k = 1$ **to** K**:**
   - i) Compute derivatives g_i and h_i for all $i = 1 : n$.
   - ii) Determine q_k by the iterative binary splitting procedure until a regularization criterion is reached.
   - iii) Fit the leaf weights $\mathbf{w}$, given q_k
   - v) Update the model with a scaled tree: $f^{(k)}(\mathbf{x}) = f^{(k-1)}(\mathbf{x}) + \delta f_k(\mathbf{x})$.

   end for
3. **Return** $f^{(K)}(\mathbf{x})$.

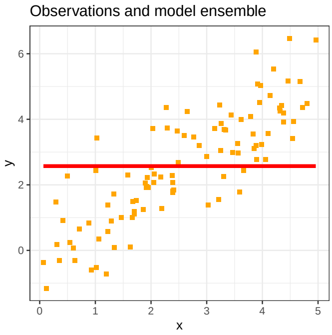
Second order gradient tree boosting

- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.



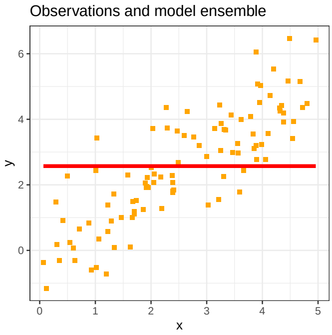
Second order gradient tree boosting

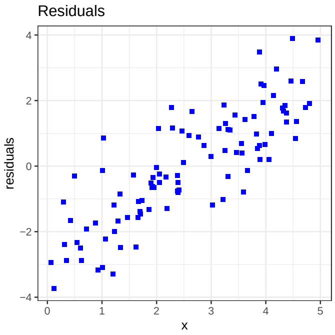
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.



Second order gradient tree boosting

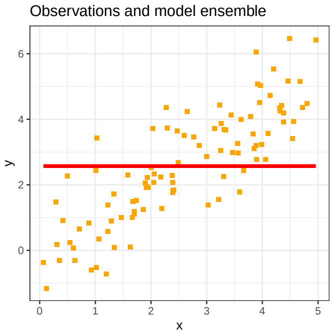
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

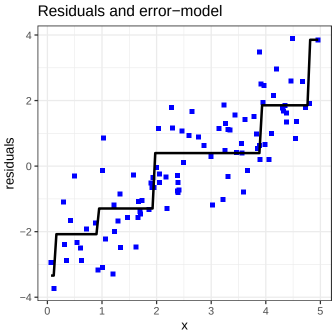




Second order gradient tree boosting

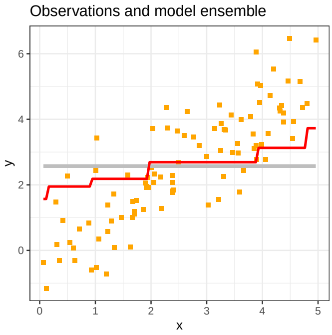
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

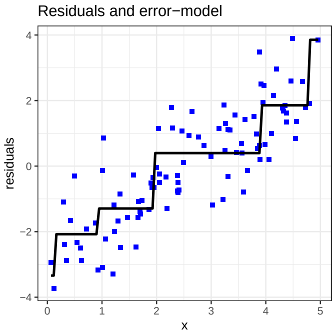




Second order gradient tree boosting

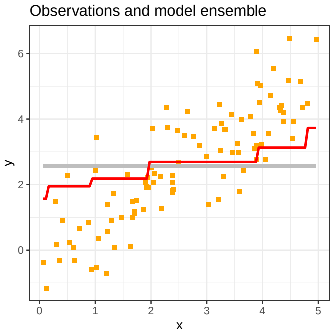
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

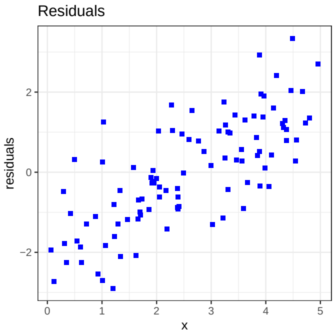




Second order gradient tree boosting

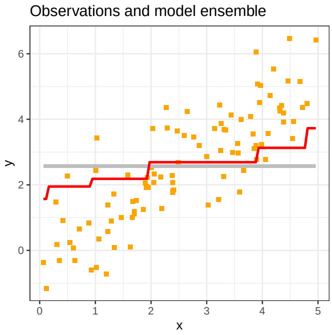
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

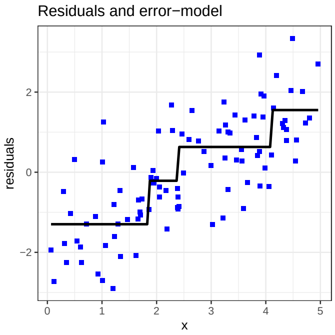




Second order gradient tree boosting

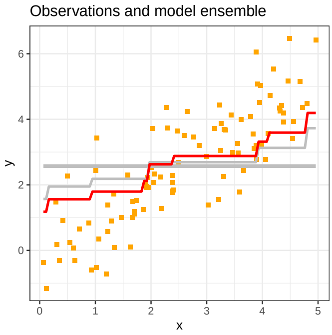
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

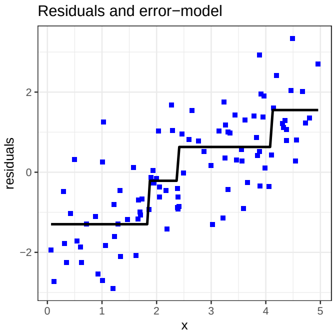




Second order gradient tree boosting

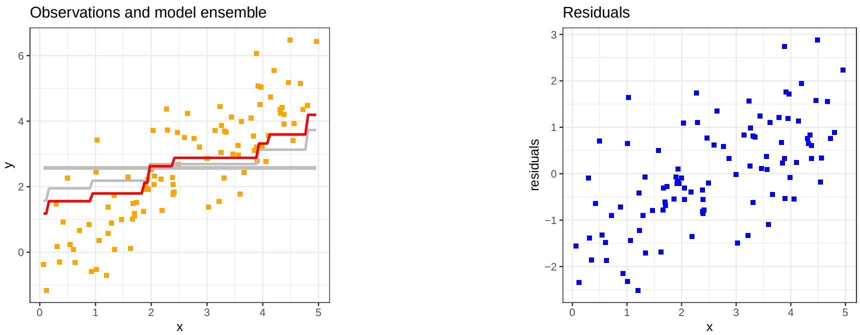
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.





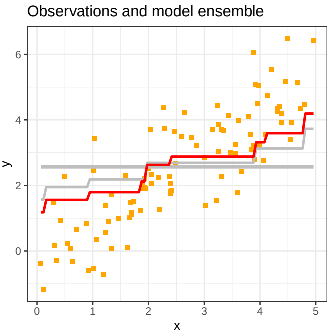
Second order gradient tree boosting

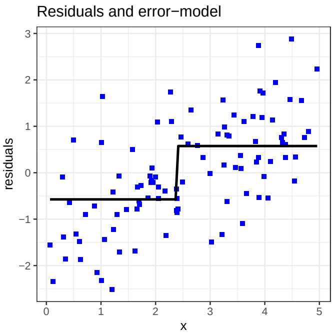
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.



Second order gradient tree boosting

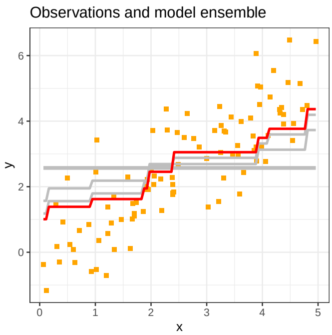
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

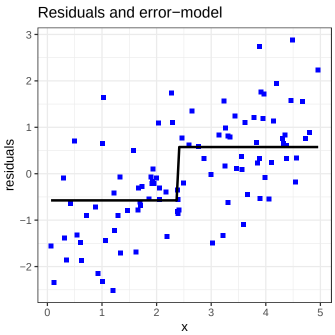




Second order gradient tree boosting

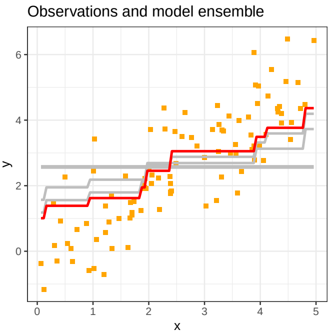
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

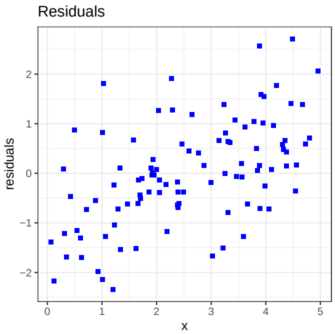




Second order gradient tree boosting

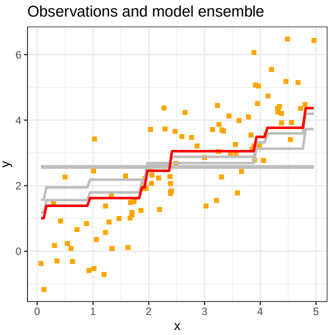
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

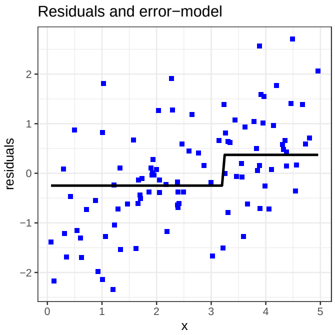




Second order gradient tree boosting

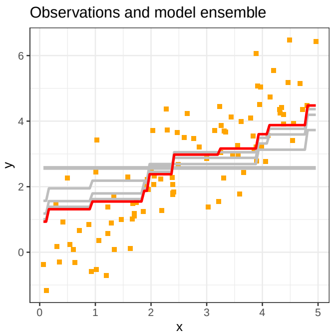
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

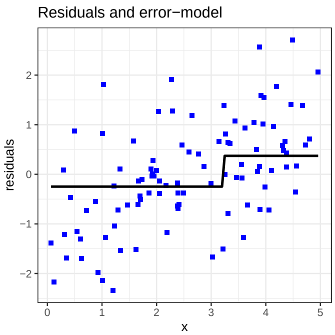




Second order gradient tree boosting

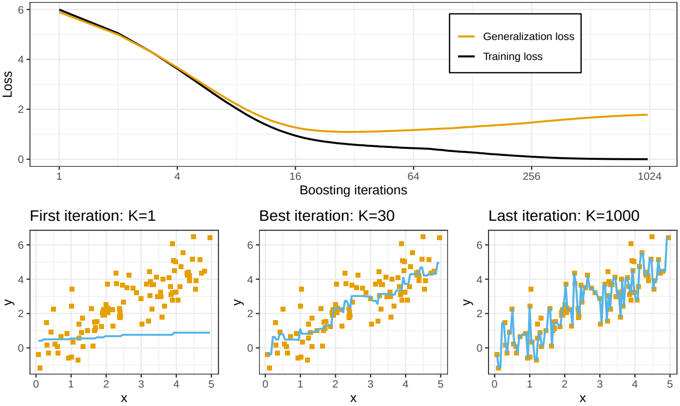
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.





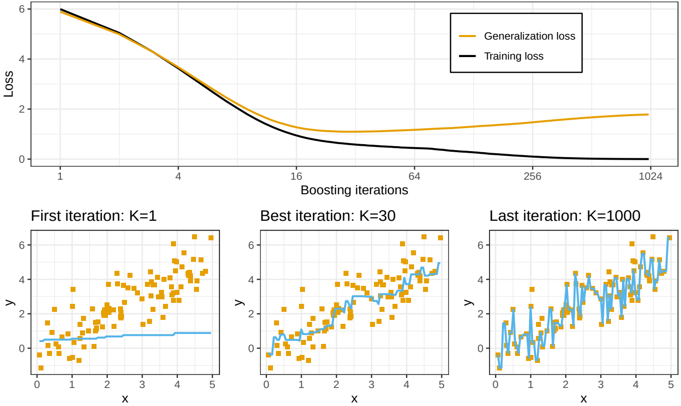
The complexity measure of trees/GTB ensemble?

- What is our measure of complexity (on the x-axis)? Boosting iterations and what else?



GTB: Complexity

- Regularization is needed to control complexity.
  - Number of boosting iterations, maximum depth, maximum number of leaf-nodes, minimum reduction in loss, minimum observations in node, ...



eXtreme Gradient Boosting (XGBoost)

Popular in both academia and industry.

- State-of-the-art machine-learning in terms of predictive power when it comes to modelling structured data.

eXtreme Gradient Boosting (XGBoost)

Popular in both academia and industry.

- State-of-the-art machine-learning in terms of predictive power when it comes to modelling structured data.

- 2'nd order Gradient Tree Boosting.

eXtreme Gradient Boosting (XGBoost)

Popular in both academia and industry.

- State-of-the-art machine-learning in terms of predictive power when it comes to modelling structured data.

- 2'nd order Gradient Tree Boosting.
- Regularization targeting multiple different measures of complexity.
  - Number of boosting iterations, maximum depth, maximum number of leaf-nodes, minimum reduction in loss, minimum observations in node, ...

eXtreme Gradient Boosting (XGBoost)

Popular in both academia and industry.

- State-of-the-art machine-learning in terms of predictive power when it comes to modelling structured data.

- 2'nd order Gradient Tree Boosting.
- Regularization targeting multiple different measures of complexity.
  - Number of boosting iterations, maximum depth, maximum number of leaf-nodes, minimum reduction in loss, minimum observations in node, ...
- As we have seen, enumeration is fast, but sorting beforehand is $n \log n$. xgboost provides an approximate enumeration through a histogram at sketched quantiles.

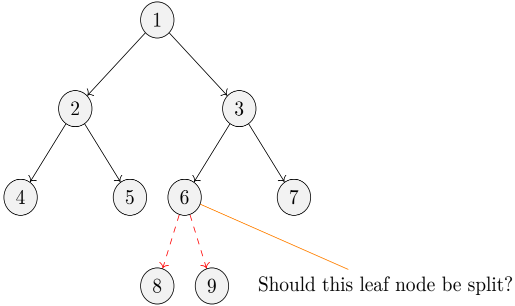
eXtreme Gradient Boosting (XGBoost)

Popular in both academia and industry.

- State-of-the-art machine-learning in terms of predictive power when it comes to modelling structured data.

- 2'nd order Gradient Tree Boosting.
- Regularization targeting multiple different measures of complexity.
  - Number of boosting iterations, maximum depth, maximum number of leaf-nodes, minimum reduction in loss, minimum observations in node, ...
- As we have seen, enumeration is fast, but sorting beforehand is $n \log n$. xgboost provides an approximate enumeration through a histogram at sketched quantiles.
- Implements sparsity awareness, parallelization, custom loss functions and more.

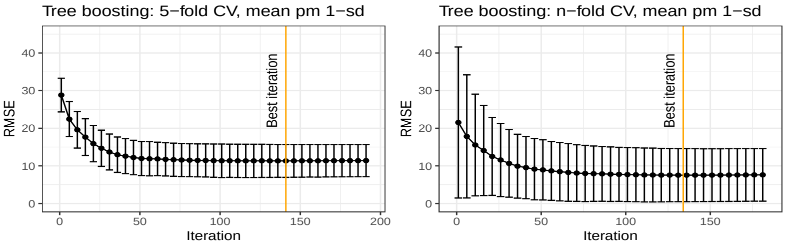
XGBoost regularization



How they regularize:

- Minimum observations in node.
- A minimum reduction in loss.
- L1 and L2 regularization.
- A maximum tree depth.
- A minimum sum of h in node.

Tuning strategies



- k-fold cross validation is implemented in the `xgb.cv()` function.
- The default-parameters are not bad. However, a bare minimum is to find the number of boosting iterations.
- For other settings, the norm is to use 5 or 10-fold CV with either
  - grid search, which has been the norm,
  - random-search. The most brainless (but surprisingly efficient) procedure.
  - or Bayesian optimization.

Honourable GTB mentions

- Other gradient boosting libraries have come later.
  - Most notably are LightGBM, CatBoost and NGBoost.
  - How are they different from XGBoost?

LightGBM

- New stochastic sampling.
- Similarity to Midzuno-Sen's method.
- New approximate histogram algorithm.

Honourable GTB mentions

- Other gradient boosting libraries have come later.
  - Most notably are LightGBM, CatBoost and NGBoost.
  - How are they different from XGBoost?

LightGBM

- New stochastic sampling.
- Similarity to Midzuno-Sen's method.
- New approximate histogram algorithm.

CatBoost

- Focus on categorical features.
- Implements algorithm to avoid certain biases.

Honourable GTB mentions

- Other gradient boosting libraries have come later.
  - Most notably are LightGBM, CatBoost and NGBoost.
  - How are they different from XGBoost?

LightGBM

- New stochastic sampling.
- Similarity to Midzuno-Sen's method.
- New approximate histogram algorithm.

CatBoost

- Focus on categorical features.
- Implements algorithm to avoid certain biases.

NGBoost

- Targets multiple directions simultaneously.
- Assume to know the true DGP-family to employ transforms to natural gradients.

Recap GTB and XGB

- Trees: simple to complex, non-linear, interaction effects, allow sparsity with boosting.
- 2'nd order GTB is employed by xgboost++. Allow for fast enumeration of possible splits.
- How to penalize complexity in trees? XGBoost does this multiple ways.
- Tuning: Learn K as a minimum.
- Other boosting methods exists, but for most applications either XGBoost or LightGBM are preferred (due to strong implementations).

Question

1. In what situations does trees, and boosting ensembles of trees, not necessarily do well?

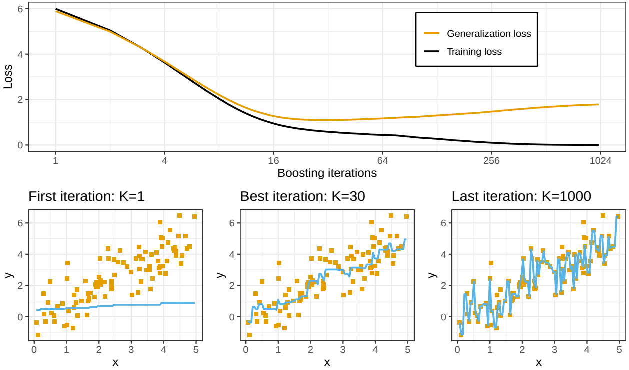
2. Why does the block-diagonal hat-matrix of trees seem to be unimportant (not used for, say `trace`($\mathbf{H}$) to measure the effective degrees of freedom)?

Automatic GTB

- Tuning of hyperparameters is computationally costly.
- This provide motivation for an information theoretic approach.
- Classical information criteria fail for trees and GTB.
- Hence, some work is needed.

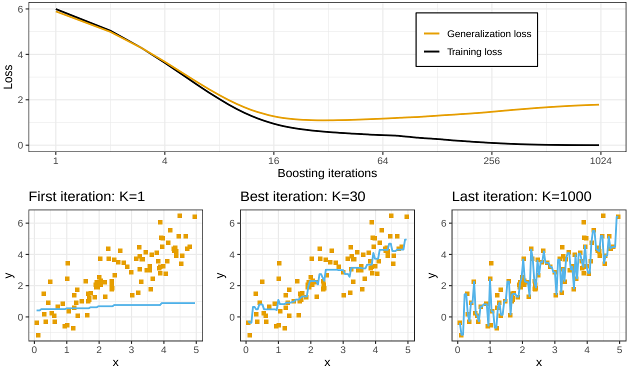
GTB: Complexity

- Regularization is needed to control complexity.



GTB: Complexity

- Regularization is needed to control complexity.
- Highly computationally expensive and require expert knowledge.



The researcher (from earlier) contemplates

The researcher goes home...

He is determined to win that ML-competition!...

The researcher (from earlier) contemplates

The researcher goes home...

He is determined to win that ML-competition!...

- But he only has the 2-gb ram laptop running Windows XP that his job graciously gave him as his every-day workhorse.

The researcher (from earlier) contemplates

The researcher goes home...

He is determined to win that ML-competition!...

- But he only has the 2-gb ram laptop running Windows XP that his job graciously gave him as his every-day workhorse.

So what does he do?

The researcher (from earlier) contemplates

The researcher goes home...

He is determined to win that ML-competition!...

- But he only has the 2-gb ram laptop running Windows XP that his job graciously gave him as his every-day workhorse.

So what does he do?

- Opt 1: Buy a new computer?

The researcher (from earlier) contemplates

The researcher goes home...

He is determined to win that ML-competition!...

- But he only has the 2-gb ram laptop running Windows XP that his job graciously gave him as his every-day workhorse.

So what does he do?

- Opt 1: Buy a new computer?
- Opt 2: Expert knowledge on data and tuning?

The researcher (from earlier) contemplates

The researcher goes home...

He is determined to win that ML-competition!...

- But he only has the 2-gb ram laptop running Windows XP that his job graciously gave him as his every-day workhorse.

So what does he do?

- Opt 1: Buy a new computer?
- Opt 2: Expert knowledge on data and tuning?
- Opt 3: Get lucky?

The researcher (from earlier) contemplates

The researcher goes home...

He is determined to win that ML-competition!...

- But he only has the 2-gb ram laptop running Windows XP that his job graciously gave him as his every-day workhorse.

So what does he do?

- Opt 1: Buy a new computer?
- Opt 2: Expert knowledge on data and tuning?
- Opt 3: Get lucky?
- Opt 4: Rebuild the algorithm to not require tuning?

The researcher (from earlier) contemplates

The researcher goes home...

He is determined to win that ML-competition!...

- But he only has the 2-gb ram laptop running Windows XP that his job graciously gave him as his every-day workhorse.

So what does he do?

- Opt 1: Buy a new computer?
- Opt 2: Expert knowledge on data and tuning?
- Opt 3: Get lucky?
- Opt 4: Rebuild the algorithm to not require tuning?

Plot twist: the researcher is me!

The researcher (from earlier) contemplates

The researcher goes home...

He is determined to win that ML-competition!...

- But he only has the 2-gb ram laptop running Windows XP that his job graciously gave him as his every-day workhorse.

So what does he do?

- Opt 1: Buy a new computer?
- Opt 2: Expert knowledge on data and tuning?
- Opt 3: Get lucky?
- Opt 4: Rebuild the algorithm to not require tuning?

Plot twist: the researcher is me!

- Opt 1: I am a PhD student...

The researcher (from earlier) contemplates

The researcher goes home...

He is determined to win that ML-competition!...

- But he only has the 2-gb ram laptop running Windows XP that his job graciously gave him as his every-day workhorse.

So what does he do?

- Opt 1: Buy a new computer?
- Opt 2: Expert knowledge on data and tuning?
- Opt 3: Get lucky?
- Opt 4: Rebuild the algorithm to not require tuning?

Plot twist: the researcher is me!

- Opt 1: I am a PhD student...
- Opt 2: I am too lazy to be an expert!

The researcher (from earlier) contemplates

The researcher goes home...

He is determined to win that ML-competition!...

- But he only has the 2-gb ram laptop running Windows XP that his job graciously gave him as his every-day workhorse.

So what does he do?

- Opt 1: Buy a new computer?
- Opt 2: Expert knowledge on data and tuning?
- Opt 3: Get lucky?
- Opt 4: Rebuild the algorithm to not require tuning?

Plot twist: the researcher is me!

- Opt 1: I am a PhD student...
- Opt 2: I am too lazy to be an expert!
- Opt 3: I like good expectations.

The researcher (from earlier) contemplates

The researcher goes home...

He is determined to win that ML-competition!...

- But he only has the 2-gb ram laptop running Windows XP that his job graciously gave him as his every-day workhorse.

So what does he do?

- Opt 1: Buy a new computer?
- Opt 2: Expert knowledge on data and tuning?
- Opt 3: Get lucky?
- Opt 4: Rebuild the algorithm to not require tuning?

Plot twist: the researcher is me!

- Opt 1: I am a PhD student...
- Opt 2: I am too lazy to be an expert!
- Opt 3: I like good expectations.
- Opt 4: Hmm...

Revisit the supervised learning problem

The goal is to find f that minimises *generalization error*:

$$\hat{f} = \arg\min_{f} E_{\hat{\theta},\mathbf{x}^0 y^0}\left[l(y^0, f(\mathbf{x}^0; \hat{\theta}))\right]$$

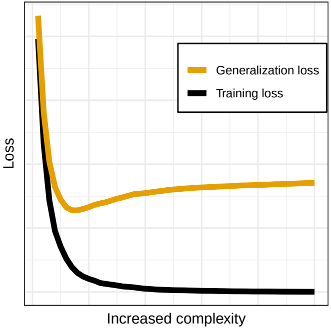
where $(\mathbf{x}^0 y^0)$ are unseen in the training-phase, and therefore independent of $\hat{\theta}$ trained from $(\mathbf{x}, y)$.

Revisit the supervised learning problem

The goal is to find f that minimises *generalization error*:

$$\hat{f} = \arg\min_f E_{\hat{\theta},\mathbf{x}^0 y^0}\left[l(y^0, f(\mathbf{x}^0; \hat{\theta}))\right]$$

where $(\mathbf{x}^0 y^0)$ are unseen in the training-phase, and therefore independent of $\hat{\theta}$ trained from $(\mathbf{x}, y)$.

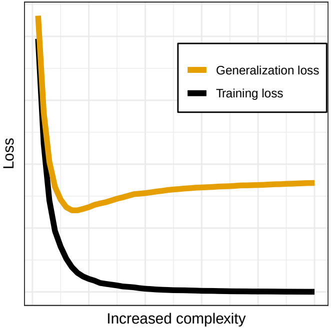


Revisit the supervised learning problem

The goal is to find f that minimises *generalization error*:

$$\hat{f} = \arg\min_f E_{\hat{\theta},\mathbf{x}^0 y^0}\left[l(y^0, f(\mathbf{x}^0;\hat{\theta}))\right]$$

where $(\mathbf{x}^0 y^0)$ are unseen in the training-phase, and therefore independent of $\hat{\theta}$ trained from $(\mathbf{x}, y)$.



- Optimism of the training loss:

$$C(\hat{\theta}) = E\left[l(y^0, f(\mathbf{x}^0;\hat{\theta})) - l(y, f(\mathbf{x};\hat{\theta}))\right]$$

- Often $C(\hat{\theta}) \approx \frac{2}{n}\sum_{i=1}^{n} \mathrm{Cov}(y_i, \hat{y}_i)$

But the generalization loss is unknown...

The main idea:

- Estimate $C(\hat{\theta})$ for trees analytically!

And hope that we may...

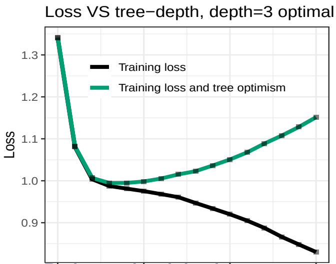
But the generalization loss is unknown...

The main idea:

- Estimate $C(\hat{\theta})$ for trees analytically!

And hope that we may...

1. Adaptively control the complexity of each tree



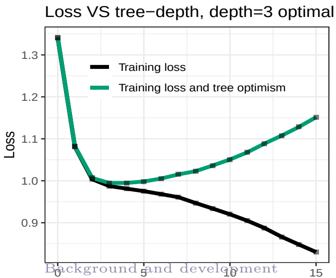
But the generalization loss is unknown...

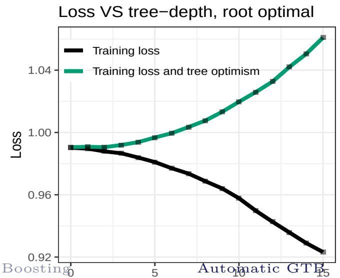
The main idea:

- Estimate $C(\hat{\theta})$ for trees analytically!

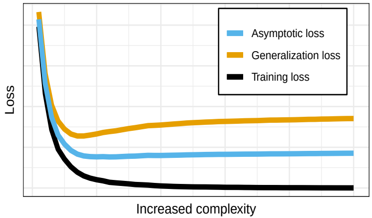
And hope that we may...

1. Adaptively control the complexity of each tree
2. Automatically stop the boosting procedure





A comment on AIC-type criteria

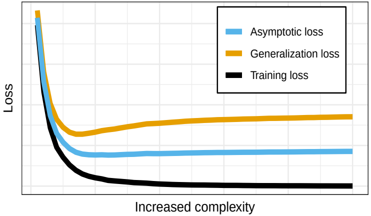


Asymptotic loss and Taylor expansions

- Useful to talk about *asymptotic loss* (blue line in the middle)

$$E\left[l(y, f(\mathbf{x};\theta_0))\right], \quad \lim_{n\to\infty} \hat{\theta} \xrightarrow{P} \theta_0$$

A comment on AIC-type criteria



Asymptotic loss and Taylor expansions

- Useful to talk about *asymptotic loss* (blue line in the middle)

$$E\left[l(y, f(\mathbf{x}; \theta_0))\right], \quad \lim_{n\to\infty} \hat{\theta} \xrightarrow{P} \theta_0$$

- AIC-type criteria result from expectations over two Taylor expansions (Train to Asymptotic and Asymptotic to Generalization) and Slutsky's theorem.

Information criteria and trees

But are AIC-type criteria applicable to trees?

Information criteria and trees

But are AIC-type criteria applicable to trees?

- No! Optimized split-points are not differentiable.

Information criteria and trees

But are AIC-type criteria applicable to trees?

- No! Optimized split-points are not differentiable.
- Okay, but what about a random split-point?

Information criteria and trees

But are AIC-type criteria applicable to trees?

- No! Optimized split-points are not differentiable.
- Okay, but what about a random split-point?

An important observation for gradient tree boosting:

Information criteria and trees

But are AIC-type criteria applicable to trees?

- No! Optimized split-points are not differentiable.
- Okay, but what about a random split-point?

An important observation for gradient tree boosting:

- All complexity is added "locally" by splitting one node at the time.

Information criteria and trees

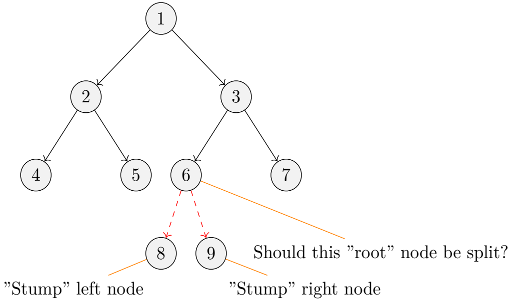
But are AIC-type criteria applicable to trees?

- No! Optimized split-points are not differentiable.
- Okay, but what about a random split-point?

An important observation for gradient tree boosting:

- All complexity is added ”locally” by splitting one node at the time.
- Focus on the ”root” (leaf) versus ”stump” (split of leaf) models.

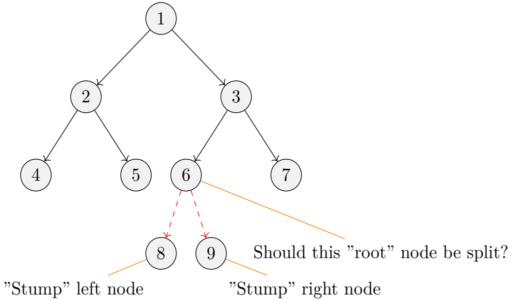
Added complexity at the local level



All added complexity is added at the local level!

- Training data is partitioned into subsets by the tree.

Added complexity at the local level



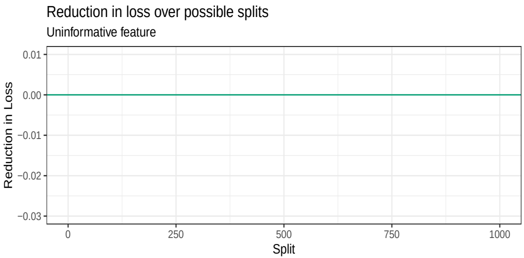
All added complexity is added at the local level!

- Training data is partitioned into subsets by the tree.
- Splitting node 6 only affects optimism of the model applied to the node 6 training subset

Inspection of loss during greedy search

Reduction in loss

$$R(s) = \text{root loss} \;-\; \text{stump loss, at split point } s$$

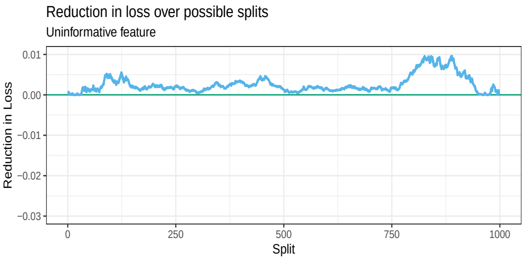


- Asymptotic loss

Inspection of loss during greedy search

Reduction in loss

$$R(s) = \text{root loss} \; - \; \text{stump loss, at split point } s$$

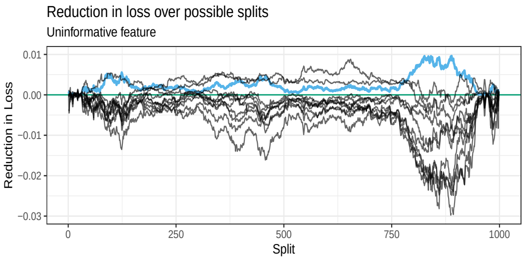


- Asymptotic loss
- Training set

Inspection of loss during greedy search

Reduction in loss

$$R(s) = \text{root loss} \; - \; \text{stump loss, at split point } s$$

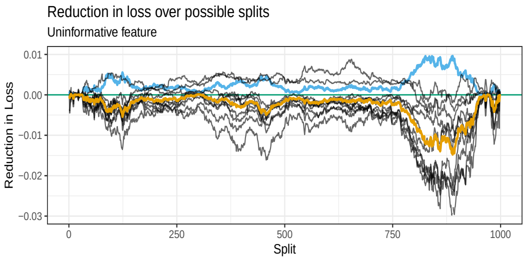


- Asymptotic loss
- Training set
- 10 different test sets

Inspection of loss during greedy search

Reduction in loss

$$R(s) = \text{root loss} \; - \; \text{stump loss, at split point } s$$



- Asymptotic loss
- Training set
- 10 different test sets
- Average of test sets

First main result: Convergence of empirical process

- Donsker's invariance principle allows extension of TIC-type developments to the entire split-profiling procedure simultaneously:

$$R_{tr}(u;\hat{\theta}) - E_{(y^0,x^0)}[R_{te}(u;\hat{\theta})] \xrightarrow[n\to\infty]{D} C_t\pi_t \frac{B(u)^2}{u(1-u)}$$

First main result: Convergence of empirical process

- Donsker's invariance principle allows extension of TIC-type developments to the entire split-profiling procedure simultaneously:

$$R_{tr}(u;\hat{\theta}) - E_{(y^0,x^0)}[R_{te}(u;\hat{\theta})] \xrightarrow[n\to\infty]{D} C_t \pi_t \frac{B(u)^2}{u(1-u)}$$

- π_t is the probability of being in node t.

First main result: Convergence of empirical process

- Donsker's invariance principle allows extension of TIC-type developments to the entire split-profiling procedure simultaneously:

$$R_{tr}(u;\hat{\theta}) - E_{(y^0,x^0)}[R_{te}(u;\hat{\theta})] \xrightarrow[n\to\infty]{D} C_t\pi_t\frac{B(u)^2}{u(1-u)}$$

- π_t is the probability of being in node t.
- C_t is the TIC optimism of training subset in node t, conditional on known structure.

First main result: Convergence of empirical process

- Donsker's invariance principle allows extension of TIC-type developments to the entire split-profiling procedure simultaneously:

$$R_{tr}(u;\hat{\theta}) - E_{(y^0,x^0)}[R_{te}(u;\hat{\theta})] \xrightarrow[n\to\infty]{D} C_t\pi_t\frac{B(u)^2}{u(1-u)}$$

- π_t is the probability of being in node t.
- C_t is the TIC optimism of training subset in node t, conditional on known structure.
- $B(u)$ is a Brownian bridge over time $u \in (0,1)$.

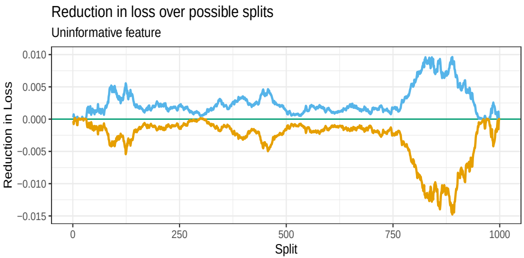
First main result: Convergence of empirical process

- Donsker's invariance principle allows extension of TIC-type developments to the entire split-profiling procedure simultaneously:

$$R_{tr}(u;\hat{\theta}) - E_{(y^0,x^0)}[R_{te}(u;\hat{\theta})] \xrightarrow[n\to\infty]{D} C_t\pi_t\frac{B(u)^2}{u(1-u)}$$

- π_t is the probability of being in node t.
- C_t is the TIC optimism of training subset in node t, conditional on known structure.
- $B(u)$ is a Brownian bridge over time $u \in (0, 1)$.
- "Time" u is defined from possible split-points.

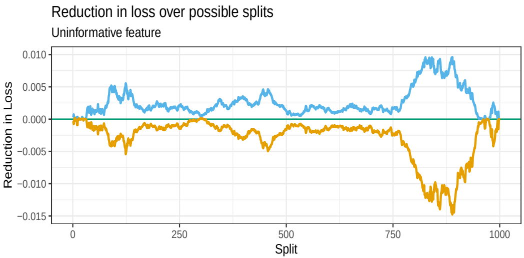
First main result: Expectations



Creating an information criterion:

- We cannot know the exact distance...

First main result: Expectations



Creating an information criterion:

- We cannot know the exact distance...
- But we can know the expected maximum:

$$\tilde{C}_R = E\left[\max\left\{R_{tr}(u) - R^0_{te}(u), 0 < u < 1\right\}\right]$$

$$= -C_t \pi_t E\left[\max\left\{\frac{B(u)^2}{u(1-u)}, 0 < u < 1\right\}\right]$$

Development towards multiple features

Non-continuous features

- Work with the maximum over discrete time-observations on the bridge.

Development towards multiple features

Non-continuous features

- Work with the maximum over discrete time-observations on the bridge.
- Works perfectly (figure later), and holds AIC-type criteria as a special case(!).

Development towards multiple features

Non-continuous features

- Work with the maximum over discrete time-observations on the bridge.
- Works perfectly (figure later), and holds AIC-type criteria as a special case(!).

Multiple features: If independent then...

- Sorted ordering (rankings) are independent, and...

Development towards multiple features

Non-continuous features

- Work with the maximum over discrete time-observations on the bridge.
- Works perfectly (figure later), and holds AIC-type criteria as a special case(!).

Multiple features: If independent then...

- Sorted ordering (rankings) are independent, and...
- The Brownian bridges are independent

Development towards multiple features

Non-continuous features

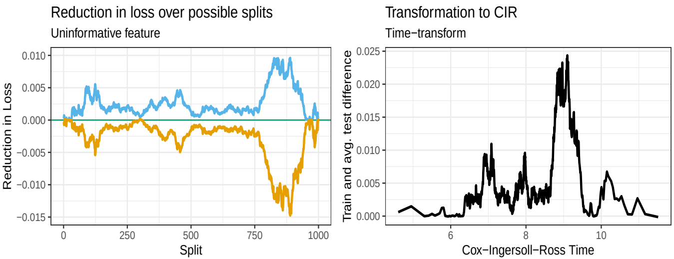
- Work with the maximum over discrete time-observations on the bridge.
- Works perfectly (figure later), and holds AIC-type criteria as a special case(!).

Multiple features: If independent then...

- Sorted ordering (rankings) are independent, and...
- The Brownian bridges are independent
- We can work with the maximum over m independent maximums on Brownian bridges(!)...
- ... and this will bound the dependent case.

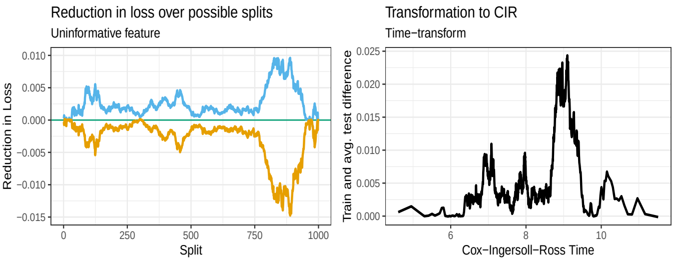
Reduction in loss transform to CIR

Let $\tau = \frac{1}{2}\log\frac{u(1-\epsilon)}{\epsilon(1-u)}$, $\epsilon \to 0$, then $S(\tau(u)) \sim \frac{B(u)^2}{u(1-u)}$ is a CIR.



Reduction in loss transform to CIR

Let $\tau = \frac{1}{2}\log\frac{u(1-\epsilon)}{\epsilon(1-u)}$, $\epsilon \to 0$, then $S(\tau(u)) \sim \frac{B(u)^2}{u(1-u)}$ is a CIR.



Cox-Ingersoll-Ross (CIR)

- The CIR process is defined through the stochastic differential equation

$$dS(\tau) = \alpha(\beta - S(\tau))d\tau + \sigma\sqrt{S(\tau)}dW(\tau).$$

- We have shown that $\alpha = 2$, $\beta = 1$ and $\sigma = 2\sqrt{2}$.

Why CIR?

- The CIR specification is important, because it allows the usage of a different asymptotic theory:

Extreme value theory

Why CIR?

- The CIR specification is important, because it allows the usage of a different asymptotic theory:

Extreme value theory

- The CIR has a gamma stationary distribution.
- Thus, the CIR is in the *maximum domain of attraction* of the Gumbel distribution...
- ... and $\max_\tau S(\tau)$ may be approximated with a Gumbel distribution!

Main result on multiple features

- Including multiple features and discrete split-points, we have:

$$\tilde{C}_R = -C_t \pi_t E \left[\max_j \left\{ \max_{\tau(u_{k,j})} S_j(\tau(u_{k,j})) \right\} \right]$$

Main result on multiple features

- Including multiple features and discrete split-points, we have:

$$\tilde{C}_R = -C_t \pi_t E\left[\max_j \left\{ \max_{\tau(u_{k,j})} S_j(\tau(u_{k,j})) \right\}\right]$$

Evaluation

- The inner maximum is asymptotically Gumbel distributed

$$Y_j = \max_{\tau(u_{k,j})} S_j(\tau(u_{k,j})) \sim Gumbel.$$

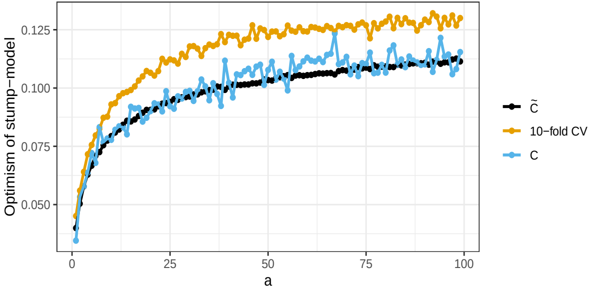
- Assuming independence the outer maximum has distribution

$$P(\max_j Y_j \leq z) = \prod_{j=1}^{m} P(Y_j \leq z) ...$$

- ... and its expectation may be evaluated as

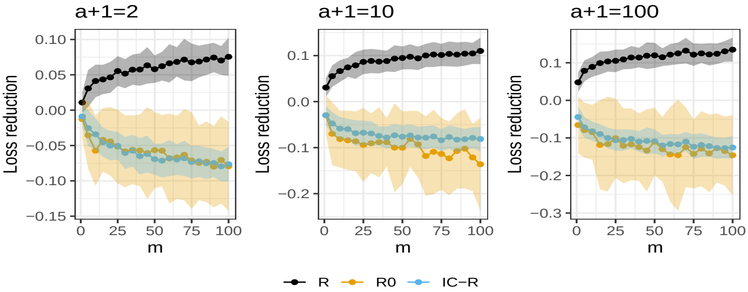
$$E[\max_j Y_j] = \int_0^\infty P(\max_j Y_j > z) dz.$$

Sanity check: Optimism vs increasing number of splits



- a is the number of possible split-points ($n = 100$).
- 10-fold CV uses only 90% of the data, thus more optimism.
- C is average of 1000 test loss.
- $\tilde{C}$ is our information criterion.
- Average values of 1000 different experiments, thus quite robust

Sanity check: Increasing dimensions



- a is the number of possible split-points ($n = 100$).
- m is the number of features.

Going back to the original idea

Our hope was to...

1. Adaptively control the complexity of each tree
2. Automatically stop the boosting procedure

Going back to the original idea

Our hope was to...

1. Adaptively control the complexity of each tree
2. Automatically stop the boosting procedure

What we do: Two inequalities

1. Stop splitting a branch when

$$R_t + \tilde{C}_{R_t} < 0,\ \tilde{C}_{R_t} = -\tilde{C}_t \pi_t E\left[\max_j \left\{ \max_{\tau(u_{k,j})} S_j(\tau(u_{k,j})) \right\}\right] \tag{5}$$

Going back to the original idea

Our hope was to...

1. Adaptively control the complexity of each tree
2. Automatically stop the boosting procedure

What we do: Two inequalities

1. Stop splitting a branch when

$$R_t + \tilde{C}_{R_t} < 0,\ \tilde{C}_{R_t} = -\tilde{C}_t \pi_t E\left[\max_j \left\{ \max_{\tau(u_{k,j})} S_j(\tau(u_{k,j})) \right\}\right] \tag{5}$$

2. Stop the iterative boosting algorithm when

$$\delta(2-\delta)R_t + \delta\tilde{C}_{R_t} < 0 \tag{6}$$

The algorithm

Input:

- A training set $\mathcal{D}_n = \{(x_i, y_i)\}_{i=1}^n$,
- a differentiable loss $l(y, f(x))$,
- a learning rate δ,
- boosting iterations K ,
- one or more tree-complexity regularization criteria.

1. Initialize model with a constant value: $f^{(0)}(\mathbf{x}) = \arg\min_{\eta} \sum_{i=1}^n l(y_i, \eta)$.

2. **for** $k = 1$ to K: **while** the inequality (2) evaluates to **`false`**

   i) Compute derivatives g_i and h_i for all $i = 1 : n$.

   ii) Determine q_k by the iterative binary splitting procedure until

   a regularization criterion is reached. the inequality (1) is **`true`**

   iii) Fit the leaf weights $\mathbf{w}$, given q_k

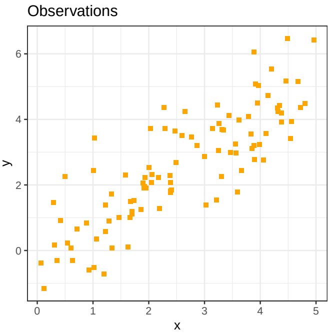
   v) Update the model with a scaled tree: $f^{(k)}(\mathbf{x}) = f^{(k-1)}(\mathbf{x}) + \delta f_k(\mathbf{x})$.

end **for** **while**

3. Output the model:
   - **Return** $f^{(K)}(\mathbf{x})$.

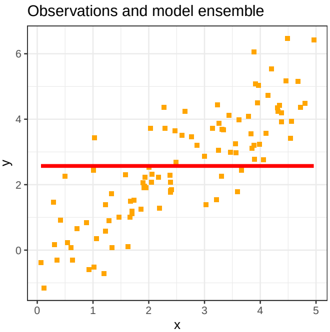
Does it work?

- The tree-boosting animation in the introduction was generated by this algorithm.



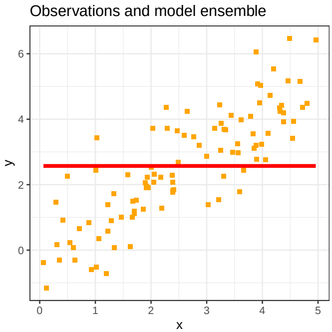
Does it work?

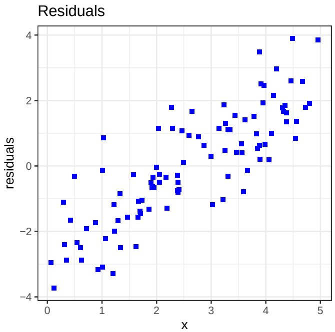
- The tree-boosting animation in the introduction was generated by this algorithm.



Does it work?

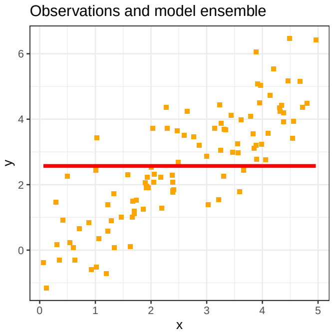
- The tree-boosting animation in the introduction was generated by this algorithm.

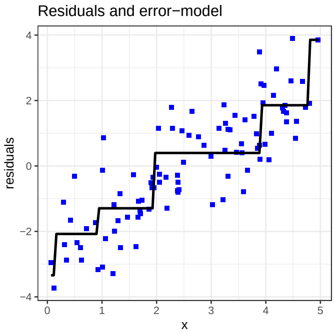




Does it work?

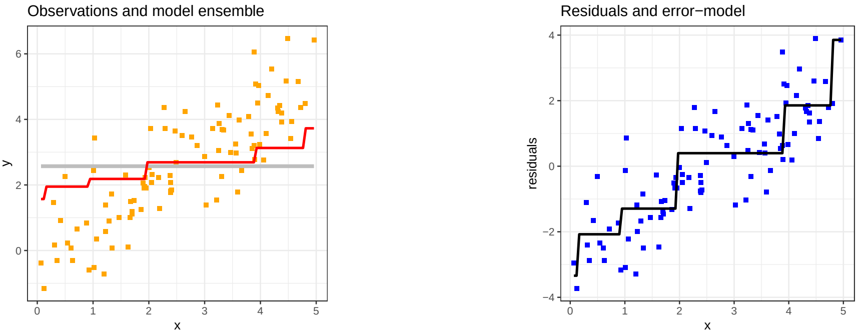
- The tree-boosting animation in the introduction was generated by this algorithm.





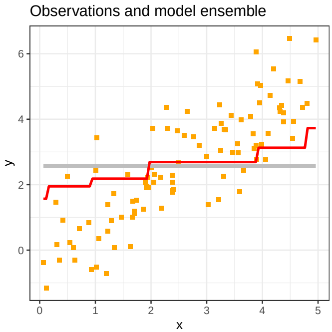
Does it work?

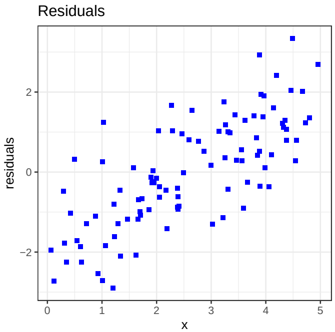
- The tree-boosting animation in the introduction was generated by this algorithm.



Does it work?

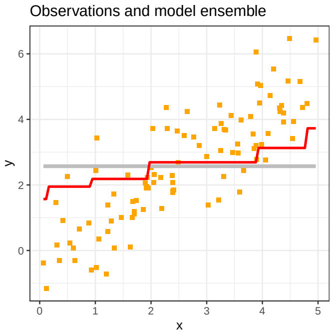
- The tree-boosting animation in the introduction was generated by this algorithm.

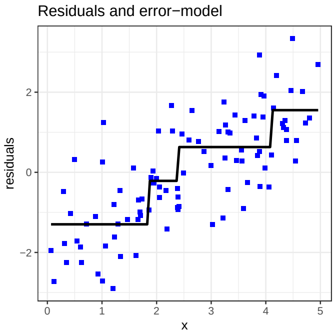




Does it work?

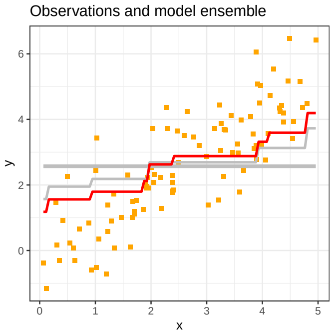
- The tree-boosting animation in the introduction was generated by this algorithm.

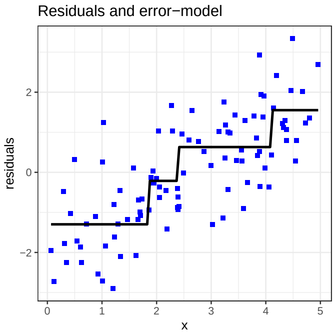




Does it work?

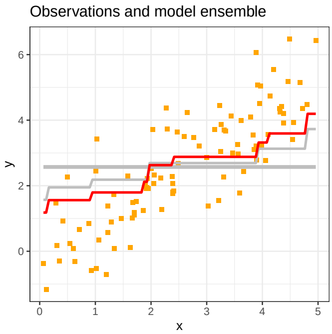
- The tree-boosting animation in the introduction was generated by this algorithm.

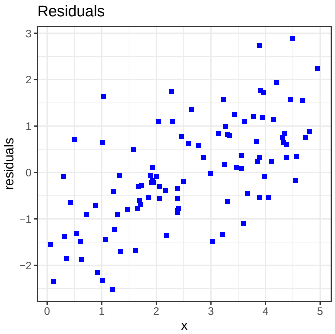




Does it work?

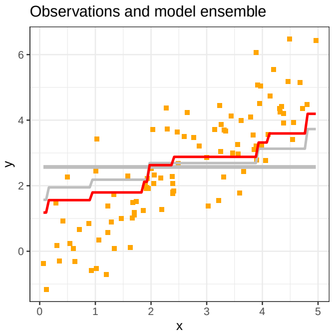
- The tree-boosting animation in the introduction was generated by this algorithm.

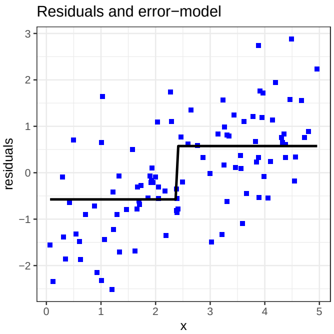




Does it work?

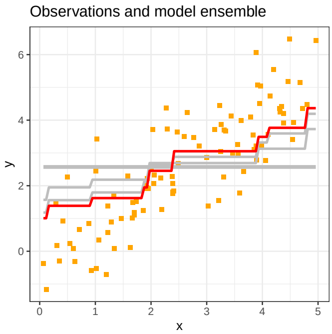
- The tree-boosting animation in the introduction was generated by this algorithm.

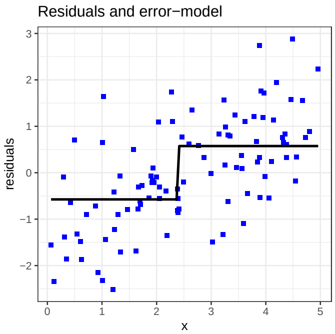




Does it work?

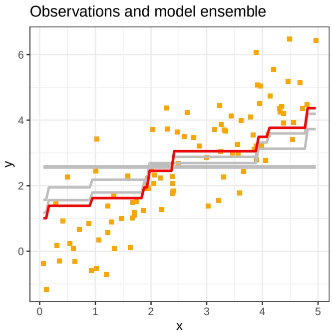
- The tree-boosting animation in the introduction was generated by this algorithm.

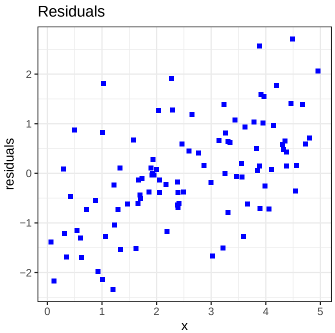




Does it work?

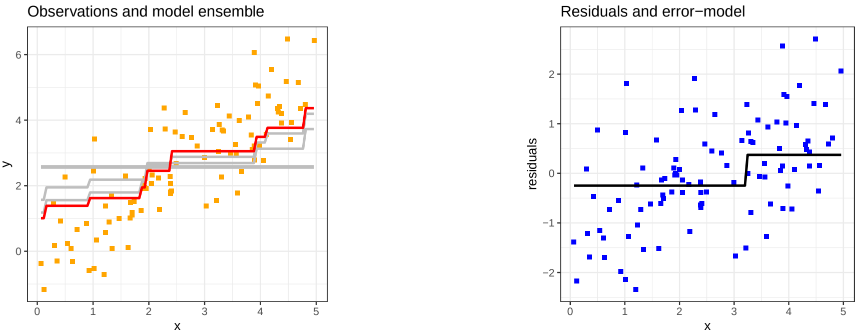
- The tree-boosting animation in the introduction was generated by this algorithm.





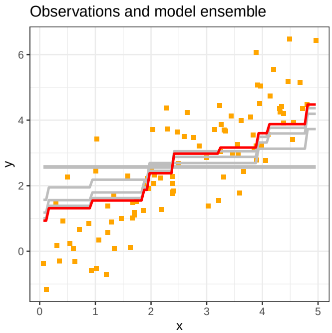
Does it work?

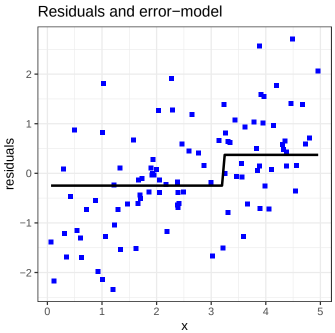
- The tree-boosting animation in the introduction was generated by this algorithm.



Does it work?

- The tree-boosting animation in the introduction was generated by this algorithm.





Does it work: Simulation experiment $p >> n$

- Simulation experiment: $n = 1000$ independent observations.

  Case 1: $m = 1$ informative feature. benchmark is un-regularized linear regression.

  Case 2: $m = 10000$, one informative feature and 9999 independent noise features. Benchmark is Lasso regression.

  Case 3: $m = 10000$ with dependent features that have different levels of information. Benchmark is Ridge regression.

Method	Case 1 ($m = 1$)			Case 2 ($m = 10000$)			Case 3 ($m = 10000$)		
	Loss	K	CPU-Time	Loss	K	CPU-Time	Loss	K	CPU-Time
linear model	0.977		0.0293	1.01		16	1.07		43
`aGTBoost`	1.01	365	0.162	1.05	294	723	1.04	348	821
`xgboost`: cv	1.11	275	4.28	1.07	357	3447	1.08	370	3908
`xgboost`: val	1.16	311	0.507	1.16	371	258	1.09	249	171

ISLR and ESL datasets

- Comparisons on real data
- Every dataset randomly split into training and test datasets 100 different ways
- Average test scores (relative to XGB) and standard deviations (parenthesis)

Dataset	xgboost	aGTBoost	random forest
Boston	1 (0.173)	1.02 (0.144)	0.877 (0.15)
Ozone	1 (0.202)	0.816 (0.2)	0.675 (0.183)
Auto	1 (0.188)	0.99 (0.119)	0.895 (0.134)
Carseats	1 (0.112)	0.956 (0.126)	1.16 (0.141)
College	1 (0.818)	1.27 (0.917)	1.07 (0.909)
Hitters	1 (0.323)	0.977 (0.366)	0.798 (0.311)
Wage	1 (1.01)	1.39 (1.64)	82.5 (21.4)
Caravan	1 (0.052)	0.983 (0.0491)	1.3 (0.167)
Default	1 (0.0803)	0.926 (0.0675)	2.82 (0.508)
OJ	1 (0.0705)	0.966 (0.0541)	1.17 (0.183)
Smarket	1 (0.00401)	0.997 (0.00311)	1.04 (0.0163)
Weekly	1 (0.00759)	0.992 (0.00829)	1.02 (0.0195)

Computational considerations

In general...

- Let k-fold cross validation be used to determine the tuning for a standard tree-boosting implementation using ”early-stopping”.
- Consider p hyperparameters, each having r candidate values.
- Then our implementation is approximately $k \times r^p + 1$ times faster.

Computational considerations

In general...

- Let k-fold cross validation be used to determine the tuning for a standard tree-boosting implementation using "early-stopping".
- Consider p hyperparameters, each having r candidate values.
- Then our implementation is approximately $k \times r^p + 1$ times faster.

A comparison with XGB

- On the OJ dataset (1070×18 classification), our implementation took 1.46 seconds to train.

Computational considerations

In general...

- Let k-fold cross validation be used to determine the tuning for a standard tree-boosting implementation using "early-stopping".
- Consider p hyperparameters, each having r candidate values.
- Then our implementation is approximately $k \times r^p + 1$ times faster.

A comparison with XGB

- On the OJ dataset (1070×18 classification), our implementation took 1.46 seconds to train.
- Using a 30% validation set, XGB took 1.3 seconds

Computational considerations

In general...

- Let k-fold cross validation be used to determine the tuning for a standard tree-boosting implementation using ”early-stopping”.
- Consider p hyperparameters, each having r candidate values.
- Then our implementation is approximately $k \times r^p + 1$ times faster.

A comparison with XGB

- On the OJ dataset (1070×18 classification), our implementation took 1.46 seconds to train.
- Using a 30% validation set, XGB took 1.3 seconds
- 8.55 seconds using 10-fold CV: the number of boosting iterations

Computational considerations

In general...

- Let k-fold cross validation be used to determine the tuning for a standard tree-boosting implementation using ”early-stopping”.
- Consider p hyperparameters, each having r candidate values.
- Then our implementation is approximately $k \times r^p + 1$ times faster.

A comparison with XGB

- On the OJ dataset (1070×18 classification), our implementation took 1.46 seconds to train.
- Using a 30% validation set, XGB took 1.3 seconds
- 8.55 seconds using 10-fold CV: the number of boosting iterations
- About 3.2 minutes to learn one additional hyperparameter

Computational considerations

In general...

- Let k-fold cross validation be used to determine the tuning for a standard tree-boosting implementation using "early-stopping".
- Consider p hyperparameters, each having r candidate values.
- Then our implementation is approximately $k \times r^p + 1$ times faster.

A comparison with XGB

- On the OJ dataset (1070×18 classification), our implementation took 1.46 seconds to train.
- Using a 30% validation set, XGB took 1.3 seconds
- 8.55 seconds using 10-fold CV: the number of boosting iterations
- About 3.2 minutes to learn one additional hyperparameter
- About 33 minutes on yet another additional hyperparameter

The aGTBoost R-package

- Implemented in `C++`, depends upon `Eigen` for linear algebra.
- Depends on `Rcpp` for the `R`-package.
- Installation possible from CRAN:

```
install.packages("agtboost")
```

- Source and development version available on Github:
  `https://github.com/Blunde1/agtboost`

Further developments

Theoretical developments:

- L1-L2 regularization.
- Stochastic sampling of both rows and columns.
- Fast histogram algorithm.

Further developments

Theoretical developments:

- L1-L2 regularization.
- Stochastic sampling of both rows and columns.
- Fast histogram algorithm.

Computational developments:

- Code optimization.
- Input sparse design matrix (Eigen sparsity).
- Parallelization (OpenMP).

Further developments

Theoretical developments:

- L1-L2 regularization.
- Stochastic sampling of both rows and columns.
- Fast histogram algorithm.

Computational developments:

- Code optimization.
- Input sparse design matrix (Eigen sparsity).
- Parallelization (OpenMP).

- Goal: ML-winning off-the-shelf algorithm.

Full lecture recap

- Boosting targets the supervised learning objective directly.
- Boosting is a technique to combine weak-learners into strong.
- AdaBoost relies crucially on its loss function.
- Gradient boosting fits base-learners to derivative information.
- There is a connection between boosting and sparsity: But dependent on base-learner.
- Trees are *almost* perfect weak-learners.
- Regularizing complexity for trees is hard. One of the reasons why all GTB implementations have many so hyperparameters.
- Hope to soon have both implementation and theory that make tuning redundant.

Questions?