An information criterion for gradient boosted trees

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

[1]Department of Mathematics and Physics
University of Stavanger

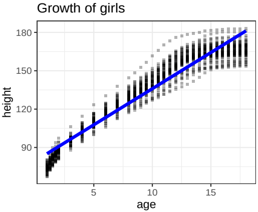
[2]Department of Mathematics
University of Bergen

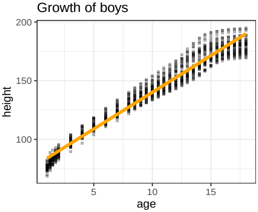
BigInsight - Wednesday Lunch
UiO, Oslo
9th October 2019

Outline

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

Question 1: Linear regression

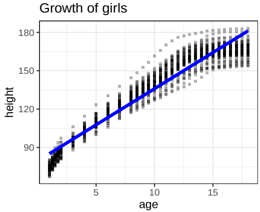


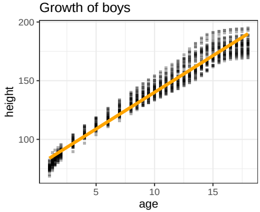


Researcher asks...

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





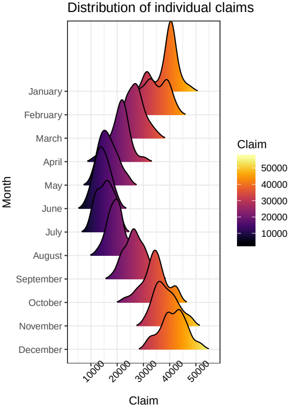
Researcher asks...

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):
$\hat{\beta} = \arg\min_\beta \sum_i \left(y_i - f(\texttt{age}_i, \texttt{sex}_i; \beta)\right)^2$.

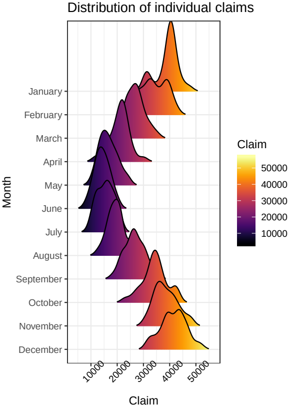
Question 2: Generalized linear models



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...

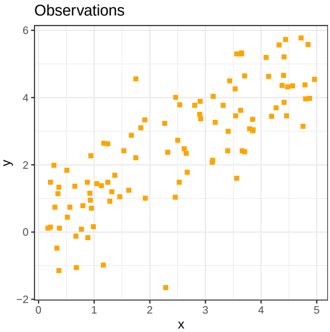
Try gradient boosting?

- 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

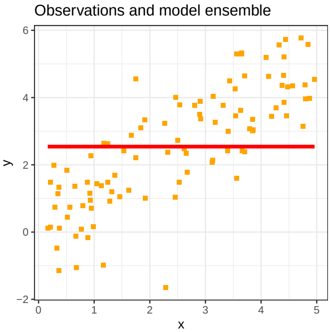
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.



Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

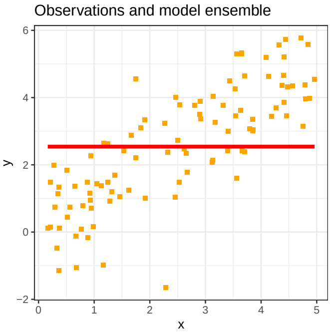
- Start with a constant value: $f^{(0)} = \arg\min_{\eta} \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

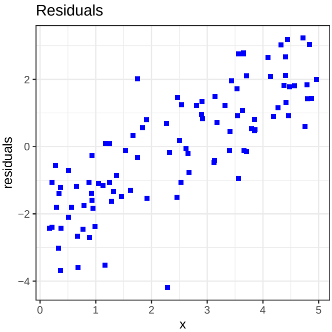


Gradient boosting, what is going on behind the scenes?

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.

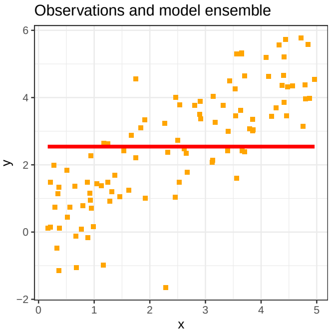


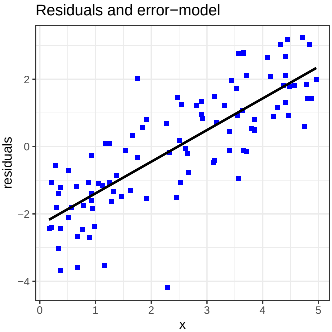


Gradient boosting, what is going on behind the scenes?

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.

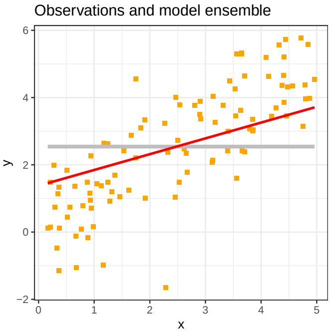


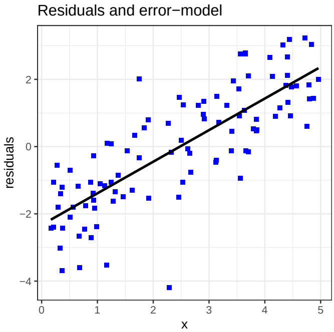


Gradient boosting, what is going on behind the scenes?

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.

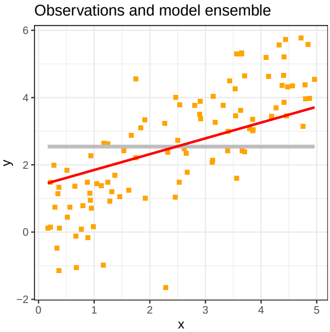


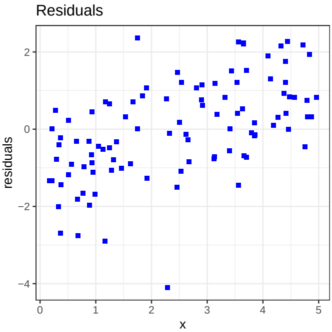


Gradient boosting, what is going on behind the scenes?

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.

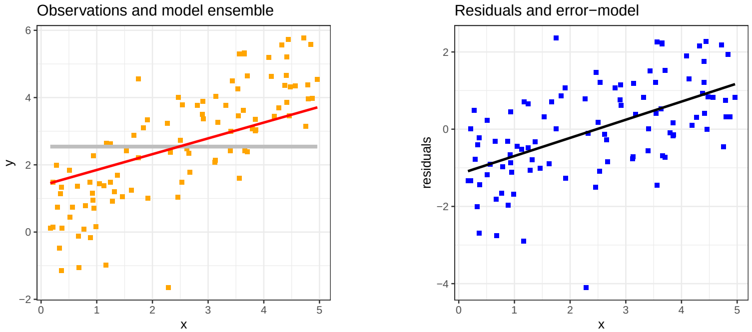




Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

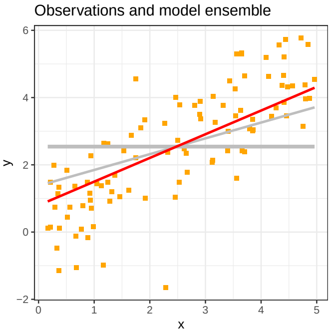
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

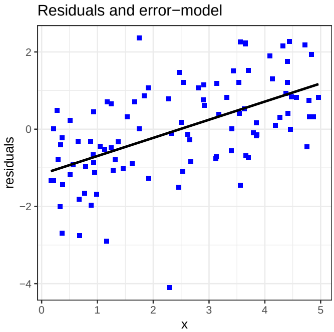


Gradient boosting, what is going on behind the scenes?

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.

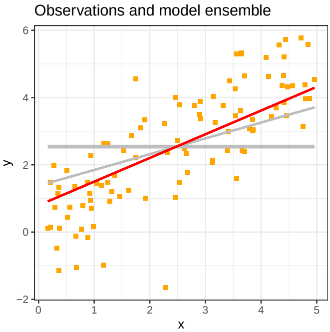


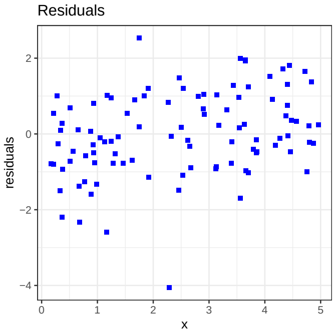


Gradient boosting, what is going on behind the scenes?

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.

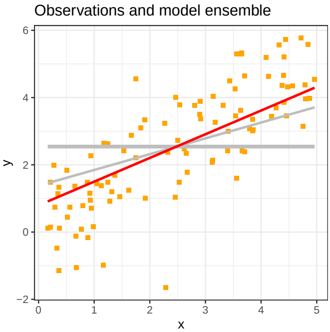


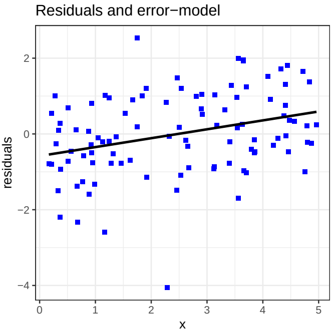


Gradient boosting, what is going on behind the scenes?

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.

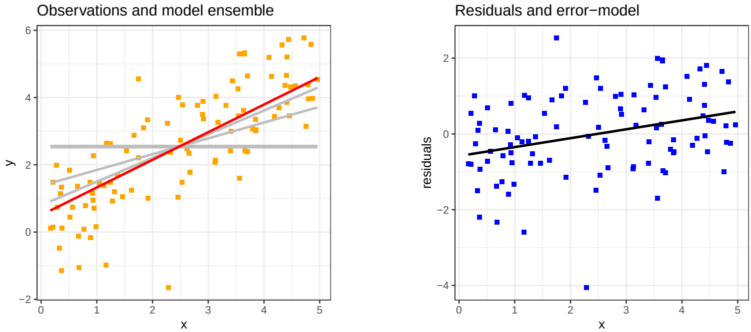




Gradient boosting, what is going on behind the scenes?

Gradient boosting attacks the supervised learning problem directly

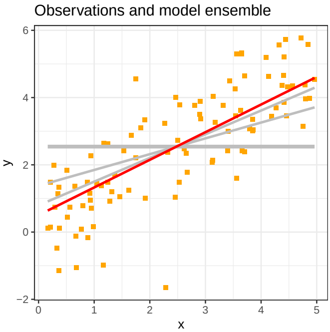
- Start with a constant value: $f^{(0)} = \arg\min_\eta \sum_i l(y_i, \eta)$
- Iteratively, add δf_k to $f^{(k-1)}$, where f_k is trained on the "error" (MSE case) of $f^{(k-1)}$, and δ is some small number scaling f_k.

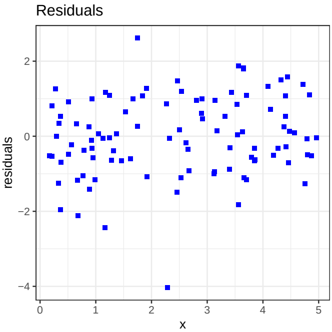


Gradient boosting, what is going on behind the scenes?

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.

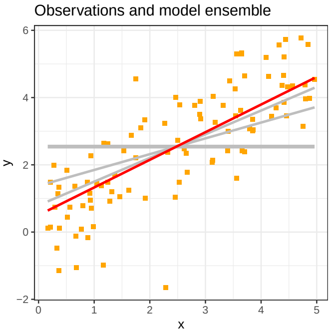


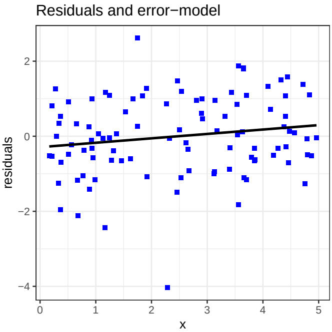


Gradient boosting, what is going on behind the scenes?

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.

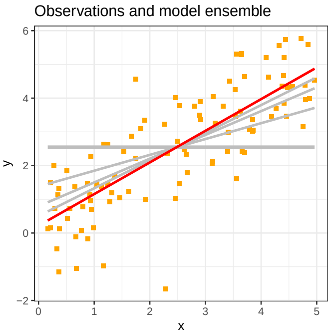


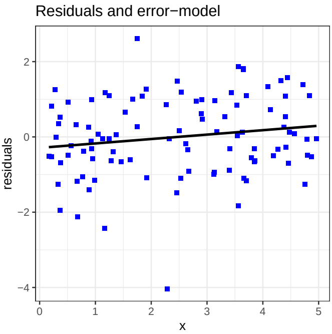


Gradient boosting, what is going on behind the scenes?

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...

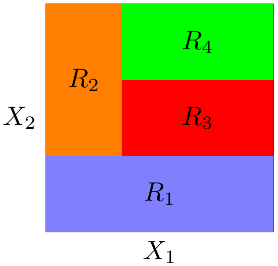
- More interesting with non-linear learning procedures for f_k
- But needs to retain the possibility of a simple (sparse) model.
- We need something that can be non-linear but adapts this to data!
- Trees: complexity from the simple mean or ”tree-stumps” to potentially a complete fit to training data.

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?

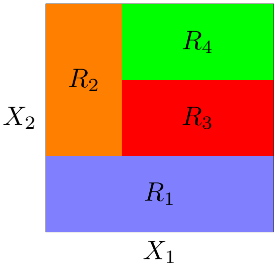


The tree-learning prcedure: recursive binary splitting

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

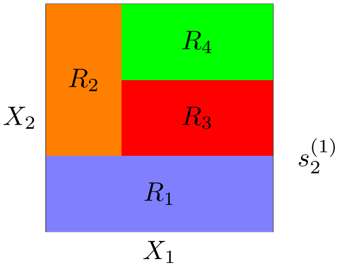
1. Start with a constant prediction for all of feature space
2. Split a leaf-node into two regions (on feature j and split-point s_j chosen by some criteria).
3. Continue step 2 recursively on all leaves.

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

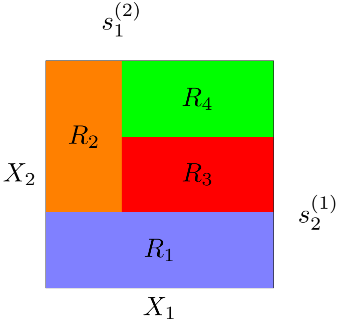
1. Start with a constant prediction for all of feature space
2. Split a leaf-node into two regions (on feature j and split-point s_j chosen by some criteria).
3. Continue step 2 recursively on all leaves.

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

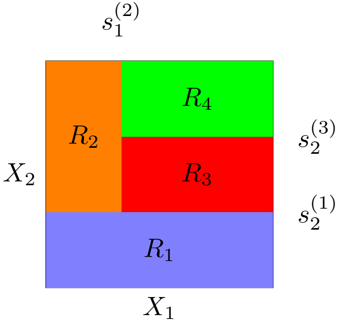
1. Start with a constant prediction for all of feature space
2. Split a leaf-node into two regions (on feature j and split-point s_j chosen by some criteria).
3. Continue step 2 recursively on all leaves.

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?



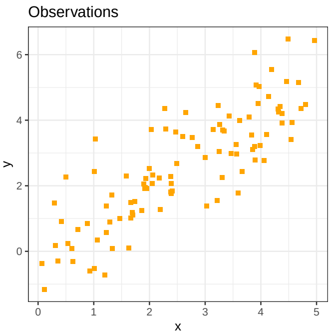
Recursive binary splitting

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

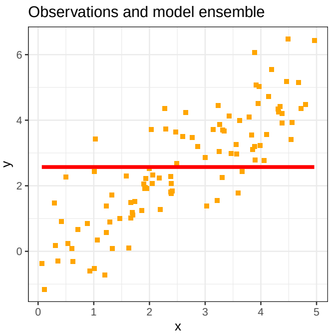
$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\bigg|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \text{ and } h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\bigg|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$



Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

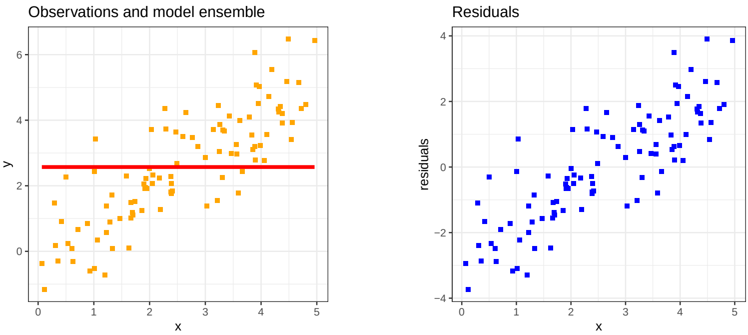
$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \text{ and } h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$



Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

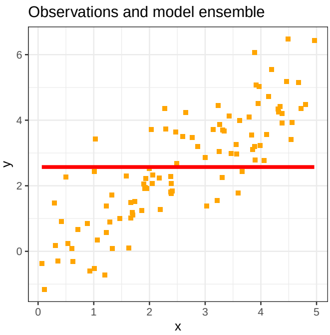
$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \quad \text{and} \quad h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$

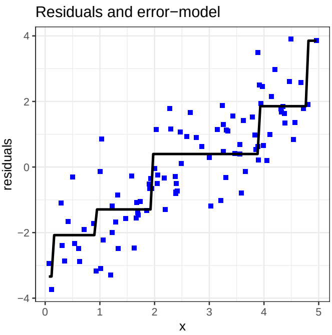


Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

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

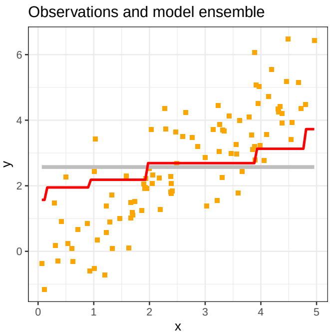


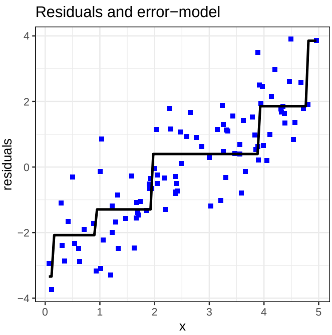


Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

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

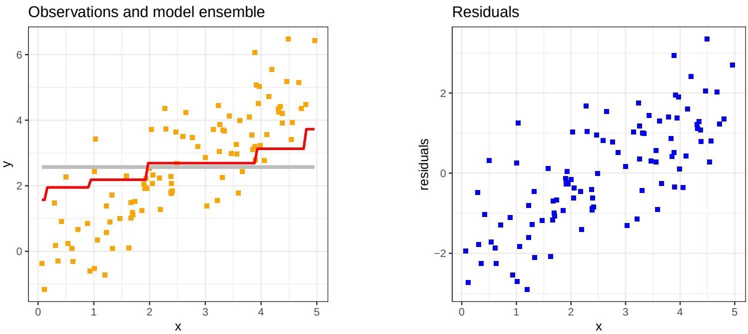




Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

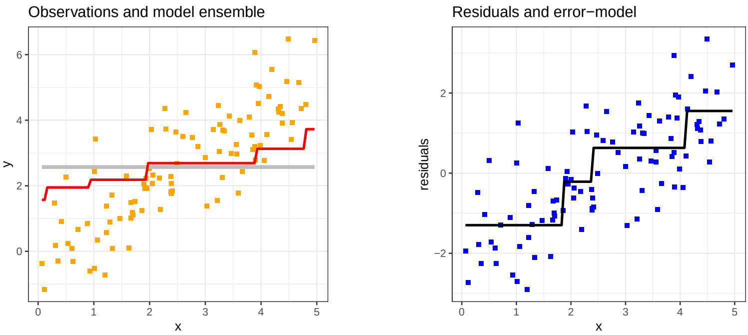
$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \quad \text{and} \quad h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$



Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

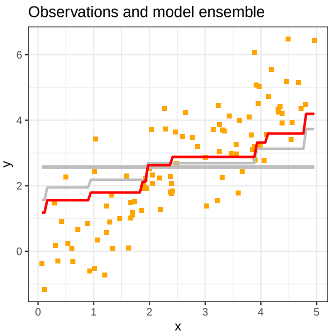
$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i) \Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \quad \text{and} \quad h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i) \Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} .$$

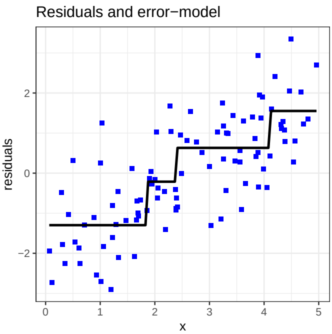


Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

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

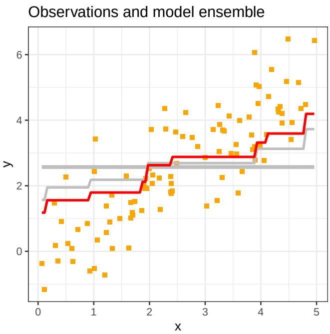


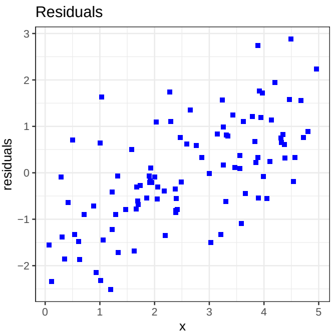


Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

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

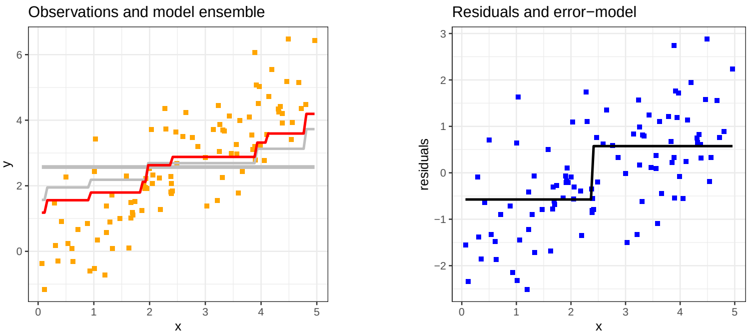




Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

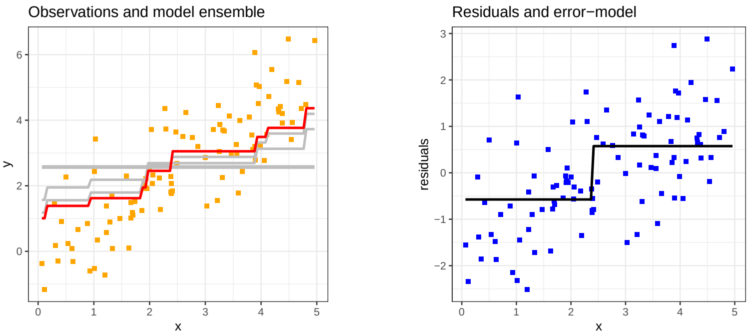
$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \text{ and } h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$



Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

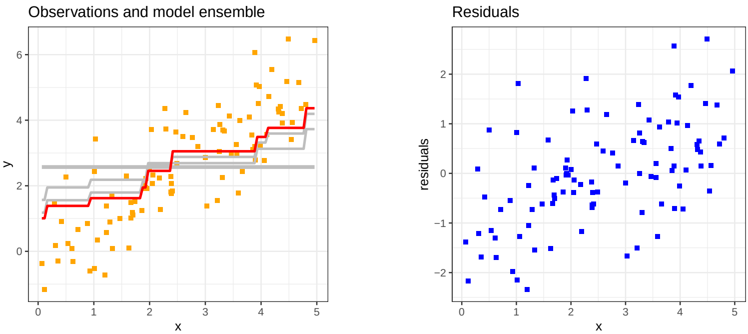
$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \quad \text{and} \quad h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$



Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

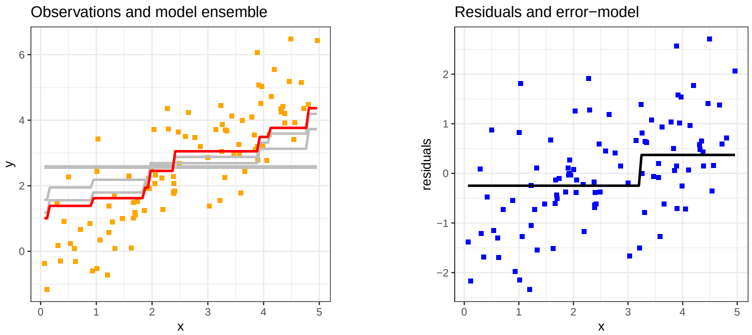
$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \quad \text{and } h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$



Second order gradient tree boosting

Iteratively add δf_k where f_k are trees trained on derivatives

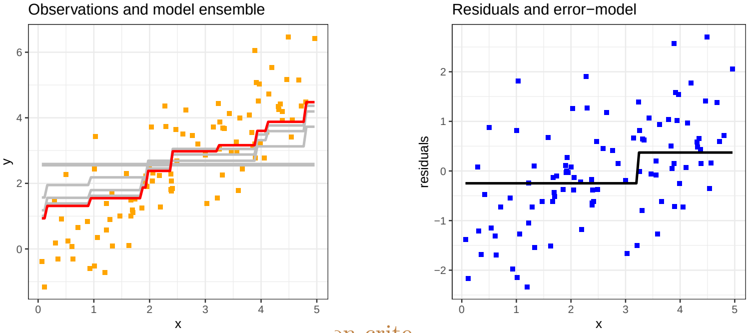
$$g_{ik} = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)} \quad \text{and} \quad h_{ik} = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)\Big|_{\hat{y}_i = f^{(k-1)}(\mathbf{x}_i)}.$$



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: Complexity

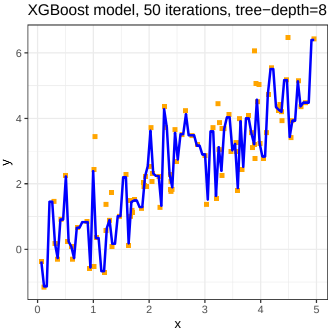
This was too easy!

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

Second order gradient tree boosting: Complexity

This was too easy!

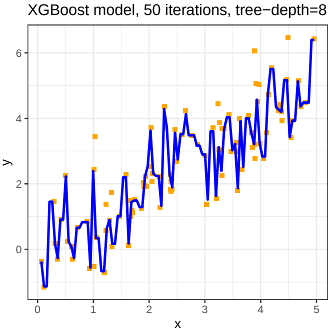
- How do we choose the complexity of the trees?
- And how many boosting iterations?



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...

Revisit the supervised learning problem

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

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

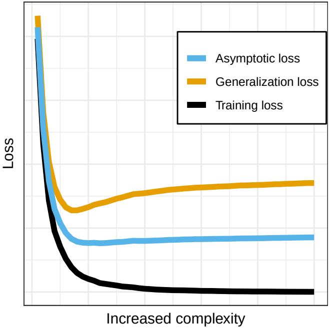
where $(\mathbf{x}^0 y^0)$ are unseen in the training-phase, and therefore independent of $\hat{\theta}$ trained from $(\mathbf{x}, y)$.

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

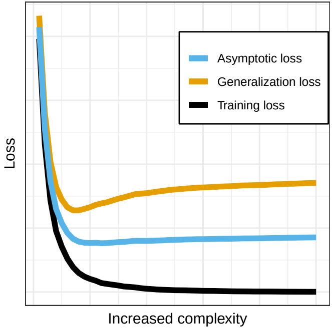


Revisit the supervised learning problem

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 $C(\hat{\theta}) \approx \frac{2}{n} \sum_{i=1}^{n} \mathrm{Cov}(y_i, \hat{y}_i)$
- Useful to talk about *asymptotic loss*

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

But the ensemble complexity is unknown...

The main idea:

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

And hope that we may...

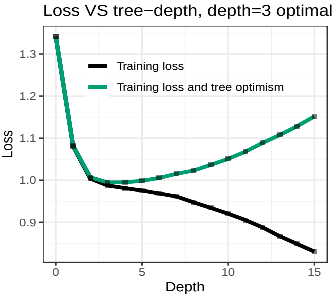
But the ensemble complexity is unknown...

The main idea:

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

And hope that we may...

1. Adaptively control the complexity of each tree



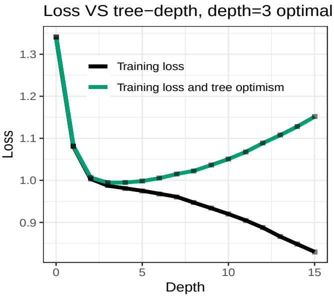
But the ensemble complexity is unknown...

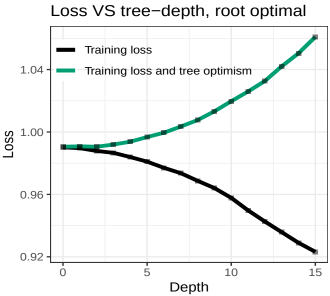
The main idea:

- 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





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

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

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

The local optimism

But can we use this for trees?

The local optimism

But can we use this for trees?

- Conditionally on known tree-topology / regions R
- We define the local optimism for a leaf node t

$$\mathcal{C}(t|q) = E_{y,\hat{w}_t}\left[\frac{\partial^2}{\partial \hat{w}_t^2}\tilde{l}(y,\hat{w}_t)\right]\mathrm{Var}_{\hat{w}_t}[\hat{w}_t]$$

q is the tree-topology, $\hat{w}_t$ is the prediction in region t

The local optimism

But can we use this for trees?

- Conditionally on known tree-topology / regions R
- We define the local optimism for a leaf node t

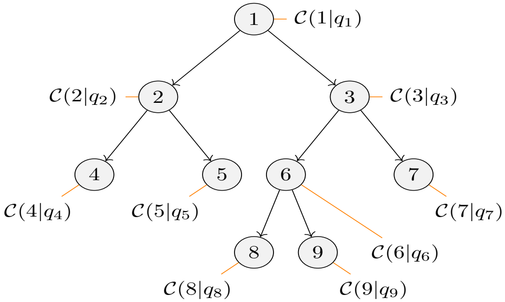
$$\mathcal{C}(t|q) = E_{y,\hat{w}_t}\left[\frac{\partial^2}{\partial \hat{w}_t^2}\tilde{l}(y,\hat{w}_t)\right]\mathrm{Var}_{\hat{w}_t}[\hat{w}_t]$$

q is the tree-topology, $\hat{w}_t$ is the prediction in region t

Correspondingly for internal nodes...

- At some stage in the tree building process every node will have been a leaf node.
- Define this to be the local optimism $\mathcal{C}(t|q)$ for the internal node t in the fully grown tree.

The local optimism: Illustration



The optimism induced from fitting the predictions $\mathbf{w}$ in the leaf-nodes conditioned on the final topology is given as

$$\hat{C}(\hat{\mathbf{w}}|q) = \sum_{t \in \{4,5,8,9,7\}} \mathcal{C}(t|q)\pi_t, \;\; \pi_t = P(q(\mathbf{x}) = t)$$

The tree-learning procedure learns the future map q

The steps - first consider only one feature

- Relate the distance between the training and asymptotic loss to a gamma distribution
- Fit the gamma using knowledge of its shape and expectation

The tree-learning procedure learns the future map q

The steps - first consider only one feature

- Relate the distance between the training and asymptotic loss to a gamma distribution
- Fit the gamma using knowledge of its shape and expectation

The steps - consider $m \geq 1$ features

- Under H_0: no feature is relevant
- Distance between training and asymptotic loss as expected maximum of multiple realizations of the gamma RV

- Warning: several approximations!

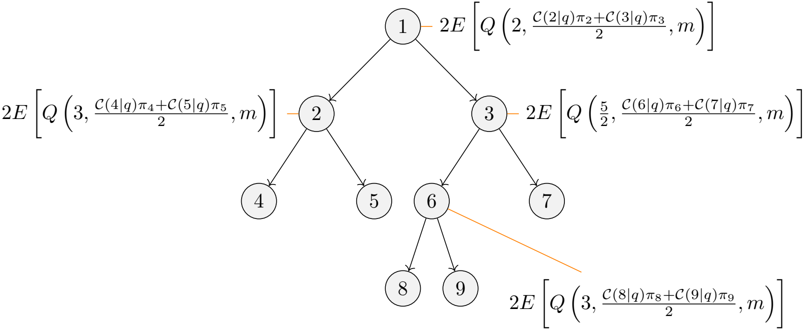
Gives the following result:

$$\hat{C}_m(\hat{\mathbf{w}}, q) = 2 \sum_{t \notin \mathcal{L}} E\left[Q\left(d(t), \frac{\mathcal{C}(L(t)|q)\pi_{L(t)} + \mathcal{C}(R(t)|q)\pi_{R(t)}}{2}, m \right) \right]$$

- $\mathcal{L}$ is the set of leaf nodes
- $Q(\alpha, \beta, m)$ is the maximum of m gamma random variables with shape α and scale β
- $L(t)$ and $R(t)$ returns the index of left and right child-nodes respectively
-

$$d(t) = \begin{cases} 3 & \text{if } L(t) \in \mathcal{L} \text{ and } R(t) \in \mathcal{L} \\ \frac{5}{2} & \text{if } L(t) \in \mathcal{L} \text{ and } R(t) \notin \mathcal{L} \\ \frac{5}{2} & \text{if } L(t) \notin \mathcal{L} \text{ and } R(t) \in \mathcal{L} \\ 2 & \text{if } L(t) \notin \mathcal{L} \text{ and } R(t) \notin \mathcal{L}. \end{cases}$$

Visualization of the result



- Sum up the node-contributions to obtain the optimism estimate

Estimation

Quantities must admit evaluation and efficient computation:

Estimation

Quantities must admit evaluation and efficient computation:

- Conditioned on q, $\hat{\mathbf{w}}$ are M-estimators.
-

$$
\begin{aligned}
\lim_{n\to\infty} \mathcal{C}(t|q)\pi_t &= E_{y,\hat{w}_t}\left[\frac{\partial^2}{\partial \hat{w}_t^2} l(y,\hat{w}_t)\right]\mathrm{Var}_{\hat{w}_t}[\hat{w}_t]\pi_t \\
&= \frac{\sum_{i\in I_t}\left(g_i + h_i\hat{w}_t\right)^2}{n\sum_{i\in I_t} h_i},\ I_t \text{ is indexes of obs in leaf } t
\end{aligned}
$$

Estimation

Quantities must admit evaluation and efficient computation:

- Conditioned on q, $\hat{\mathbf{w}}$ are M-estimators.
-

$$\lim_{n\to\infty} \mathcal{C}(t|q)\pi_t = E_{y,\hat{w}_t}\left[\frac{\partial^2}{\partial \hat{w}_t^2} l(y,\hat{w}_t)\right] \mathrm{Var}_{\hat{w}_t}[\hat{w}_t]\pi_t$$

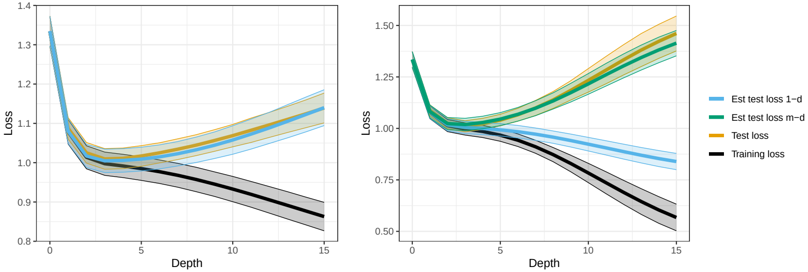
$$= \frac{\sum_{i\in I_t}(g_i + h_i\hat{w}_t)^2}{n\sum_{i\in I_t} h_i},\ I_t \text{ is indexes of obs in leaf } t$$

- Assume features independent, then $E[Q]$ is the expected $m-$th order statistic
- Estimate asymptotically using the gamma-quantile function:

$$E[Q] \sim z\left(\frac{m}{m+1}\right)$$

Some sanity checks: Simulation experiments

100 Datasets and 100 trees, trained on 1 and 100 features.



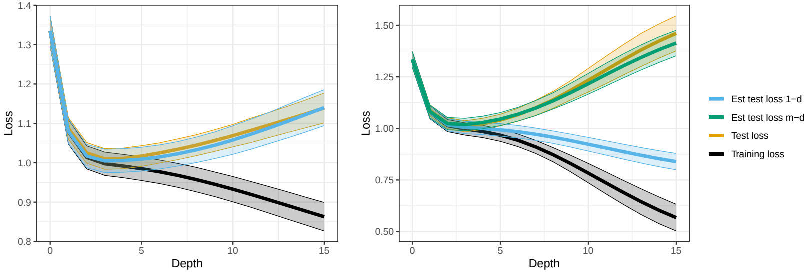
(a) One (relevant) feature

(b) 100 features of which 99 are noise

Figure: Average $\pm 2SD$ for training (black) and test (orange) MSE loss, together with estimated optimism (blue: 1 feature, green: 100 features)

Some sanity checks: Simulation experiments

100 Datasets and 100 trees, trained on 1 and 100 features.



(a) One (relevant) feature

(b) 100 features of which 99 are noise

Figure: Average $\pm 2SD$ for training (black) and test (orange) MSE loss, together with estimated optimism (blue: 1 feature, green: 100 features)

- Not crazy!

Berent Å. S. Lunde An information criterion for gradient boosted trees

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

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. For two hierarchical trees, q^0 and q^1, where q^1 holds one more split than q^0, don't split if

   $$E\left[\hat{l}(y, f(\mathbf{x}; \hat{\mathbf{w}}^0, \hat{q}^0)) - \hat{l}(y, f(\mathbf{x}; \hat{\mathbf{w}}^1, \hat{q}^1))\right] + C_m(\hat{\mathbf{w}}^0, \hat{q}^0) - C_m(\hat{\mathbf{w}}^1, \hat{q}^1)$$

   is smaller than zero.

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. For two hierarchical trees, q^0 and q^1, where q^1 holds one more split than q^0, don't split if

$$E\left[\hat{l}(y, f(\mathbf{x}; \hat{\mathbf{w}}^0, \hat{q}^0)) - \hat{l}(y, f(\mathbf{x}; \hat{\mathbf{w}}^1, \hat{q}^1))\right] + C_m(\hat{\mathbf{w}}^0, \hat{q}^0) - C_m(\hat{\mathbf{w}}^1, \hat{q}^1)$$

   is smaller than zero.

2. Stop the iterative boosting algorithm when

$$\frac{\delta\left(\delta-2\right)}{2n} \sum_{t \in \mathcal{L}_k} \frac{G_{tk}^2}{H_{tk}} + \delta C_m\left(\hat{\mathbf{w}}_{t,k}, q_{t,k}\right) > 0.$$

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: $f^{(0)}(\mathbf{x}) = \hat{\eta} = \arg\min_{\eta} \sum_{i=1}^n l(y_i, \eta)$.

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.

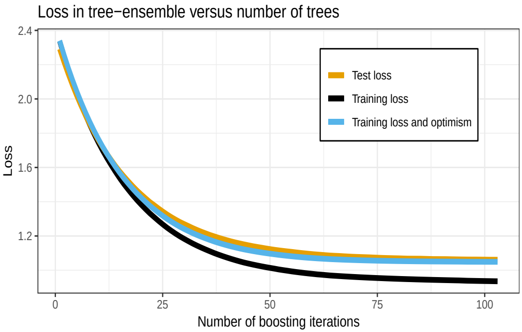


Figure: Training (black) and test loss (orange) and estimated generalization error (blue), for a tree-boosting ensemble trained on 1000 observations from a linear model: $y \sim N(\mathbf{x}, 1)$. The blue line visualizes inequality 2.

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	Dimensions	GBTorch	GLM	Random forest	XGBoost
Boston	506×14	1.07 (0.162)	1.3 (0.179)	0.876 (0.15)	1 (0.176)
Ozone	111×4	0.827 (0.22)	0.666 (0.131)	0.669 (0.182)	1 (0.202)
Auto	392×311	1.16 (0.136)	11.1 (14.5)	0.894 (0.134)	1 (0.188)
Carseats	400×12	1.2 (0.168)	0.413 (0.0432)	1.16 (0.141)	1 (0.115)
College	777×18	1.3 (0.948)	0.55 (0.154)	1.07 (0.906)	1 (0.818)
Hitters	263×20	1.05 (0.362)	1.21 (0.347)	0.796 (0.31)	1 (0.318)
Wage	3000×26	1.96 (1.72)	289 (35.4)	82.2 (21.3)	1 (1.01)
Caravan	5822×86	1.02 (0.0508)	1.12 (0.115)	1.31 (0.168)	1 (0.0513)
Default	10000×4	0.938 (0.068)	0.902 (0.0698)	2.83 (0.51)	1 (0.0795)
OJ	1070×18	0.996 (0.0496)	0.952 (0.0721)	1.17 (0.183)	1 (0.0703)
Smarket	1250×7	0.999 (0.00285)	1 (0.00651)	1.04 (0.0164)	1 (0.00259)
Weekly	1089×7	0.992 (0.0082)	0.995 (0.0123)	1.02 (0.0195)	1 (0.00791)

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 Caravan dataset (5822×86 classification), our implementation took 2.68 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 Caravan dataset (5822×86 classification), our implementation took 2.68 seconds to train.
- Using a 30% validation set, XGB took 4.85 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 Caravan dataset (5822×86 classification), our implementation took 2.68 seconds to train.
- Using a 30% validation set, XGB took 4.85 seconds
- One minute 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 Caravan dataset (5822×86 classification), our implementation took 2.68 seconds to train.
- Using a 30% validation set, XGB took 4.85 seconds
- One minute using 10-fold CV: the number of boosting iterations
- About 16 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 Caravan dataset (5822×86 classification), our implementation took 2.68 seconds to train.
- Using a 30% validation set, XGB took 4.85 seconds
- One minute using 10-fold CV: the number of boosting iterations
- About 16 minutes to learn one additional hyperparameter
- About 2.65 hours 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

GBTorch package

- Algorithm implemented in the `GBTorch` project on Github: `https://github.com/Blunde1/gbtorch`
- Install the R-package from GitHub:

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

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

GBTorch package

- Algorithm implemented in the GBTorch project on Github: https://github.com/Blunde1/gbtorch
- Install the R-package from GitHub:

```
devtools::install_github("Blunde1/gbtorch/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 gbtorch 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 in GBTorch: `gbt.train(y, x, greedy_complexities=T)`
- Better results than XGB on ISLR / ESL data!
  0.941 0.794 1.02 0.984 1.1 0.975 0.989 0.996 0.914 0.964 0.998 0.991

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:

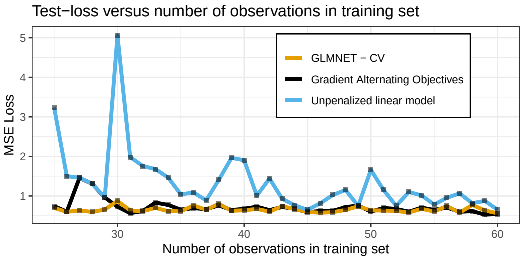
- Given trainin data, how can we automatically know how strongly we should believe in a prior about some H_0?

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

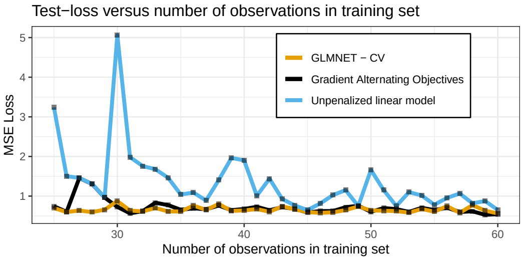
Figure: Hitters data: dimensions 263×20



- 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?

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

- Writing a paper on the work presented
- Robustness of theory
- Will continue development after submission
- And write more papers on future developments

Conclusion

Work being done

- Writing a paper on the work presented
- Robustness of theory
- Will continue development after submission
- And write more papers on future developments

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.