An information criterion for automatic gradient tree boosting

Berent Ånund Strømnes Lunde[1]
Tore Selland Kleppe[1] Hans Julius Skaug[2]

[1]Department of Mathematics and Physics
University of Stavanger

[2]Department of Mathematics
University of Bergen

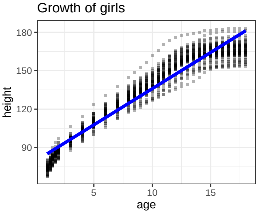
Alberta Statistics and Probability Seminar
University of Alberta, Canada
30th September 2020

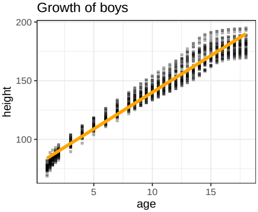
Outline

Background outline

- Motivate the boosting technique.
- Understand why gradient tree boosting works.
- Discuss computational deficiencies with Cross Validation...
- ... and thus motivate an information theoretic approach.

Question 1: Linear regression

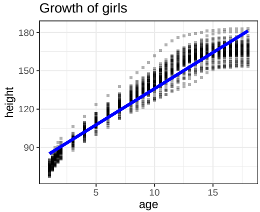


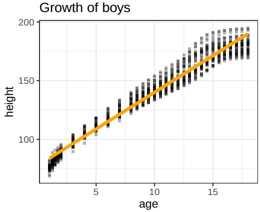


Researcher asks...

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





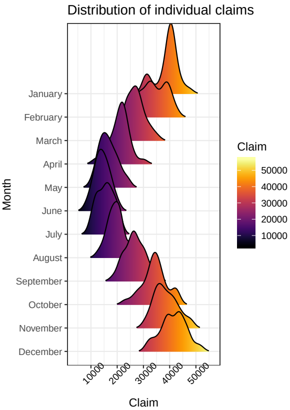
Researcher asks...

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

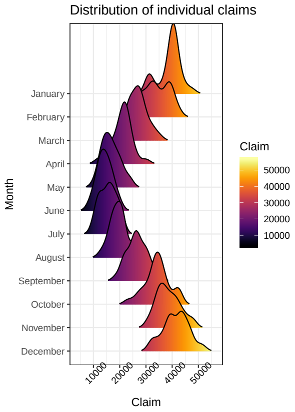
Question 2: Generalized linear models



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:

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

The loss often correspond to a nll.

Supervised learning

- The above problems may be framed as supervised learning:

Supervised learning

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

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

The loss often correspond to a nll.

User restricted f, is it...

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

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...

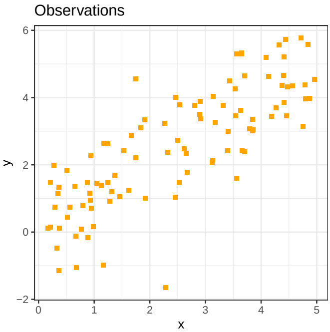
Try gradient boosting?

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

Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

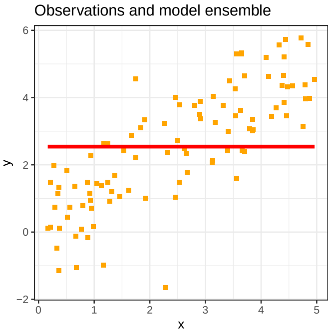
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.



Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

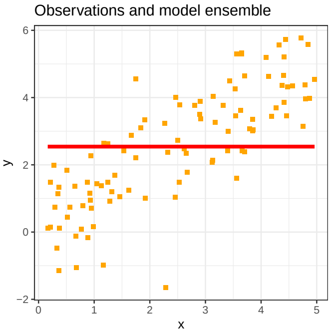
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

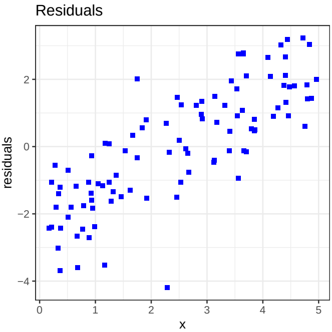


Gradient boosting, what is going on behind the scenes?

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.

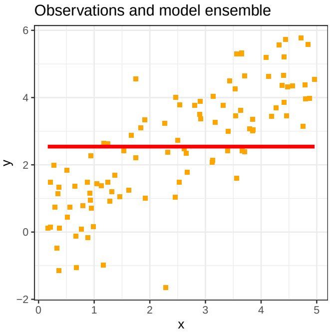


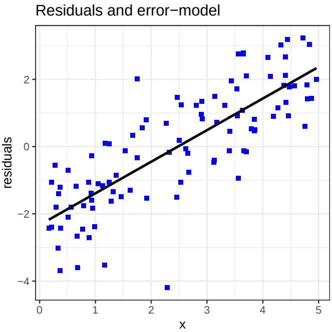


Gradient boosting, what is going on behind the scenes?

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.

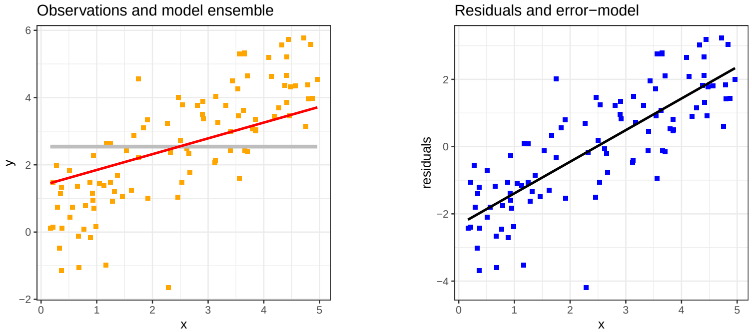




Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

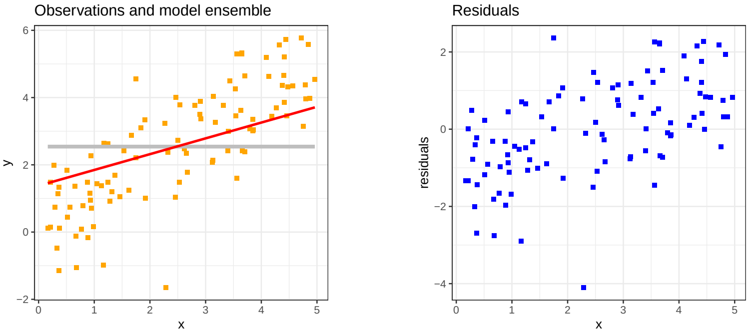
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.



Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

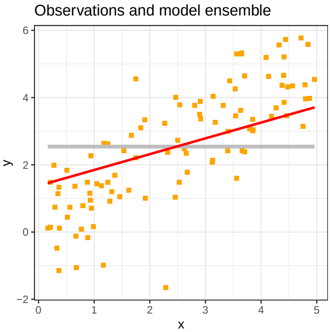
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

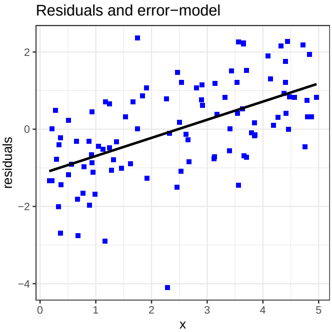


Gradient boosting, what is going on behind the scenes?

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.

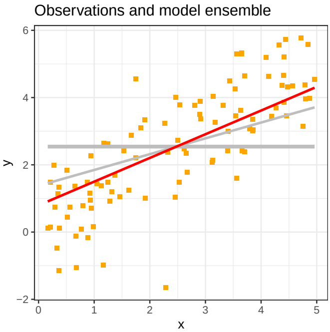


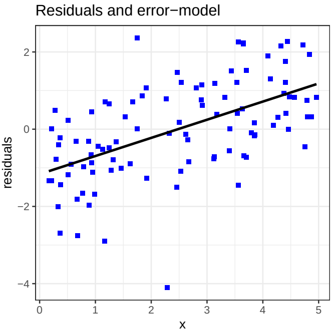


Gradient boosting, what is going on behind the scenes?

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.

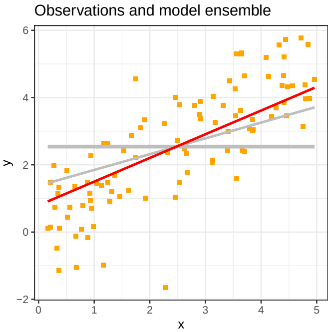


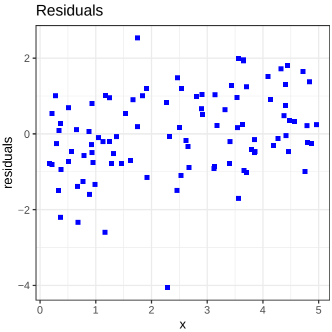


Gradient boosting, what is going on behind the scenes?

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.

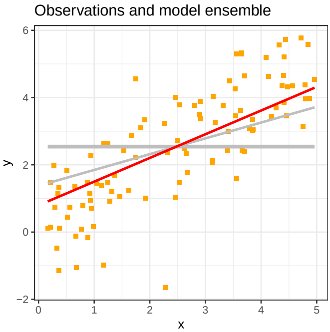


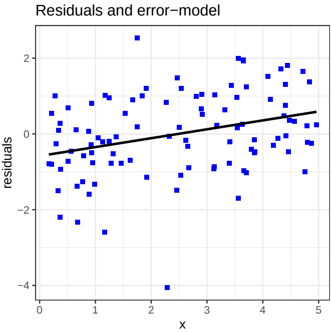


Gradient boosting, what is going on behind the scenes?

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.

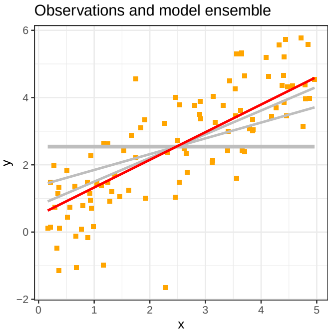


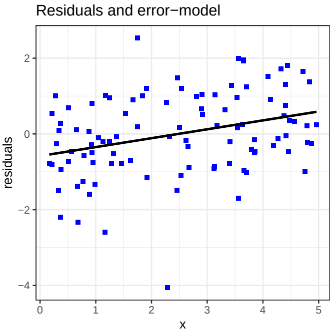


Gradient boosting, what is going on behind the scenes?

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.

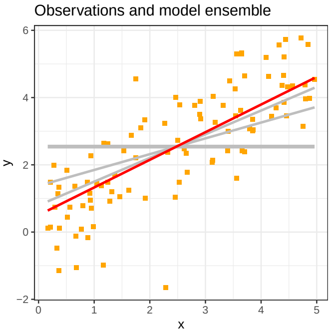


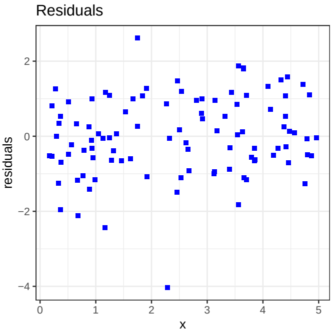


Gradient boosting, what is going on behind the scenes?

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.

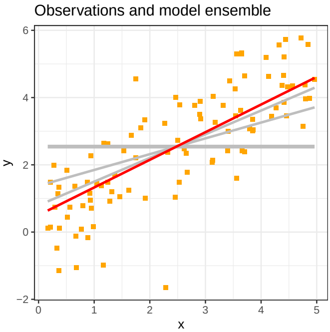


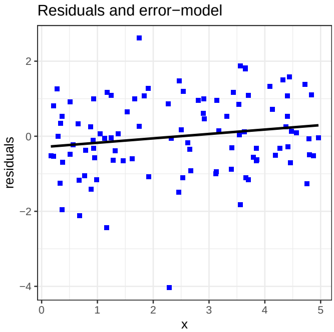


Gradient boosting, what is going on behind the scenes?

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.

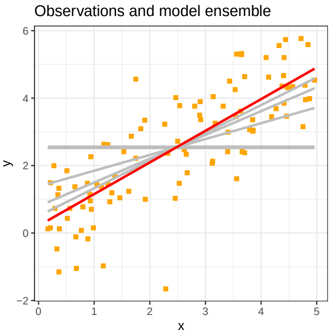


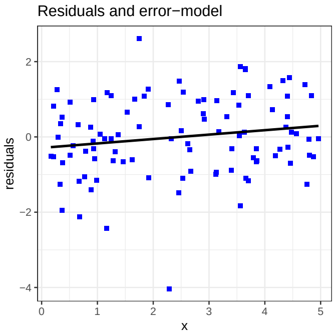


Gradient boosting, what is going on behind the scenes?

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.

It can be generalized beyond MSE:

- 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.

Trees: where boosting gets interesting

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

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

We used a linear model for "base learners" f_k:

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

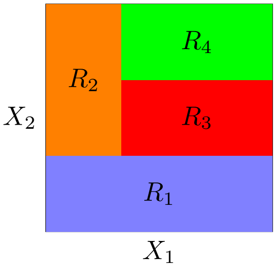
- More interesting with non-linear learning procedures for f_k
- But needs to retain the possibility of a simple (sparse) model.
- We need something that can be non-linear but adapts this to data!
- Trees: complexity from the simple mean or "tree-stumps" to potentially a complete fit to training data.

The tree-learning prcedure: 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?

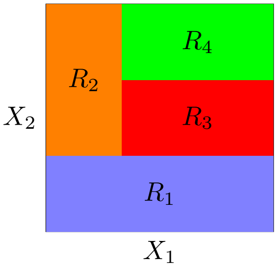


The tree-learning prcedure: recursive binary splitting

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

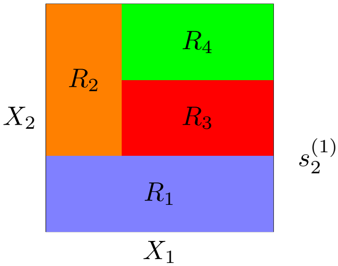
1. Start with a constant prediction for all of feature space
2. Split a leaf-node into two regions (on feature j and split-point s_j chosen by some criteria).
3. Continue step 2 recursively on all leaves.

The tree-learning prcedure: 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

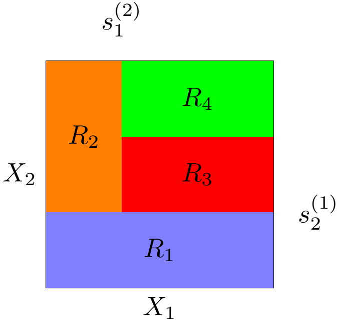
1. Start with a constant prediction for all of feature space
2. Split a leaf-node into two regions (on feature j and split-point s_j chosen by some criteria).
3. Continue step 2 recursively on all leaves.

The tree-learning prcedure: 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

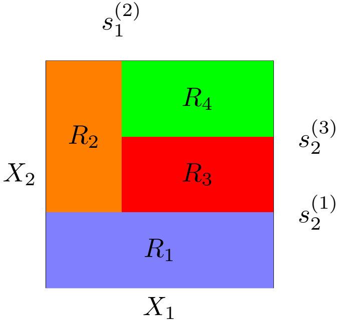
1. Start with a constant prediction for all of feature space
2. Split a leaf-node into two regions (on feature j and split-point s_j chosen by some criteria).
3. Continue step 2 recursively on all leaves.

The tree-learning prcedure: 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?



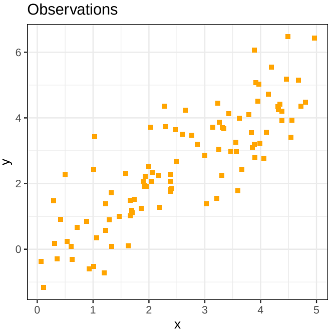
Recursive binary splitting

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.

Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

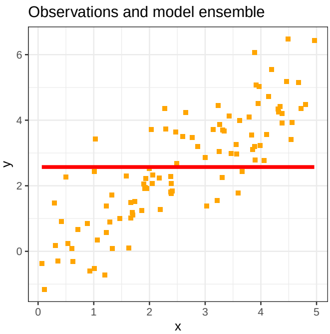
$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \text{ and } h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$



Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

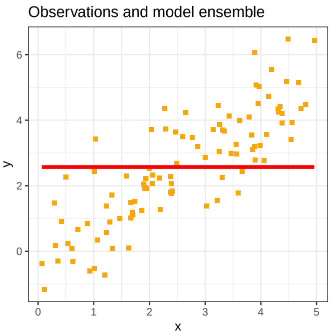
$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i) \Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \text{ and } h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i) \Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$

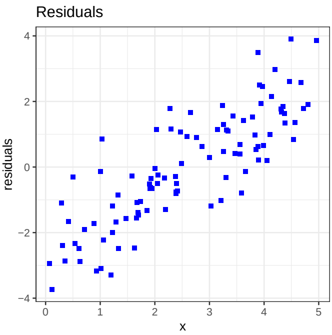


Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \text{ and } h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$

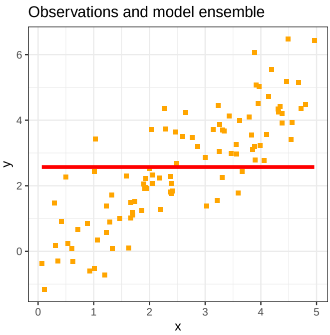


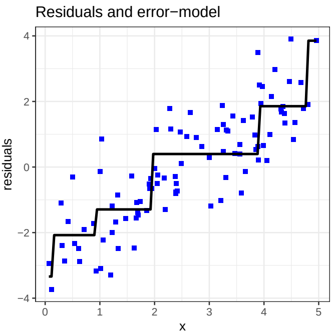


Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \text{ and } h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$

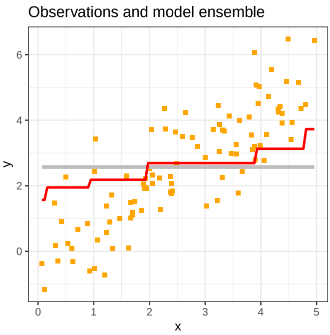


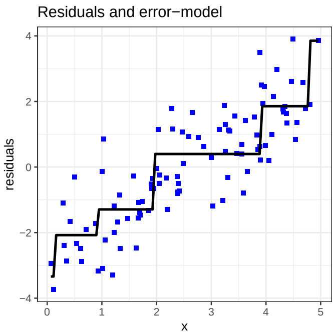


Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \quad \text{and} \quad h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$

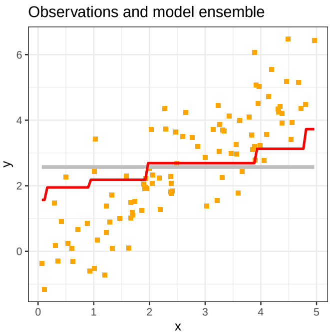


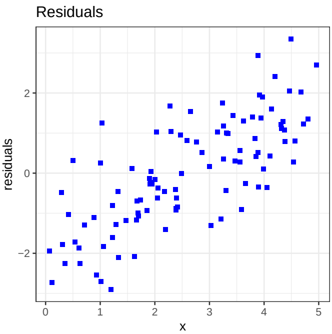


Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \quad \text{and} \quad h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$

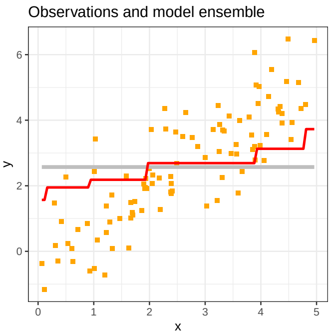


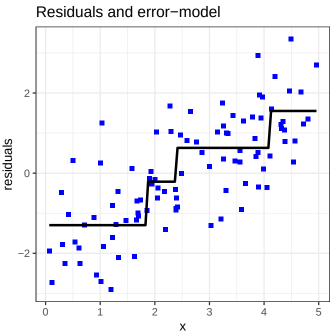


Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i) \Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \quad \text{and} \quad h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i) \Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} .$$

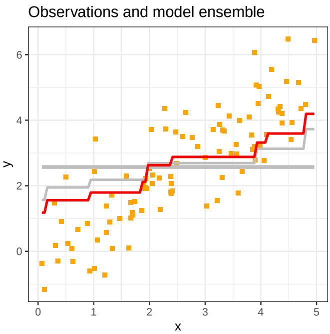


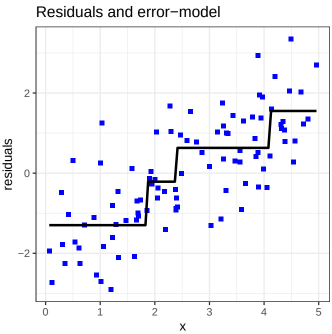


Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \quad \text{and} \quad h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$

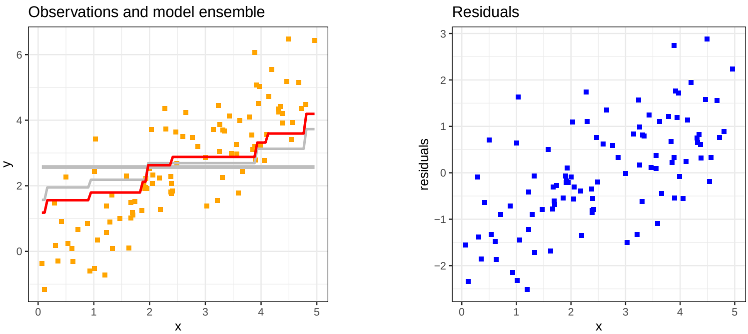




Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

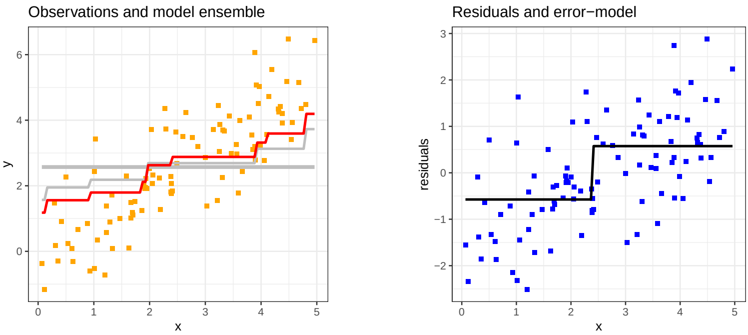
$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \quad \text{and} \quad h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$



Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

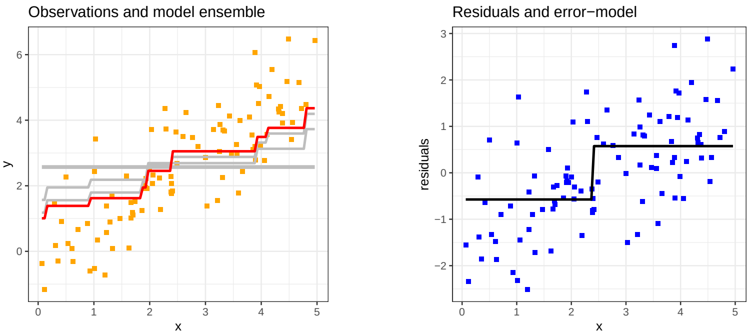
$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \quad \text{and} \quad h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$



Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

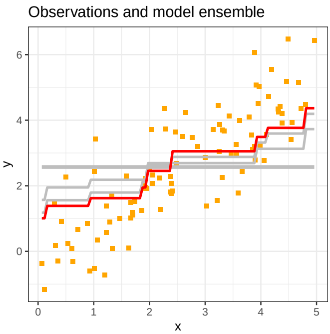
$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \quad \text{and} \quad h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$

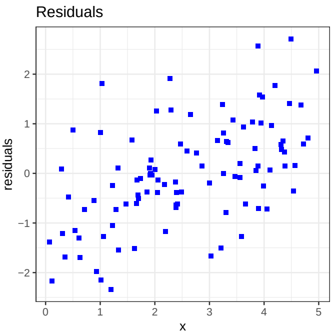


Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \quad \text{and} \quad h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$

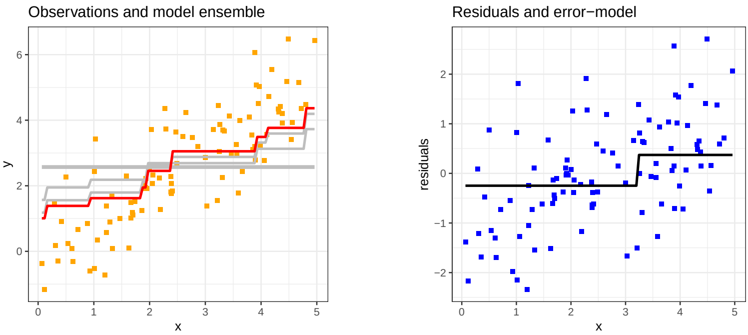




Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

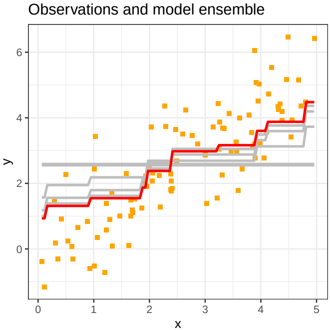
$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \quad \text{and} \quad h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$

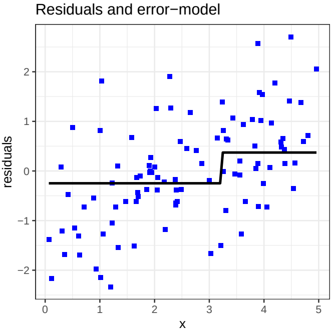


Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\bigg|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \quad \text{and } h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\bigg|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$





Second order gradient tree boosting: Complexity

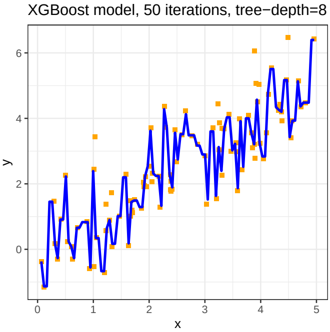
This was too easy!

- How do we choose the complexity of the trees?
- And how many boosting iterations?

Second order gradient tree boosting: Complexity

This was too easy!

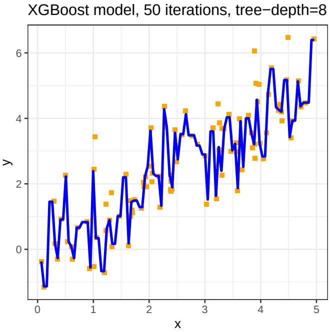
- How do we choose the complexity of the trees?
- And how many boosting iterations?



Second order gradient tree boosting: Complexity

This was too easy!

- How do we choose the complexity of the trees?
- And how many boosting iterations?



Regularization

- Choose a maximum depth?
- A maximum number of leaf-nodes?
- A minimum observations in node?
- A minimum reduction in loss when splitting?
- A set number of boosting iterations?

The researcher contemplates

The researcher goes home...

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

The researcher 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 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 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 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 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 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 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 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 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 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 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...

Information outline

- Information criteria in the context of supervised learning.
- Why AIC-type criteria fails for trees (and necessarily GTB).
- The ideas behind our information criteria for trees.
- Some figures for visual validation.

Revisit the supervised learning problem

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

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

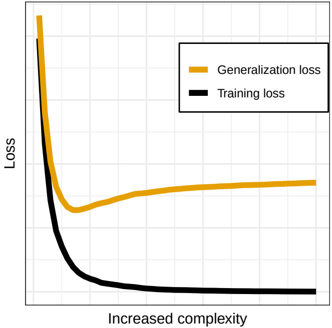
where $(\mathbf{x}^0 y^0)$ are unseen in the training-phase, and therefore independent of $\hat{\theta}$ trained from $(\mathbf{x}, y)$.

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 $(\mathbf{x}, y)$.

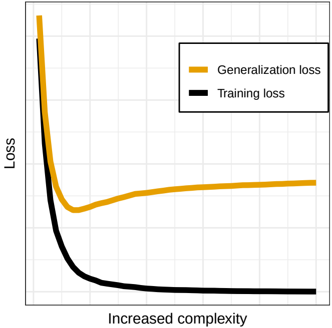


Revisit the supervised learning problem

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...

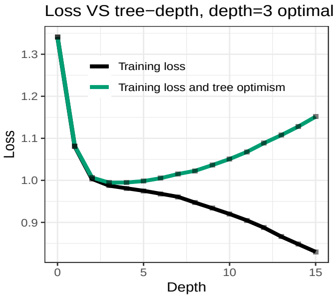
But the generalization loss is unknown...

The main idea:

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

And hope that we may...

1. Adaptively control the complexity of each tree



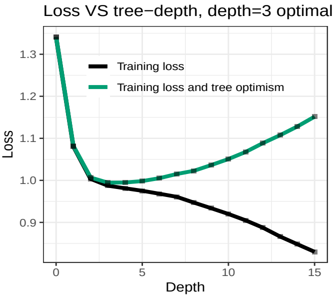
But the generalization loss is unknown...

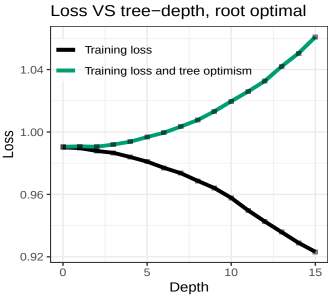
The main idea:

- 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





B... on crite

Information criteria: Akaike and beyond...

The poor researcher has no processing power...

- An analytic, not a data-driven approach, is needed!

Information criteria: Akaike and beyond...

The poor researcher has no processing power...

- An analytic, not a data-driven approach, is needed!

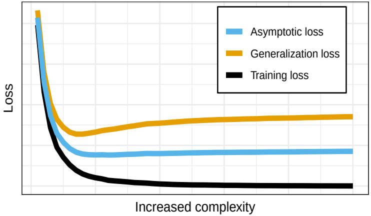
A brief history of (some) information criteria

- [Akaike, 1974] AIC: $C = p$ for NLL. Assumptions on true model
- [Takeuchi, 1976] TIC: $C = \mathrm{tr}(QH^{-1})$ also for NLL, but no assumption on the true model
- [Murata et al., 1994] NIC: $C = \mathrm{tr}(QH^{-1})$ also for differentiable loss

$$H = E\left[\nabla^2_{\theta_0} l(y, f(\mathbf{x};\theta_0))\right]$$

$$Q = E\left[\left(\nabla_{\theta_0} l(y, f(\mathbf{x};\theta_0))\right)\left(\nabla_{\theta_0} l(y, f(\mathbf{x};\theta_0))\right)^{\intercal}\right]$$

A comment on AIC-type criteria

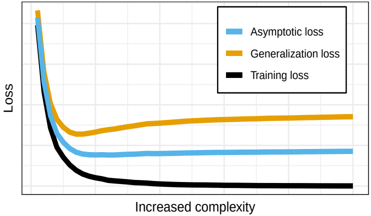


Asymptotic loss and Taylor expansions

- 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)

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

- 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?

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

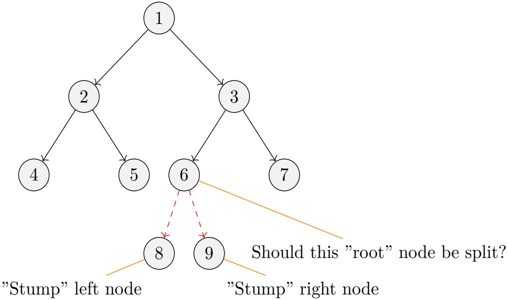
But are AIC-type criteria applicable to trees?

- 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.

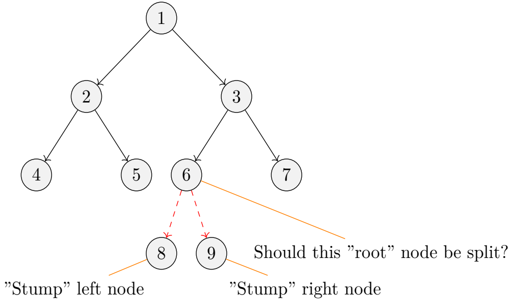
Added complexity at the local level



All added complexity is added at the local level!

- Training data is partitioned into subsets by the tree.

Added complexity at the local level



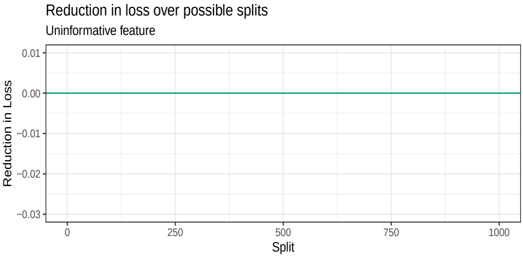
All added complexity is added at the local level!

- 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

$$R(s) = -\frac{1}{2n}\left[\frac{\left(\sum_{i\in I_t} g_i\right)^2}{\sum_{i\in I_t} h_i} - \left(\frac{\left(\sum_{i\in I_l} g_i\right)^2}{\sum_{i\in I_l} h_i} + \frac{\left(\sum_{i\in I_r} g_i\right)^2}{\sum_{i\in I_r} h_i}\right)\right]$$

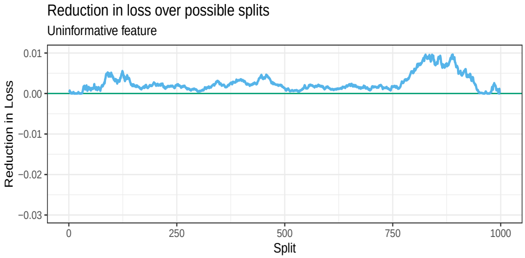


- Asymptotic loss

Inspection of loss during greedy search

Reduction in loss

$$R(s) = -\frac{1}{2n}\left[\frac{\left(\sum_{i\in I_t} g_i\right)^2}{\sum_{i\in I_t} h_i} - \left(\frac{\left(\sum_{i\in I_l} g_i\right)^2}{\sum_{i\in I_l} h_i} + \frac{\left(\sum_{i\in I_r} g_i\right)^2}{\sum_{i\in I_r} h_i}\right)\right]$$

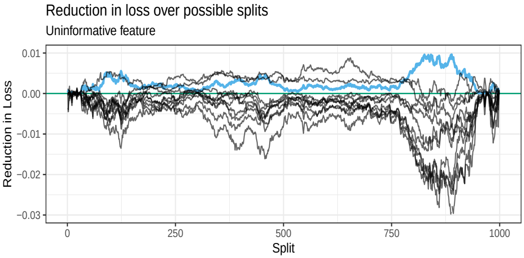


- Asymptotic loss
- Training set

Inspection of loss during greedy search

Reduction in loss

$$R(s) = -\frac{1}{2n}\left[\frac{\left(\sum_{i\in I_t} g_i\right)^2}{\sum_{i\in I_t} h_i} - \left(\frac{\left(\sum_{i\in I_l} g_i\right)^2}{\sum_{i\in I_l} h_i} + \frac{\left(\sum_{i\in I_r} g_i\right)^2}{\sum_{i\in I_r} h_i}\right)\right]$$

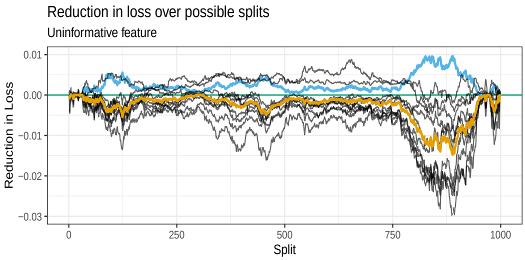


- Asymptotic loss
- Training set
- 10 different test sets

Inspection of loss during greedy search

Reduction in loss

$$R(s) = -\frac{1}{2n}\left[\frac{\left(\sum_{i\in I_t} g_i\right)^2}{\sum_{i\in I_t} h_i} - \left(\frac{\left(\sum_{i\in I_l} g_i\right)^2}{\sum_{i\in I_l} h_i} + \frac{\left(\sum_{i\in I_r} g_i\right)^2}{\sum_{i\in I_r} h_i}\right)\right]$$



- 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:

$$R_{tr}(u) - E_0[R^0_{te}(u)] \xrightarrow[n\to\infty]{D} C_t\pi_t \frac{B(u)^2}{u(1-u)}$$

First main result: Convergence of empirical process

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

$$R_{tr}(u) - E_0[R^0_{te}(u)] \xrightarrow[n\to\infty]{D} C_t \pi_t \frac{B(u)^2}{u(1-u)}$$

- 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:

$$R_{tr}(u) - E_0[R_{te}^0(u)] \xrightarrow[n\to\infty]{D} C_t \pi_t \frac{B(u)^2}{u(1-u)}$$

- 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:

$$R_{tr}(u) - E_0[R_{te}^0(u)] \xrightarrow[n\to\infty]{D} C_t \pi_t \frac{B(u)^2}{u(1-u)}$$

- 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 $u \in (0,1)$.

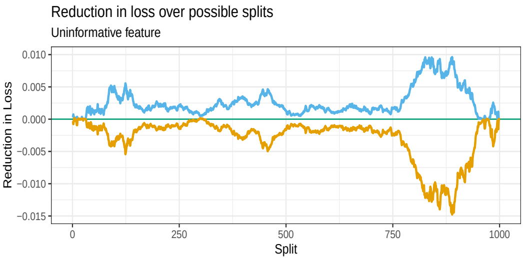
First main result: Convergence of empirical process

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

$$R_{tr}(u) - E_0[R^0_{te}(u)] \xrightarrow[n\to\infty]{D} C_t \pi_t \frac{B(u)^2}{u(1-u)}$$

- 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.

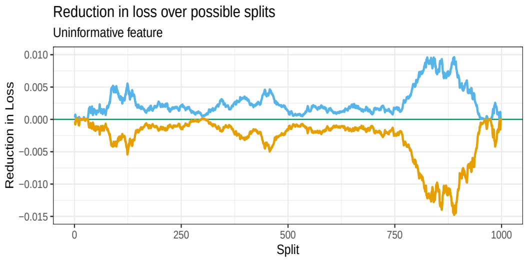
First main result: Expectations



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:

$$\tilde{C}_R = E\left[\max\left\{R_{tr}(u) - R^0_{te}(u), 0 < u < 1\right\}\right]$$
$$= -C_t \pi_t E\left[\max\left\{\frac{B(u)^2}{u(1-u)}, 0 < u < 1\right\}\right]$$

Comments on the Brownian bridge

The Brownian bridge is well studied

- Definition: $B(u) := W(u)|W(1) = 0$ (standard Brownian bridge).
- $W(u)$ is a Brownian motion.
- $E\left[B(u)\right] = 0$ and $Var\left[B(u)\right] = u(1-u)$.
- $Cov\left[B(u), B(v)\right] = u(1-v)$ for $u < v$.

Comments on the Brownian bridge

The Brownian bridge is well studied

- Definition: $B(u) := W(u)|W(1) = 0$ (standard Brownian bridge).
- $W(u)$ is a Brownian motion.
- $E\left[B(u)\right] = 0$ and $Var\left[B(u)\right] = u(1-u)$.
- $Cov\left[B(u), B(v)\right] = u(1-v)$ for $u < v$.

If only things were that easy...

- We have not yet included multiple features!

Comments on the Brownian bridge

The Brownian bridge is well studied

- Definition: $B(u) := W(u)|W(1) = 0$ (standard Brownian bridge).
- $W(u)$ is a Brownian motion.
- $E\left[B(u)\right] = 0$ and $Var\left[B(u)\right] = u(1-u)$.
- $Cov\left[B(u), B(v)\right] = u(1-v)$ for $u < v$.

If only things were that easy...

- We have not yet included multiple features!
- And only treated the continuous feature case...

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

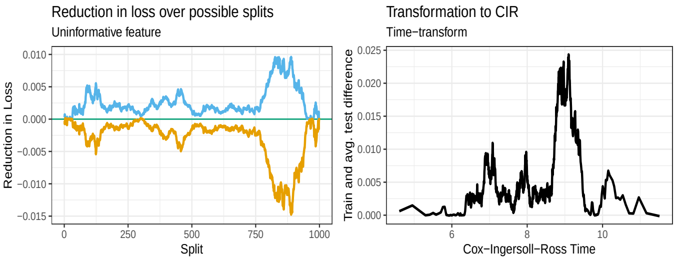
- Work with the maximum over discrete time-observations on the bridge.
- Works perfectly (figure later), and holds AIC-type criteria as a special case(!).

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.

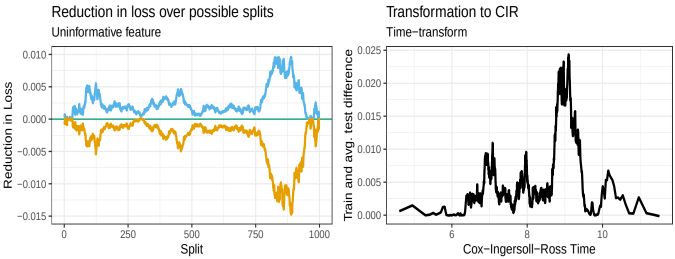
Reduction in loss transform to CIR

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 proved 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

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

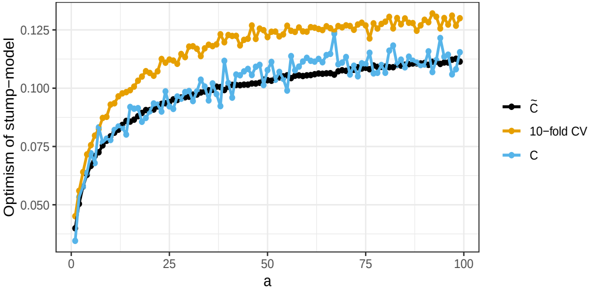
- Assuming independence the outer maximum has distribution

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

- ... and its expectation may be evaluated as

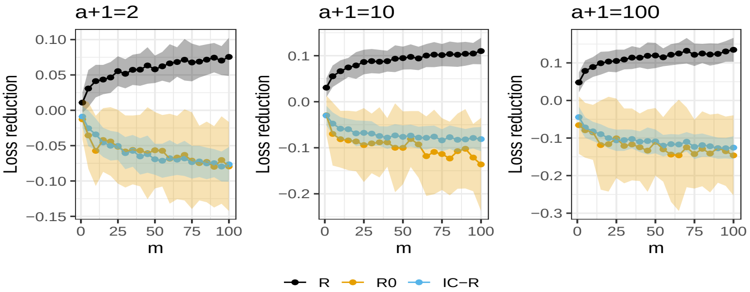
$$E[\max_j Y_j] = \int_0^\infty P(\max_j Y_j > z) dz.$$

Sanity check: Optimism vs increasing number of splits



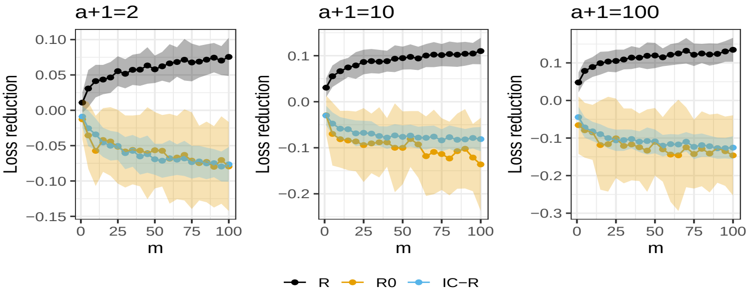
- a is the number of possible split-points ($n = 100$).
- 10-fold CV uses only 90% of the data, thus more optimism.
- C is average of 1000 test loss.
- $\tilde{C}$ is our information criterion.
- Average values of 1000 different experiments, thus quite robust

Sanity check: Increasing dimensions



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

Sanity check: Increasing dimensions



- a is the number of possible split-points ($n = 100$).
- m is the number of features.
- Not crazy!

Boosting applications outline

- How the information criterion is employed to GTB.
- Does it actually work?
- Some studies on real and simulated data.
- Would the researcher win the ML competition?

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

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

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

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

2. Stop the iterative boosting algorithm when

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

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.

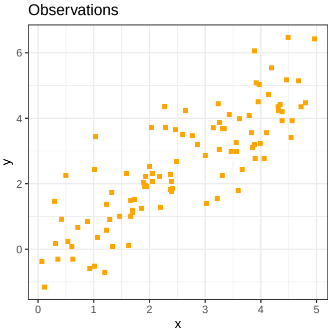
1. Initialize model with a constant value: $f^{(0)}(\mathbf{x}) = \arg\min_{\eta} \sum_{i=1}^n l(y_i, \eta)$.
2. **for** $k = 1$ to K: **while** the inequality (2) evaluates to **`false`**
   i) Compute derivatives g_i and h_i for all $i = 1 : n$.
   ii) Determine q_k by the iterative binary splitting 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** $f^{(K)}(\mathbf{x})$.

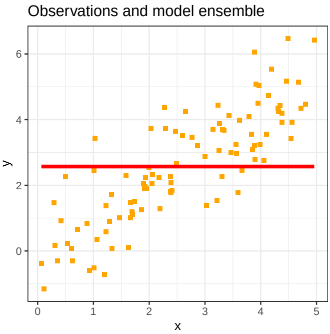
Does it work?

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



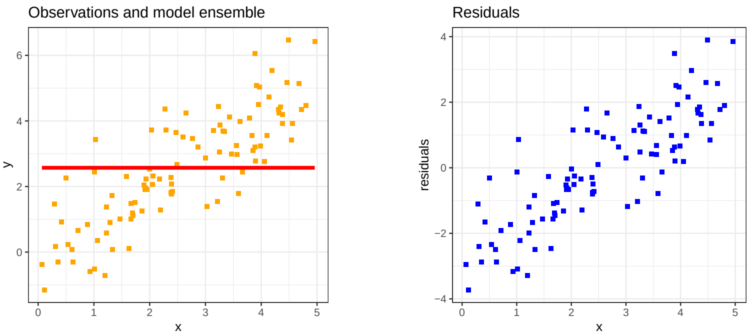
Does it work?

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



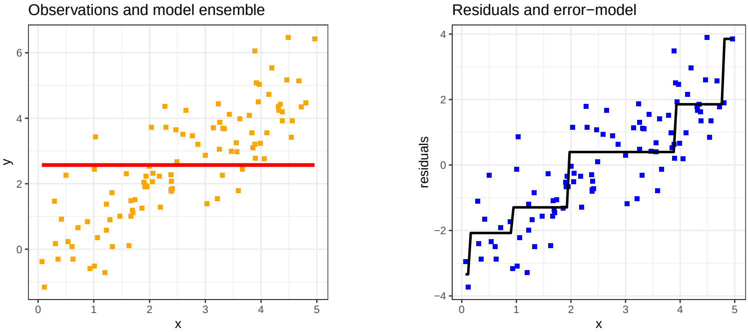
Does it work?

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



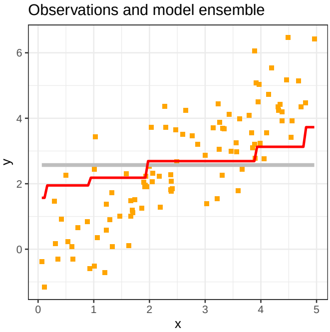
Does it work?

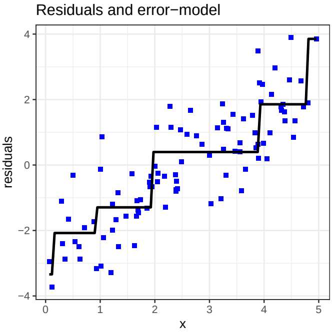
- The tree-boosting animation in the introduction was generated by this algorithm.



Does it work?

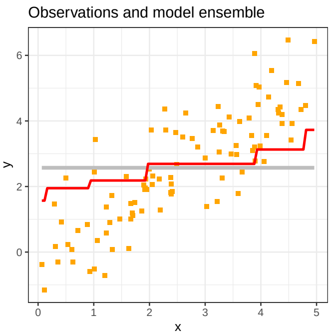
- The tree-boosting animation in the introduction was generated by this algorithm.

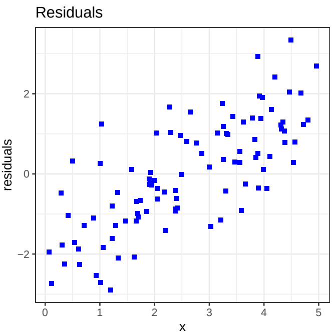




Does it work?

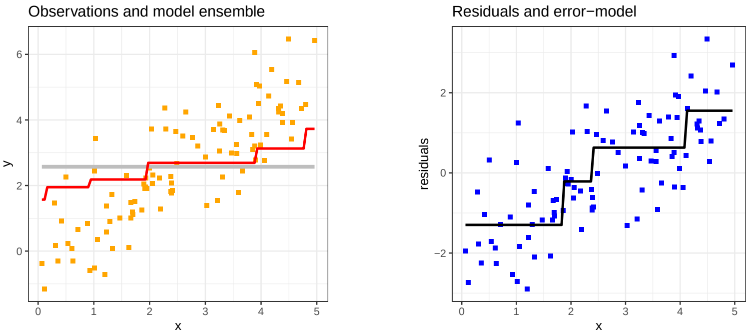
- The tree-boosting animation in the introduction was generated by this algorithm.





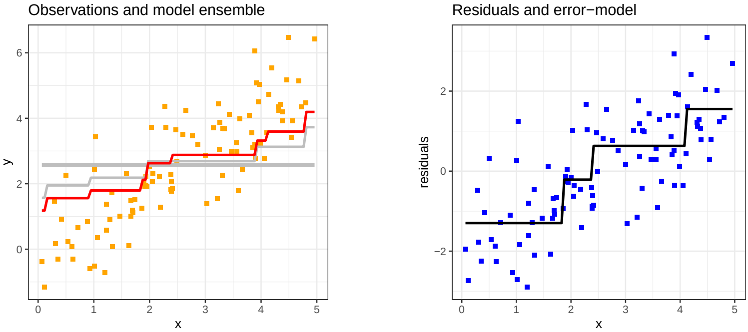
Does it work?

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



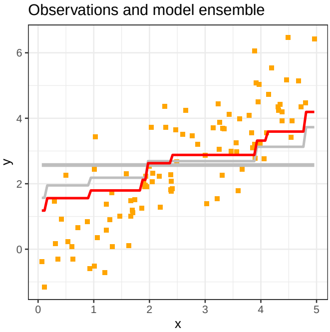
Does it work?

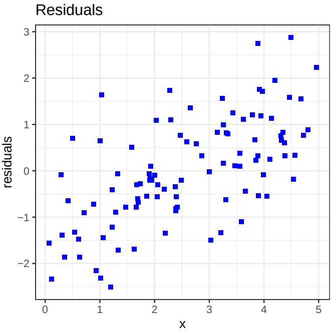
- The tree-boosting animation in the introduction was generated by this algorithm.



Does it work?

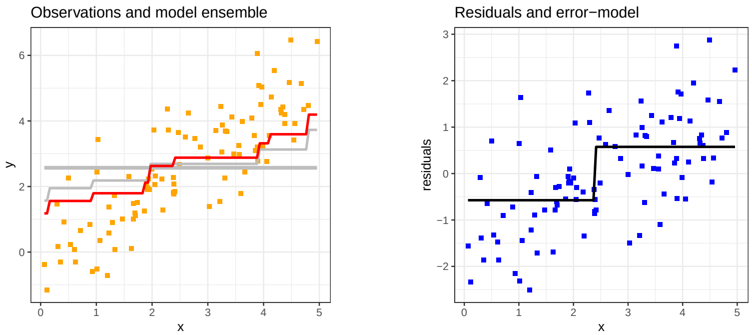
- The tree-boosting animation in the introduction was generated by this algorithm.





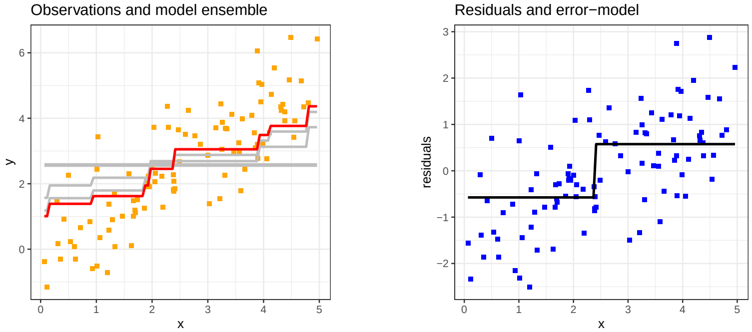
Does it work?

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



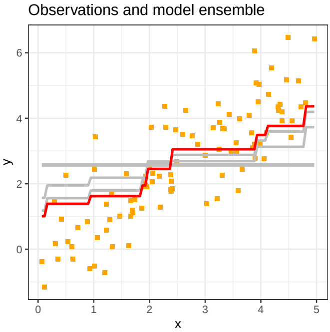
Does it work?

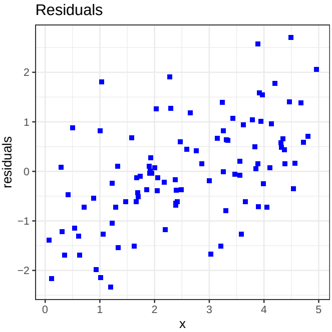
- The tree-boosting animation in the introduction was generated by this algorithm.



Does it work?

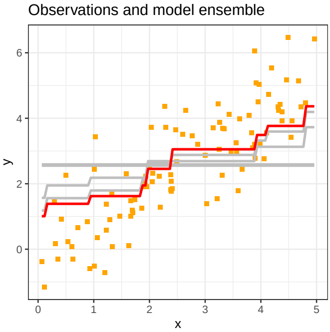
- The tree-boosting animation in the introduction was generated by this algorithm.

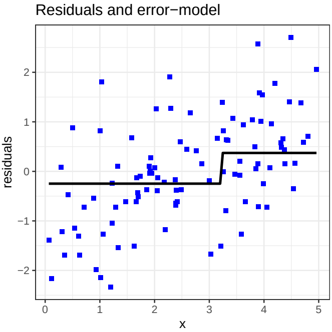




Does it work?

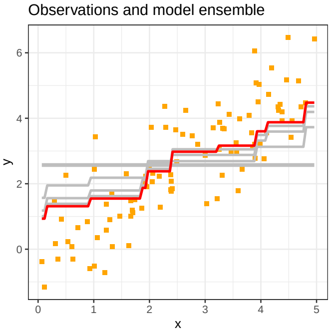
- The tree-boosting animation in the introduction was generated by this algorithm.

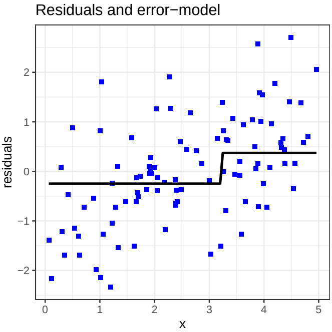




Does it work?

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





Does it work?

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

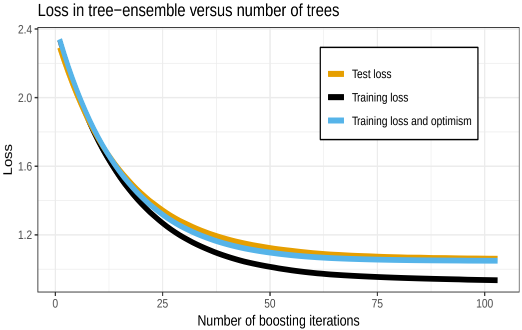


Figure: Training (black) and test loss (orange) and estimated generalization error (blue), for a tree-boosting ensemble trained on 1000 observations from a linear model: $y \sim N(\mathbf{x}, 1)$. The blue line visualizes inequality 2.

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 researcher enters the ML competition

- Would he win?

The researcher enters the ML competition

- Would he win?

There are additional techniques for improvement

Most notably...

The researcher enters the ML competition

- Would he win?

There are additional techniques for improvement

Most notably...

- L1-L2 regularization
- Stochastic sampling of both rows and columns
- Our trees are optimal if they all were the last (unscaled) tree

The researcher enters the ML competition

- Would he win?

There are additional techniques for improvement

Most notably...

- L1-L2 regularization
- Stochastic sampling of both rows and columns
- Our trees are optimal if they all were the last (unscaled) tree

But there are benefits!

- The key to many ML competitions is the feature engineering
- Possibility of very quickly (and automatically) testing for relevant features

1. Background
2. An information theoretic approach
3. Applications to the boosting algorithm
4. Implementation and notes on future developments

Implementation and notes outline

- The AGTBoost package.
- Future developments.

AGTBoost package

- Algorithm implemented in the `GBTorch` project on Github: `https://github.com/Blunde1/agtboost`
- Install the `R`-package from GitHub (soon on CRAN):

```r
devtools::install_github("Blunde1/agtboost/R-package")
```

- Implemented in `C++`, depends upon `Eigen` for linear algebra
- Depends on `Rcpp` for the `R`-package

AGTBoost package

- Algorithm implemented in the `GBTorch` project on Github: `https://github.com/Blunde1/agtboost`
- Install the `R`-package from GitHub (soon on CRAN):

```
devtools::install_github("Blunde1/agtboost/R-package")
```

- Implemented in `C++`, depends upon `Eigen` for linear algebra
- Depends on `Rcpp` for the `R`-package
- Designed to be super easy:

```
x <- runif( 500, 0, 4 )
y <- rnorm( 500, x, 1 )
x.test <- runif( 500, 0, 4 )
y.test <- rnorm( 500, x.test, 1 )
# Train agtboost ensemble
mod <- gbt.train( y, as.matrix(x) )
y.pred <- predict( mod, as.matrix( x.test ) )
# Plot predictions on test data
plot( x.test, y.test )
points( x.test, y.pred, col="red" )
```

Future development #1

There are additional techniques for improvement

Most notably...

- L1-L2 regularization
- Stochastic sampling of both rows and columns
- Our trees are optimal if they all were the last (unscaled) tree

Solved!

- Philosophy from LARS / FS_0: Only add as much complexity in a certain direction as it deserves...
- Modifies the standard greedy recursive binary splitting procedure...
- Implemented: `gbt.train(y, x, algorithm="global_subset")`
- Illustrate difference in R

Future development #2

There are additional techniques for improvement

Most notably...

- L1-L2 regularization
- Stochastic sampling of both rows and columns
- Our trees are optimal if they all were the last (unscaled) tree

Hmm!

- Can we automatically tune this?
- Weights, $\mathbf{w}$ resulting from a L-2 regularized objective are still M-estimators...

Future development #2

Around Christmas 2018 I was thinking about this problem in general:

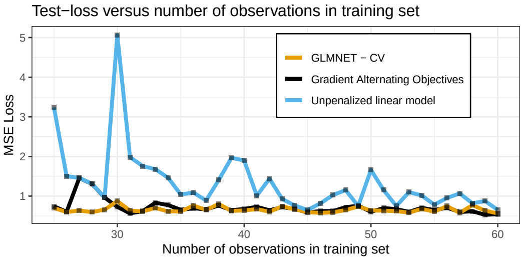
- Given trainin data, how can we automatically know how strongly we should believe in a prior about some H_0?

Solution

- Create an information criterion, taking into account the alternation between the regularized and un-regularized objectives.
- Make it differentiable...
- Gradient descent: $\nabla_\lambda \left[l(y, f(x; \hat{\theta}(\lambda))) + \mathrm{tr}(Q(\lambda) H(\lambda)^\intercal) \right]$

Future development #2

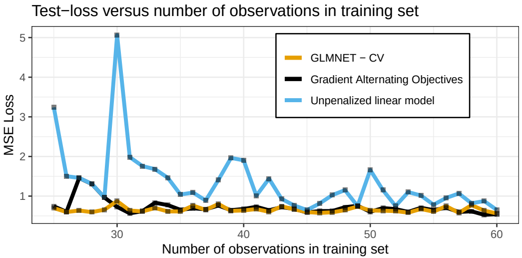
Figure: Hitters data: dimensions 263×20



- Gradient descent: $\nabla_\lambda \left[l(y, f(x; \hat{\theta}(\lambda))) + \mathrm{tr}(Q(\lambda)H(\lambda)^\intercal) \right]$
- Equivalent results to GLMNET for ridge regression.
- Extremely computationally expensive...

Future development #2

Figure: Hitters data: dimensions 263×20



- Gradient descent: $\nabla_\lambda \left[l(y, f(x; \hat{\theta}(\lambda))) + \mathrm{tr}(Q(\lambda)H(\lambda)^\intercal) \right]$
- Equivalent results to GLMNET for ridge regression.
- Extremely computationally expensive...
- But, what is locally constant and the base-learners of choice?

Future development #3

There are additional techniques for improvement

Most notably...

- L1-L2 regularization
- Stochastic sampling of both rows and columns
- Our trees are optimal if they all were the last (unscaled) tree

Future development #3

There are additional techniques for improvement

Most notably...

- L1-L2 regularization
- Stochastic sampling of both rows and columns
- Our trees are optimal if they all were the last (unscaled) tree

I have no idea!...

Future development #3

There are additional techniques for improvement

Most notably...

- L1-L2 regularization
- Stochastic sampling of both rows and columns
- Our trees are optimal if they all were the last (unscaled) tree

I have no idea!...

- ... But it is a super interesting subject!
- Bootstrapping or subsampling? Not theoretically clear why one over the other.
- *Might* have to adjust the information criterion.

We could make this even better!

When this project has matured...

any help on the following subjects are welcome:

- Utilizing sparsity (possibly Eigen sparsity)
- Parallelisation (CPU and/or GPU)
- Distribution (Python, Java, Scala, ...)

Conclusion

Work being done

- Two papers are on arxiv and submitted to journals
- Theory-paper: `https://arxiv.org/abs/2008.05926`
- Implementation: `https://arxiv.org/abs/2008.12625`
- Some assumptions may be relaxed
- CRAN within two weeks
- And writing more papers

Conclusion

Work being done

- Two papers are on arxiv and submitted to journals
- Theory-paper: `https://arxiv.org/abs/2008.05926`
- Implementation: `https://arxiv.org/abs/2008.12625`
- Some assumptions may be relaxed
- CRAN within two weeks
- And writing more papers

Why I'm excited

- Tree-boosting is very popular!
- Removing manual tuning may potentially help quite a few people...
- and opens up for new applications
- Training a highly competitive model will be computationally trivial

Bibliography

Akaike, H. (1974).
A new look at the statistical model identification.
IEEE transactions on automatic control, 19(6):716–723.

Murata, N., Yoshizawa, S., and Amari, S.-i. (1994).
Network information criterion-determining the number of hidden units for an artificial neural network model.
IEEE Transactions on Neural Networks, 5(6):865–872.

Takeuchi, K. (1976).
Distribution of information statistics and validity criteria of models.
Mathematical Science, 153:12–18.