Ensemble Methods (Part II)
Gradient Boosting

Schwartz

January 5, 2017

The most powerful prediction algorithm on the planet

In "The BellKor Solution to the Netflix Grand Prize" Yehuda Koren reported on the use of gradient boosted decision trees (GBDT) in the top performing algorithm. The Netflix Prize was an open competition for the best collaborative filtering algorithm to predict user ratings for films. The competition was open to anyone not connected with Netflix or a resident of Cuba, Iran, Syria, North Korea, Myanmar or Sudan. On 21 September 2009, the grand prize of $1,000,000 was given to the Pragmatic Chaos team which bested Netflix's own prediction algorithm by 10.06%. "The Ensemble" tied the best score 19 minutes and 56 seconds later but got nothing. BellKor's solution hasn't been used in production as it is "too complicated". Today gradient boosting (usually in the form of XGBoost) satisfies itself with being the winningest algorithm in much smaller challenges such as Kaggle competitons.

Objectives

- Understand gradient boosting

Objectives

- Understand gradient boosting
  - as a basic, simple, general algorithm

Objectives

- Understand gradient boosting
  - as a basic, simple, general algorithm
  - as an ensemble of sequential "weak learners"

Objectives

- Understand gradient boosting
  - as a basic, simple, general algorithm
  - as an ensemble of sequential "weak learners"
  - as a series of regression on sequential residuals

Objectives

- Understand gradient boosting
  - as a basic, simple, general algorithm
  - as an ensemble of sequential "weak learners"
  - as a series of regression on sequential residuals
  - as a general gradient decent algorithm

Objectives

- Understand gradient boosting
  - as a basic, simple, general algorithm
  - as an ensemble of sequential "weak learners"
  - as a series of regression on sequential residuals
  - as a general gradient decent algorithm
  - in it's earliest/latest incarnation AdaBoost/XGBoost

Objectives

- Understand gradient boosting
  - as a basic, simple, general algorithm
  - as an ensemble of sequential "weak learners"
  - as a series of regression on sequential residuals
  - as a general gradient decent algorithm
  - in it's earliest/latest incarnation AdaBoost/XGBoost
- Understand the tuning parameters of tree-based boosting

Objectives

- Understand gradient boosting
  - as a basic, simple, general algorithm
  - as an ensemble of sequential "weak learners"
  - as a series of regression on sequential residuals
  - as a general gradient decent algorithm
  - in it's earliest/latest incarnation AdaBoost/XGBoost
- Understand the tuning parameters of tree-based boosting
  - Distinguish tree parameters versus boosting parameters

Objectives

- Understand gradient boosting
  - as a basic, simple, general algorithm
  - as an ensemble of sequential "weak learners"
  - as a series of regression on sequential residuals
  - as a general gradient decent algorithm
  - in it's earliest/latest incarnation AdaBoost/XGBoost
- Understand the tuning parameters of tree-based boosting
  - Distinguish tree parameters versus boosting parameters
  - Understand implementation/tuning via grid search

Objectives

- Understand gradient boosting
  - as a basic, simple, general algorithm
  - as an ensemble of sequential "weak learners"
  - as a series of regression on sequential residuals
  - as a general gradient decent algorithm
  - in it's earliest/latest incarnation AdaBoost/XGBoost
- Understand the tuning parameters of tree-based boosting
  - Distinguish tree parameters versus boosting parameters
  - Understand implementation/tuning via grid search
- *Critique* and *Sell* gradient boosting

Objectives

- Understand gradient boosting
  - as a basic, simple, general algorithm
  - as an ensemble of sequential "weak learners"
  - as a series of regression on sequential residuals
  - as a general gradient decent algorithm
  - in it's earliest/latest incarnation AdaBoost/XGBoost
- Understand the tuning parameters of tree-based boosting
  - Distinguish tree parameters versus boosting parameters
  - Understand implementation/tuning via grid search
- *Critique* and *Sell* gradient boosting
  - As compared to other ensemble methods

Sequential “weak learners”

1. Student 1 takes exam

Sequential "weak learners"

1. Student 1 takes exam
2. Student 2 takes exam *knowing how student 1 did*

Sequential "weak learners"

1. Student 1 takes exam
2. Student 2 takes exam *knowing how student 1 did*

   ⋮

3. Student k takes exam *knowing how student* $k-1$ *did*

Sequential "weak learners"

1. Student 1 takes exam
2. Student 2 takes exam *knowing how student 1 did*

   ⋮

3. Student k takes exam *knowing how student* $k-1$ *did*

   ⋮

4. Student n takes exam *knowing how student* $n-1$ *did*

Thought Experiment

- *Sequential learners* focus effort on outcomes predicted poorly by previous learners as opposed to those predicted accurately

Thought Experiment

- *Sequential learners* focus effort on outcomes predicted poorly by previous learners as opposed to those predicted accurately
- *Weak learners* consistently predict outcomes but with (high bias, low variance) accuracy only ϵ slightly better than chance

Thought Experiment

- *Sequential learners* focus effort on outcomes predicted poorly by previous learners as opposed to those predicted accurately
- *Weak learners* consistently predict outcomes but with (high bias, low variance) accuracy only ϵ slightly better than chance

Can many *sequential weak learners*
be used to make a powerful prediction tool?

Thought Experiment

- *Sequential learners* focus effort on outcomes predicted poorly by previous learners as opposed to those predicted accurately
- *Weak learners* consistently predict outcomes but with (high bias, low variance) accuracy only ϵ slightly better than chance

Can many *sequential weak learners*
be used to make a powerful prediction tool?

Do you think such an approach could suffer from overfitting?

Sequential Estimators Notation

- Let W be an untrained weak learner and $\phi_k(\boldsymbol{x})$ be an estimator that predicts outcome Y based on features $\boldsymbol{x}$

  Define a new estimator

  $$\phi_{k+1}(\boldsymbol{x}) = \phi_k(\boldsymbol{x}) + \alpha_k W(\boldsymbol{x}, \gamma_k)$$

  where W is trained on $\boldsymbol{x}$ "in the presence" of $\phi_k(\boldsymbol{x})$

Sequential Estimators Notation

- Let W be an untrained weak learner and $\phi_k(\boldsymbol{x})$ be an estimator that predicts outcome Y based on features $\boldsymbol{x}$

  Define a new estimator

  $$\phi_{k+1}(\boldsymbol{x}) = \phi_k(\boldsymbol{x}) + \alpha_k W(\boldsymbol{x}, \gamma_k)$$

  where W is trained on $\boldsymbol{x}$ "in the presence" of $\phi_k(\boldsymbol{x})$

- E.g., let W be a tree and $\gamma_k =$ number of splits $= 1$

Sequential Estimators Notation

- Let W be an untrained weak learner and $\phi_k(\boldsymbol{x})$ be an estimator that predicts outcome Y based on features $\boldsymbol{x}$

  Define a new estimator

  $$\phi_{k+1}(\boldsymbol{x}) = \phi_k(\boldsymbol{x}) + \alpha_k W(\boldsymbol{x}, \gamma_k)$$

  where W is trained on $\boldsymbol{x}$ "in the presence" of $\phi_k(\boldsymbol{x})$

- E.g., let W be a tree and $\gamma_k =$ number of splits $= 1$
  So W is a stump – a weak learner*

Sequential Estimators Notation

- Let W be an untrained weak learner and $\phi_k(\boldsymbol{x})$ be an estimator that predicts outcome Y based on features $\boldsymbol{x}$

  Define a new estimator

  $$\phi_{k+1}(\boldsymbol{x}) = \phi_k(\boldsymbol{x}) + \alpha_k W(\boldsymbol{x}, \gamma_k)$$

  where W is trained on $\boldsymbol{x}$ "in the presence" of $\phi_k(\boldsymbol{x})$

- E.g., let W be a tree and $\gamma_k =$ number of splits $= 1$
  So W is a stump – a weak learner*
- W is fit to *improve* the available prediction of Y from $\phi_k(\boldsymbol{x})$

Sequential Estimators Notation

- Let W be an untrained weak learner and $\phi_k(\boldsymbol{x})$ be an estimator that predicts outcome Y based on features $\boldsymbol{x}$

  Define a new estimator

  $$\phi_{k+1}(\boldsymbol{x}) = \phi_k(\boldsymbol{x}) + \alpha_k W(\boldsymbol{x}, \gamma_k)$$

  where W is trained on $\boldsymbol{x}$ "in the presence" of $\phi_k(\boldsymbol{x})$

- E.g., let W be a tree and $\gamma_k =$ number of splits $= 1$
  So W is a stump – a weak learner*
- W is fit to *improve* the available prediction of Y from $\phi_k(\boldsymbol{x})$
- But the negligible impact of W^* will be further muted by α_k
- α_k **controls the learning rate of the boosting algorithm**

Compare and Contrast

(A) Tree
(B) Bagging
(C) Random Forests
(D) Sequential Weak Learners

Compare and Contrast

(A) Tree $\hat{f}^{(1)}(\boldsymbol{x})$

(B) Bagging $\frac{1}{m}\sum \hat{f}^{(k)}(\boldsymbol{x})$

(C) Random Forests $\frac{1}{m}\sum \hat{f}^{(k)}(\boldsymbol{x})$, with small Cor$[\hat{f}^{(k)}, \hat{f}^{(k')}]$

(D) Sequential Weak Learners $\sum \alpha_k \hat{W}(\boldsymbol{x}, \gamma_k)$

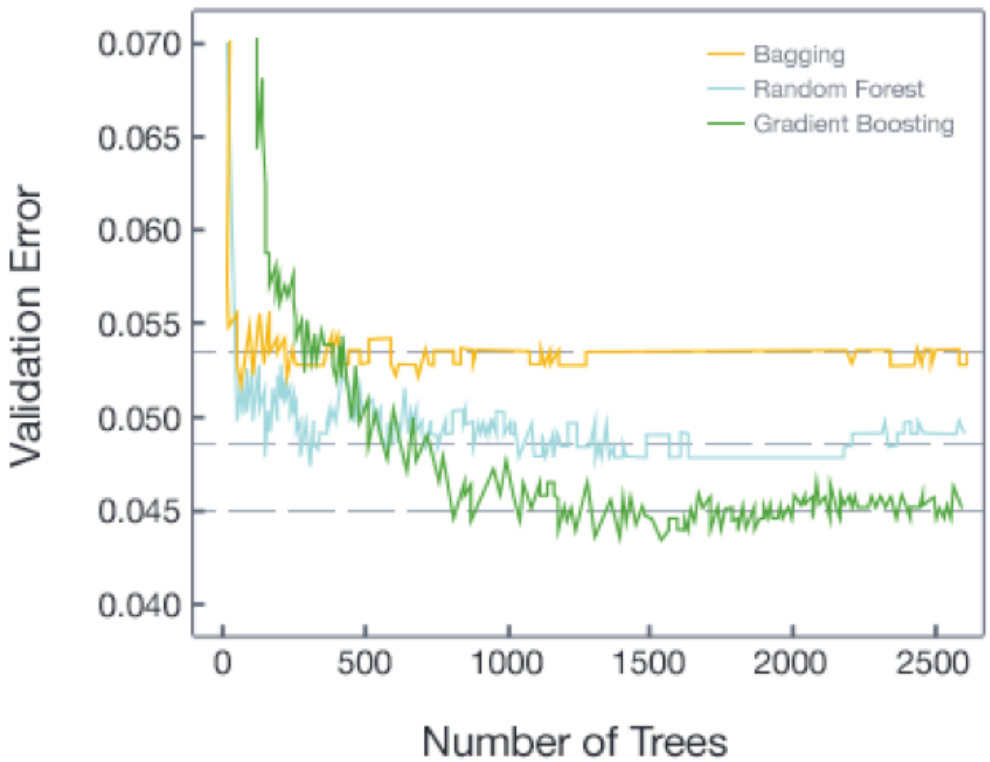
Compare and Contrast

(A) Tree $\hat{f}^{(1)}(\boldsymbol{x})$

(B) Bagging $\frac{1}{m}\sum \hat{f}^{(k)}(\boldsymbol{x})$

(C) Random Forests $\frac{1}{m}\sum \hat{f}^{(k)}(\boldsymbol{x})$, with small $\text{Cor}[\hat{f}^{(k)}, \hat{f}^{(k')}]$

(D) Sequential Weak Learners $\sum \alpha_k \hat{W}(\boldsymbol{x}, \gamma_k)$



Gradient Boosting *Tuning*

Gradient Boosting is typically applied using trees

Gradient Boosting *Tuning*

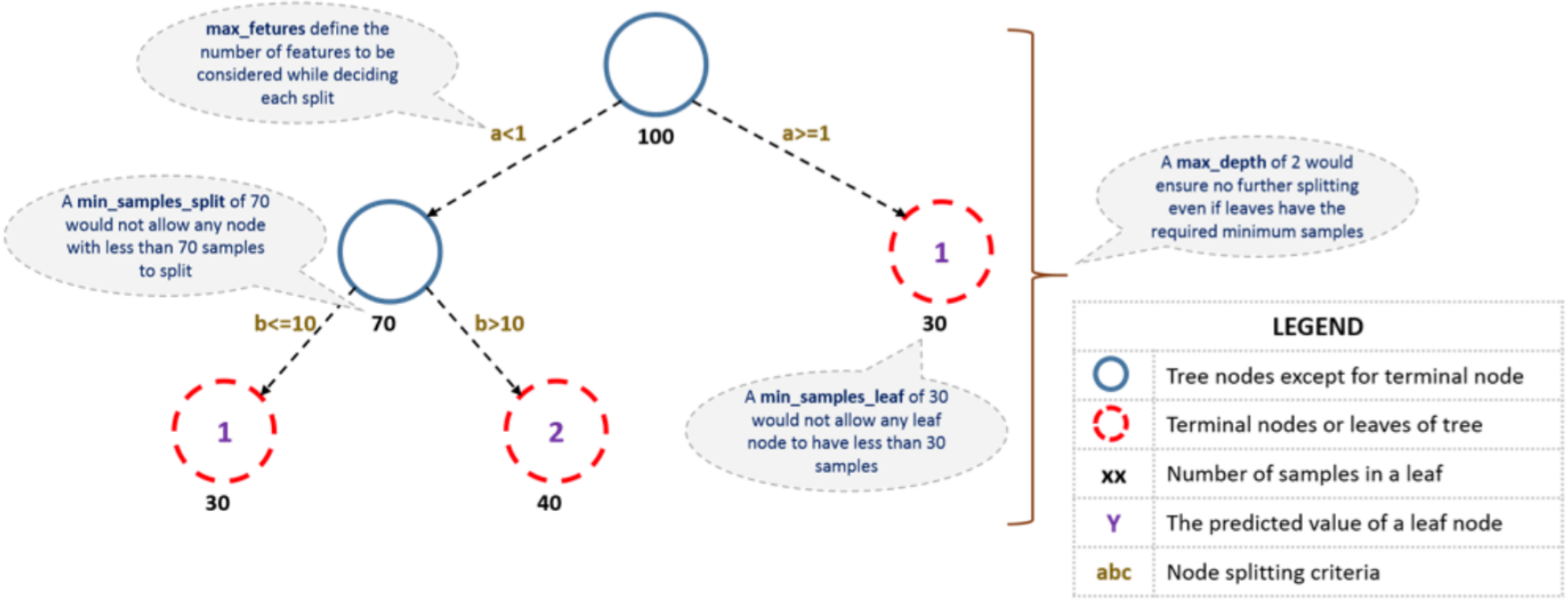
Gradient Boosting is typically applied using trees

- Shallow trees (stumps?) provide weak learners/reduce tuning

Gradient Boosting *Tuning*

Gradient Boosting is typically applied using trees

- Shallow trees (stumps?) provide weak learners/reduce tuning
- Nonetheless, number of leaves, depth, minimum split size, minimum leaf, size, and restricted features* can be tuned...



Gradient Boosting *Tuning*

Gradient Boosting is typically applied using trees

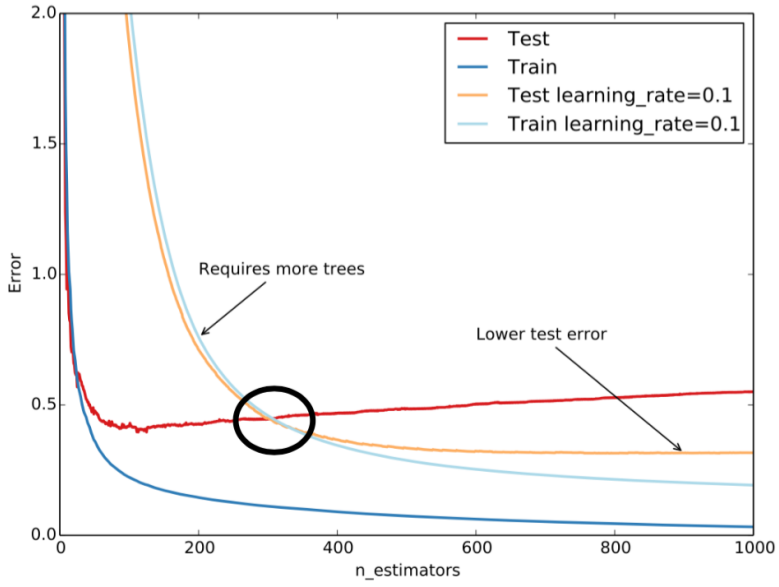
- Shallow trees (stumps?) provide weak learners/reduce tuning
- Nonetheless, number of leaves, depth, minimum split size, minimum leaf, size, and restricted features* can be tuned...
- The learning rate/number of trees α/m needs to be tuned

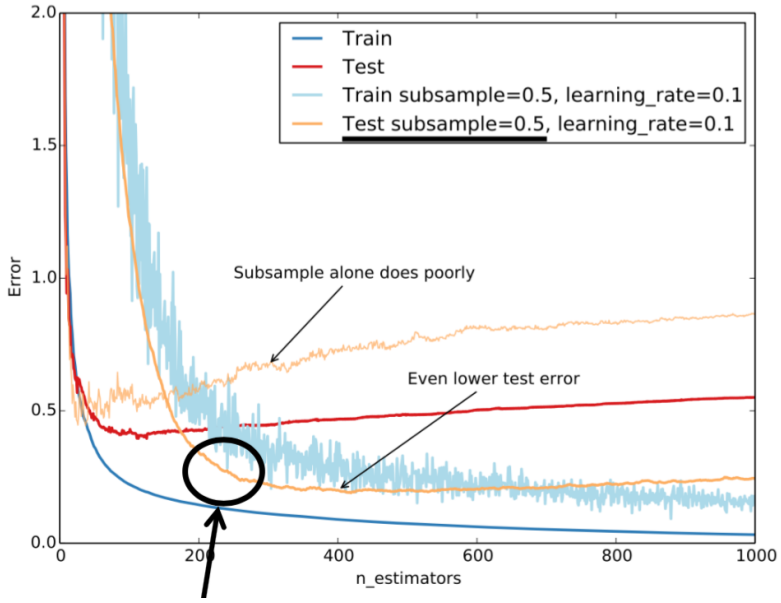
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.grid_search import GridSearchCV
param_grid = {'learning_rate': [0.001, 0.01, 0.1],
              'max_depth': [4, 6],
              'min_samples_leaf': [4, 8, 16],
              'max_features': [0.75, 0.9, 1]}
model = GradientBoostingRegressor(n_estimators=3000)
gs_cv = GridSearchCV(model, param_grid).fit(X, y)

gs_cv.best_params_, gs_cv.best_score_, gs_cv.best_estimator_
```

Gradient Boosting *Tuning*

Gradient Boosting is typically applied using trees

- Shallow trees (stumps?) provide weak learners/reduce tuning
- Nonetheless, number of leaves, depth, minimum split size, minimum leaf, size, and restricted features* can be tuned...
- The learning rate/number of trees α/m needs to be tuned
- Stochastic Gradient Boosting using subsamples also available

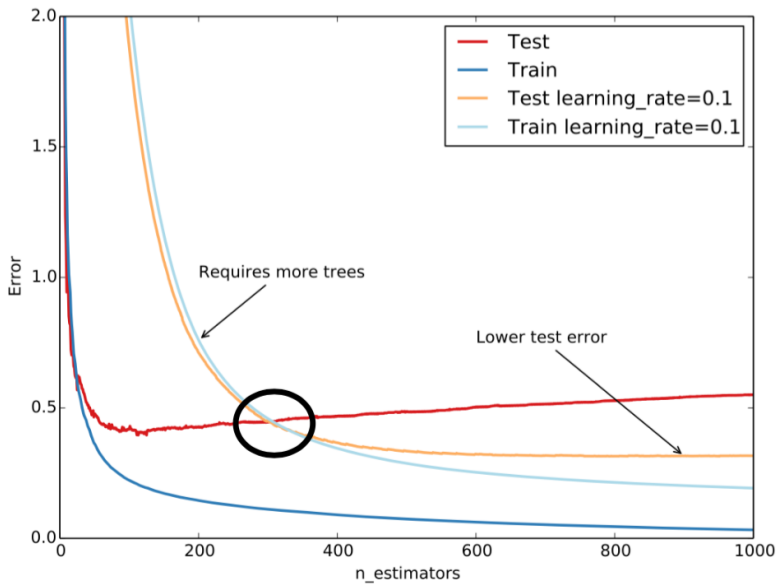


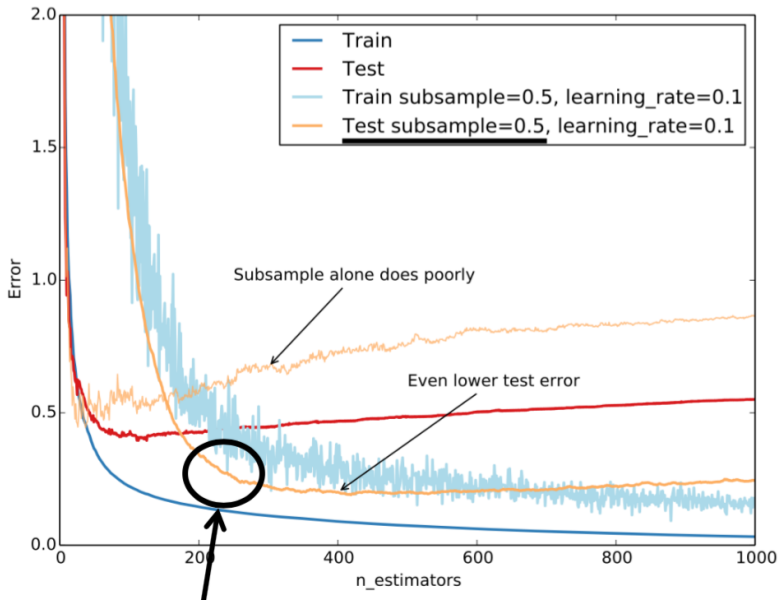


Gradient Boosting *Tuning*

Gradient Boosting is typically applied using trees

- Shallow trees (stumps?) provide weak learners/reduce tuning
- Nonetheless, number of leaves, depth, minimum split size, minimum leaf, size, and restricted features* can be tuned...
- The learning rate/number of trees α/m needs to be tuned
- Stochastic Gradient Boosting using subsamples also available
- Careful parameter tuning is necessary for good performance





Buy or Sell Gradient Boosting?

- Pros
- Cons

Buy or Sell Gradient Boosting?

- Pros
  - Netflix Prize & Kaggle Competitions
- Cons

Buy or Sell Gradient Boosting?

- Pros
  - Netflix Prize & Kaggle Competitions
- Cons
  - Extensive tuning required

Buy or Sell Gradient Boosting?

- Pros
  - Netflix Prize & Kaggle Competitions
- Cons
  - Extensive tuning required
    Unlike Bagging/Random Forests, Boosting *CAN* overfit

Buy or Sell Gradient Boosting?

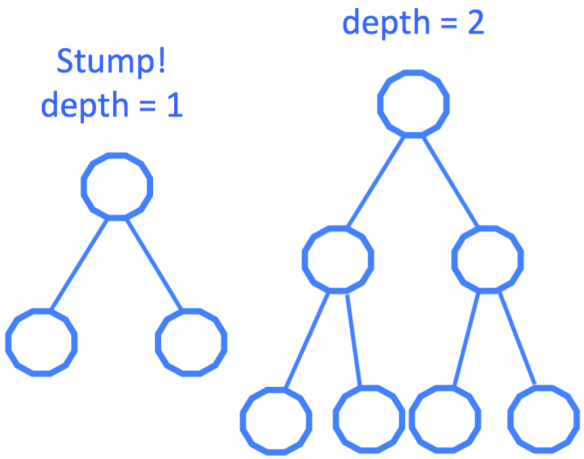
- Pros
  - Netflix Prize & Kaggle Competitions
- Cons
  - Extensive tuning required
    Unlike Bagging/Random Forests, Boosting *CAN* overfit
  - Inherently non-parallelizable

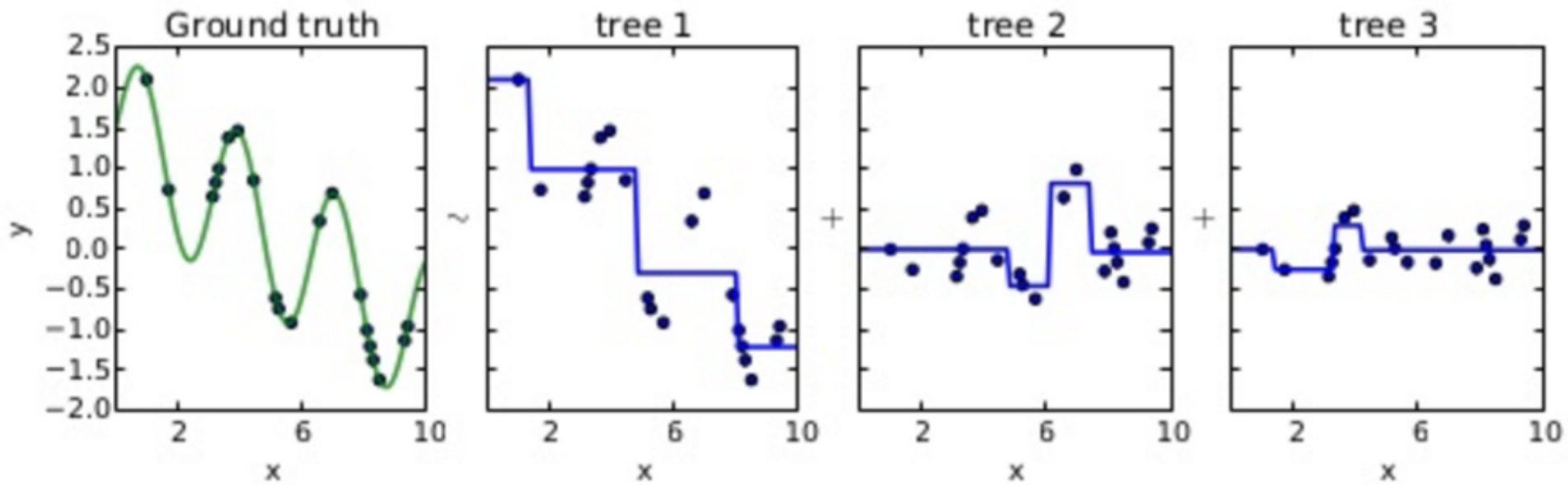
Buy or Sell Gradient Boosting?

- Pros
  - Netflix Prize & Kaggle Competitions
- Cons
  - Extensive tuning required
    Unlike Bagging/Random Forests, Boosting *CAN* overfit
  - Inherently non-parallelizable

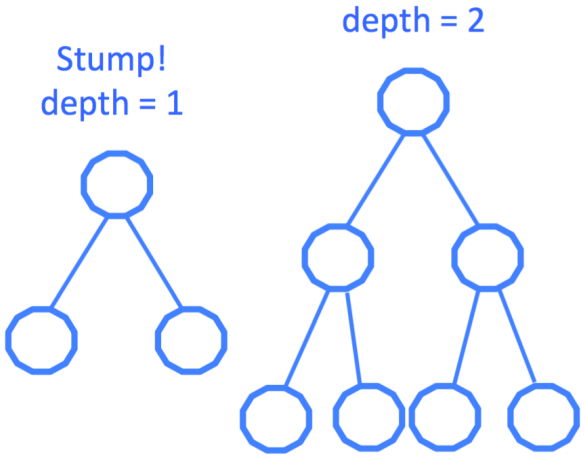
Why does this work better than Random Forests, etc.?

Fitting regressions on sequential residuals

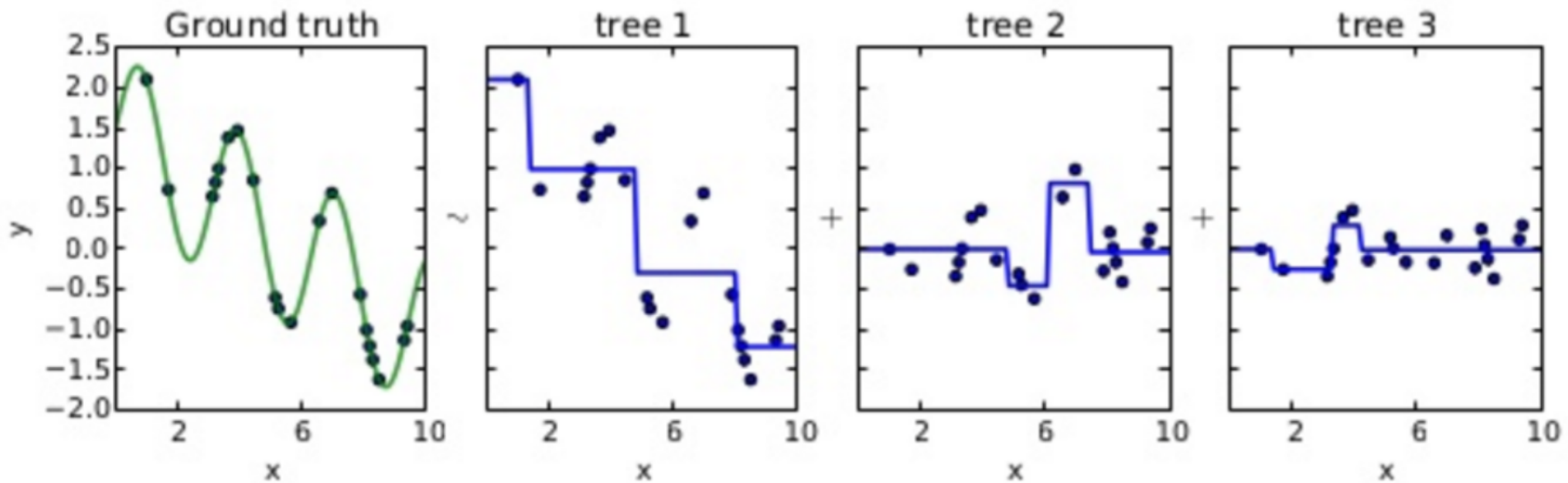




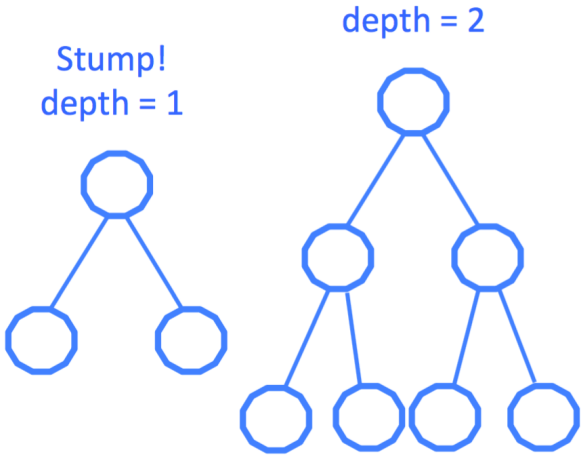
Fitting regressions on sequential residuals



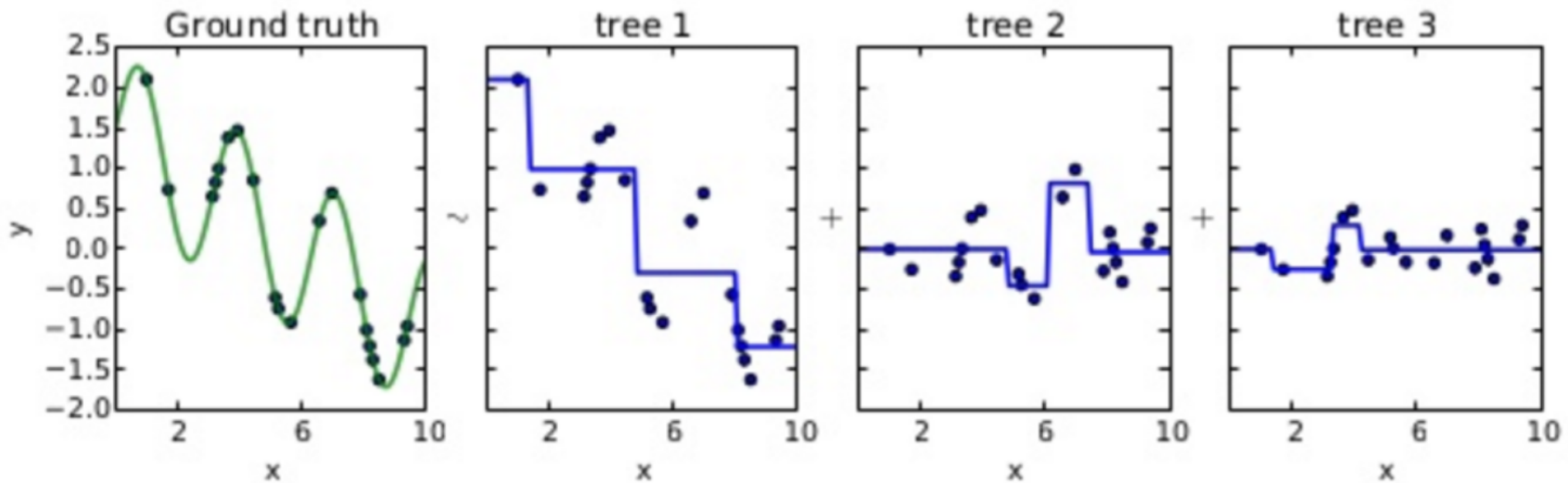
1. Set $\hat{f}^{(0)}(\boldsymbol{x}_i) = 0$ and $\hat{\epsilon}_i^{(1)} = Y_i$



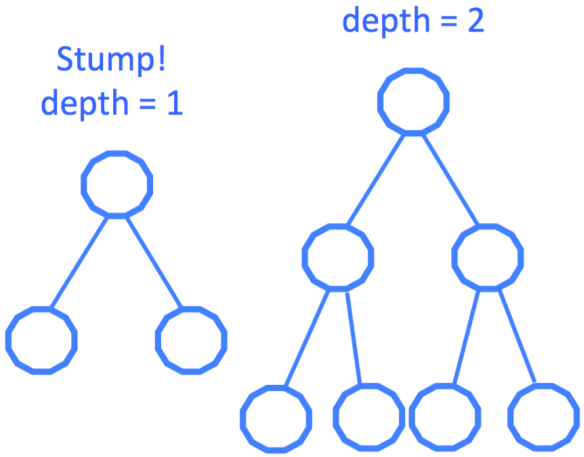
Fitting regressions on sequential residuals



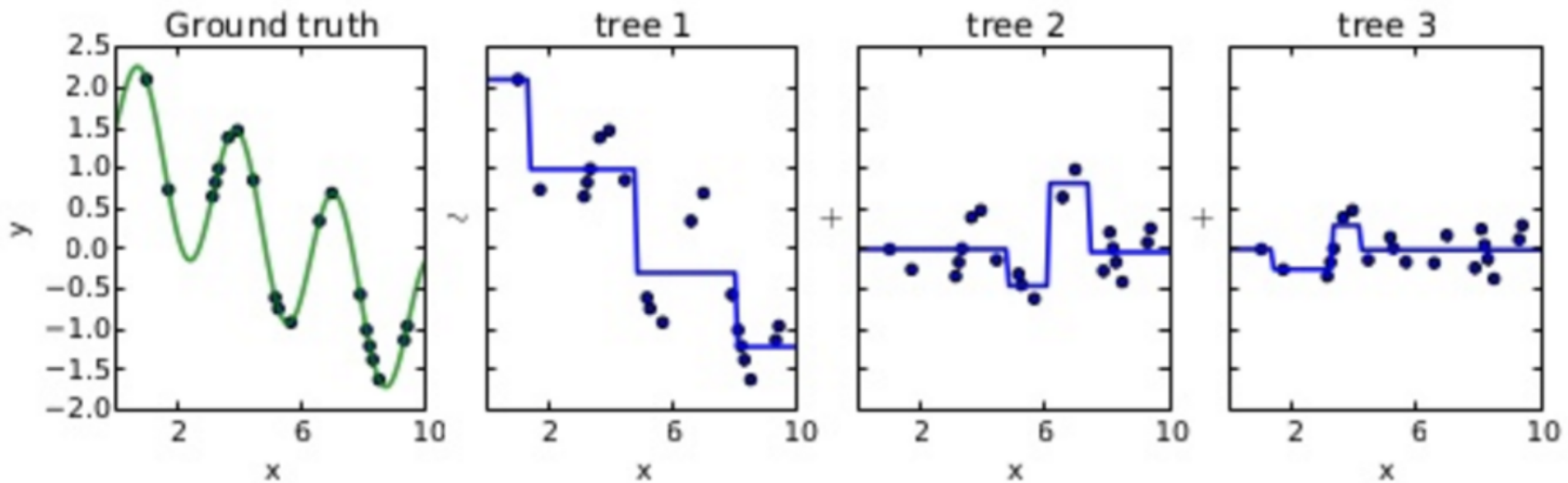
1. Set $\hat{f}^{(0)}(\boldsymbol{x}_i) = 0$ and $\hat{\epsilon}_i^{(1)} = Y_i$
2. For $k = 1, \cdots m$



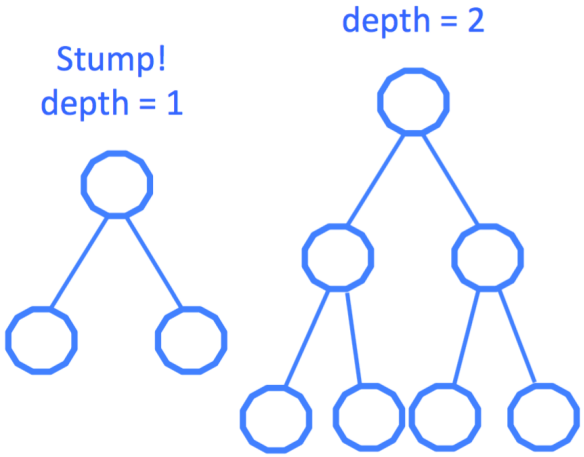
Fitting regressions on sequential residuals



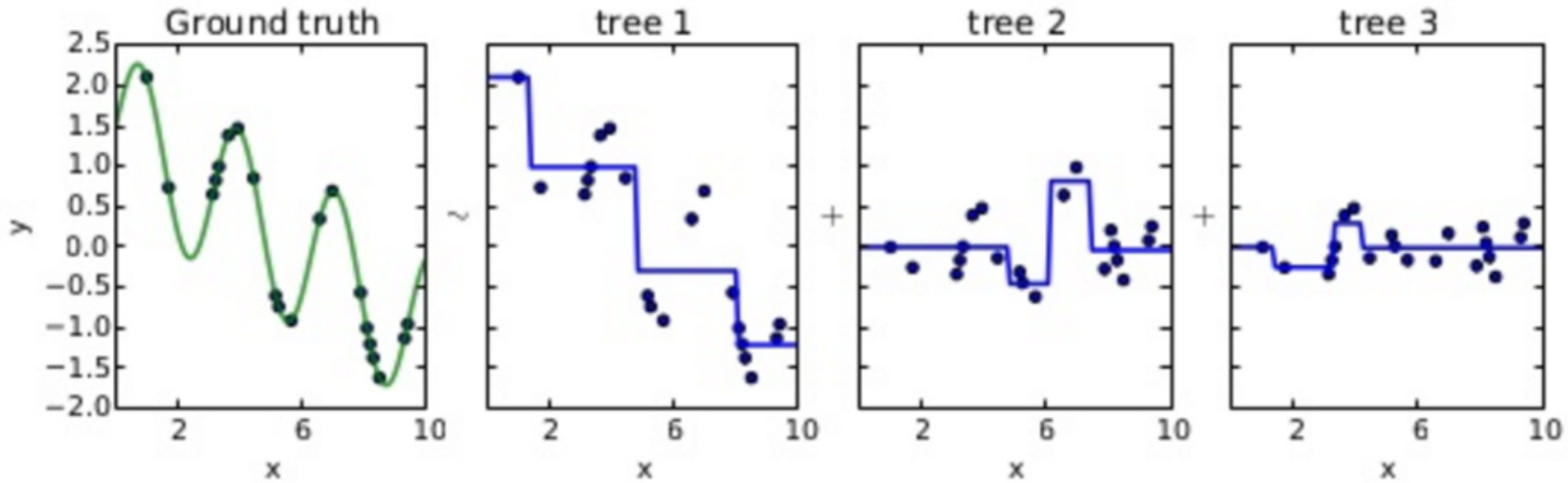
1. Set $\hat{f}^{(0)}(\boldsymbol{x}_i) = 0$ and $\hat{\epsilon}_i^{(1)} = Y_i$
2. For $k = 1, \cdots m$
   2.1 Fit a tree $\hat{f}^{(k)}$ to $\hat{\epsilon}^{(k)}$ using features $\boldsymbol{x}$



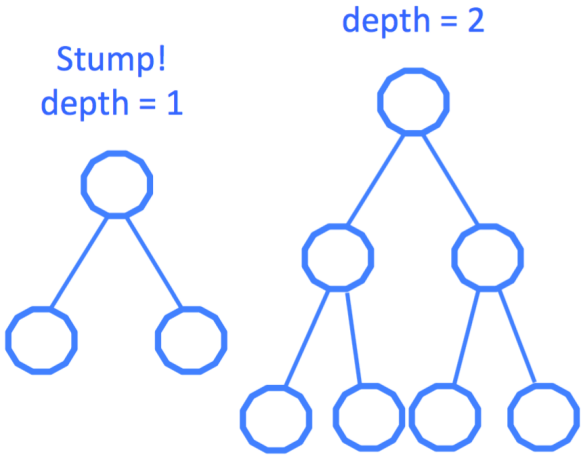
Fitting regressions on sequential residuals



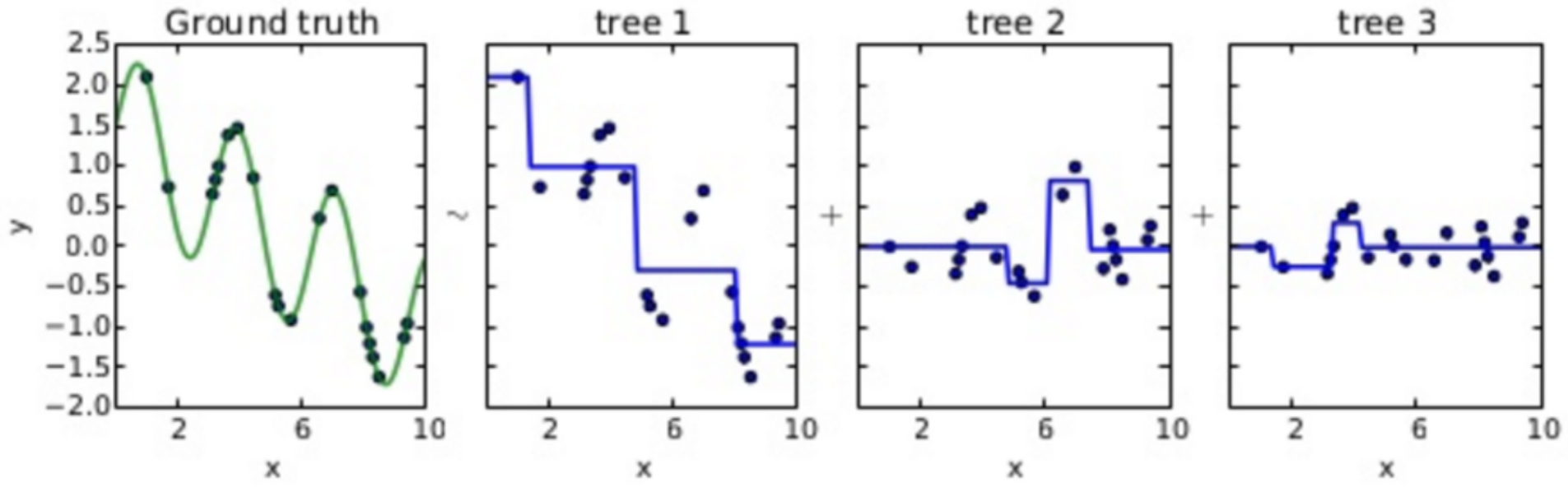
1. Set $\hat{f}^{(0)}(x_i) = 0$ and $\hat{\epsilon}_i^{(1)} = Y_i$
2. For $k = 1, \cdots m$
   - 2.1 Fit a tree $\hat{f}^{(k)}$ to $\hat{\epsilon}^{(k)}$ using features x
   - 2.2 Update the estimator $\hat{f}^{(k+1)} = f^{(k)} + \alpha_k f^{(k-1)}$



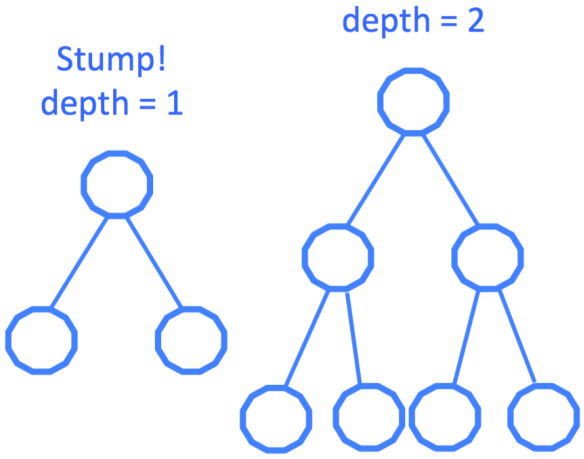
Fitting regressions on sequential residuals



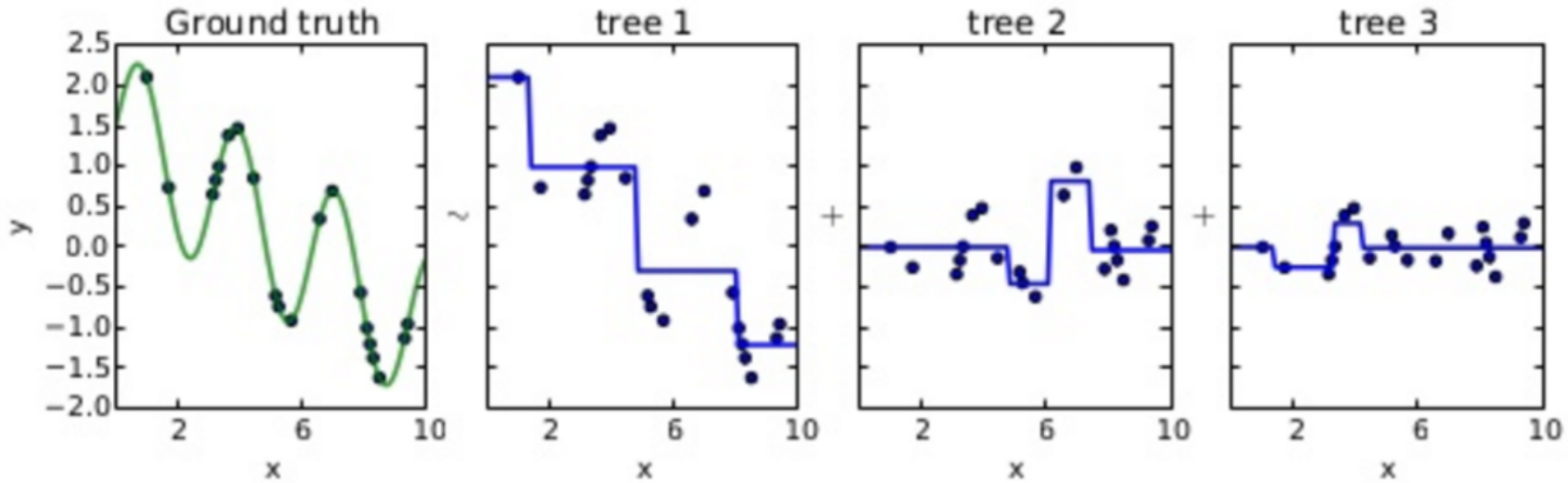
1. Set $\hat{f}^{(0)}(\boldsymbol{x}_i) = 0$ and $\hat{\epsilon}_i^{(1)} = Y_i$
2. For $k = 1, \cdots m$
   - 2.1 Fit a tree $\hat{f}^{(k)}$ to $\hat{\epsilon}^{(k)}$ using features $\boldsymbol{x}$
   - 2.2 Update the estimator $\hat{f}^{(k+1)} = f^{(k)} + \alpha_k f^{(k-1)}$
   - 2.3 Update the residuals $\hat{\epsilon}_i^{(k+1)} = \hat{\epsilon}_i^{(k)} - \alpha_k f^{(k)}(\boldsymbol{x}_i)$



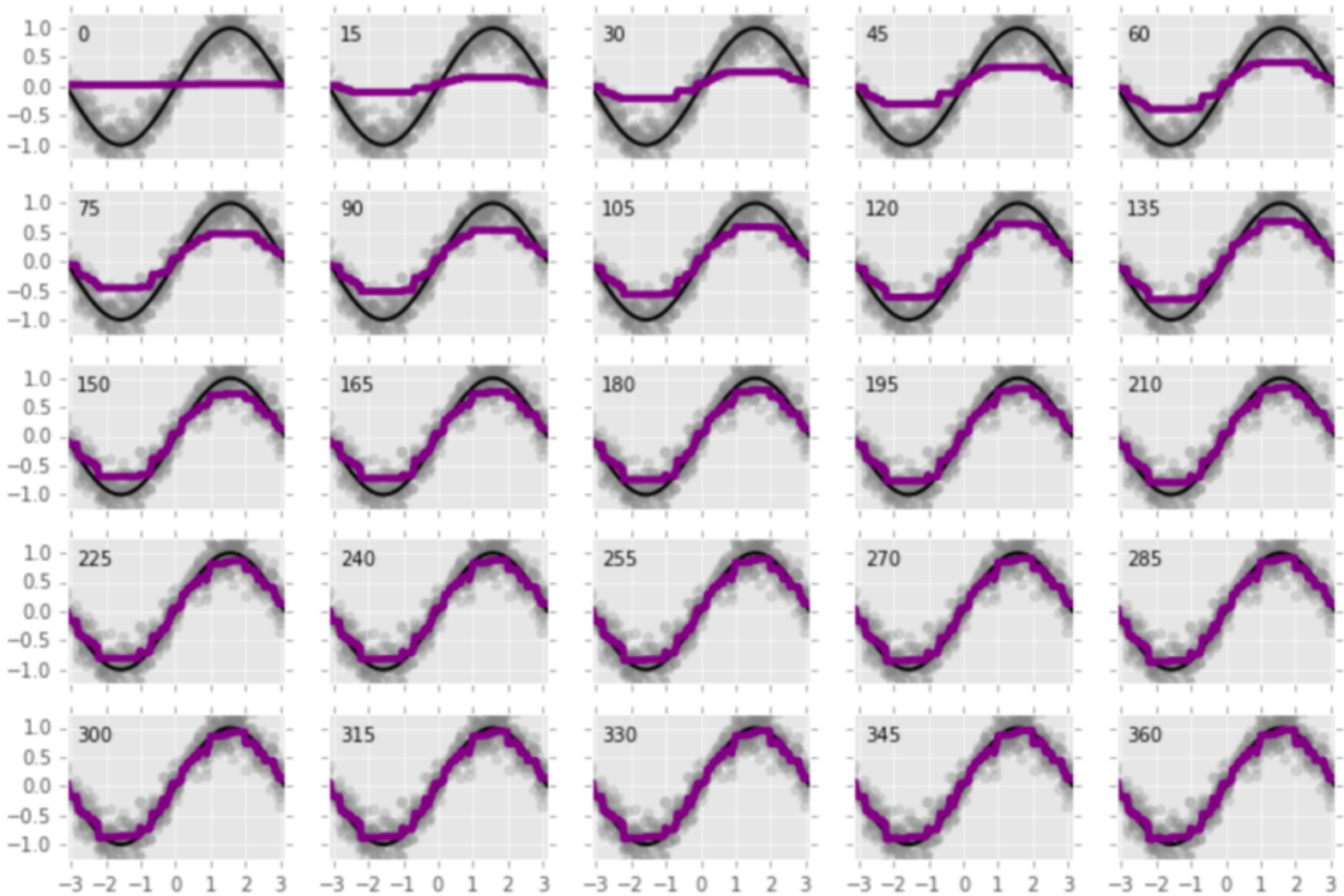
Fitting regressions on sequential residuals



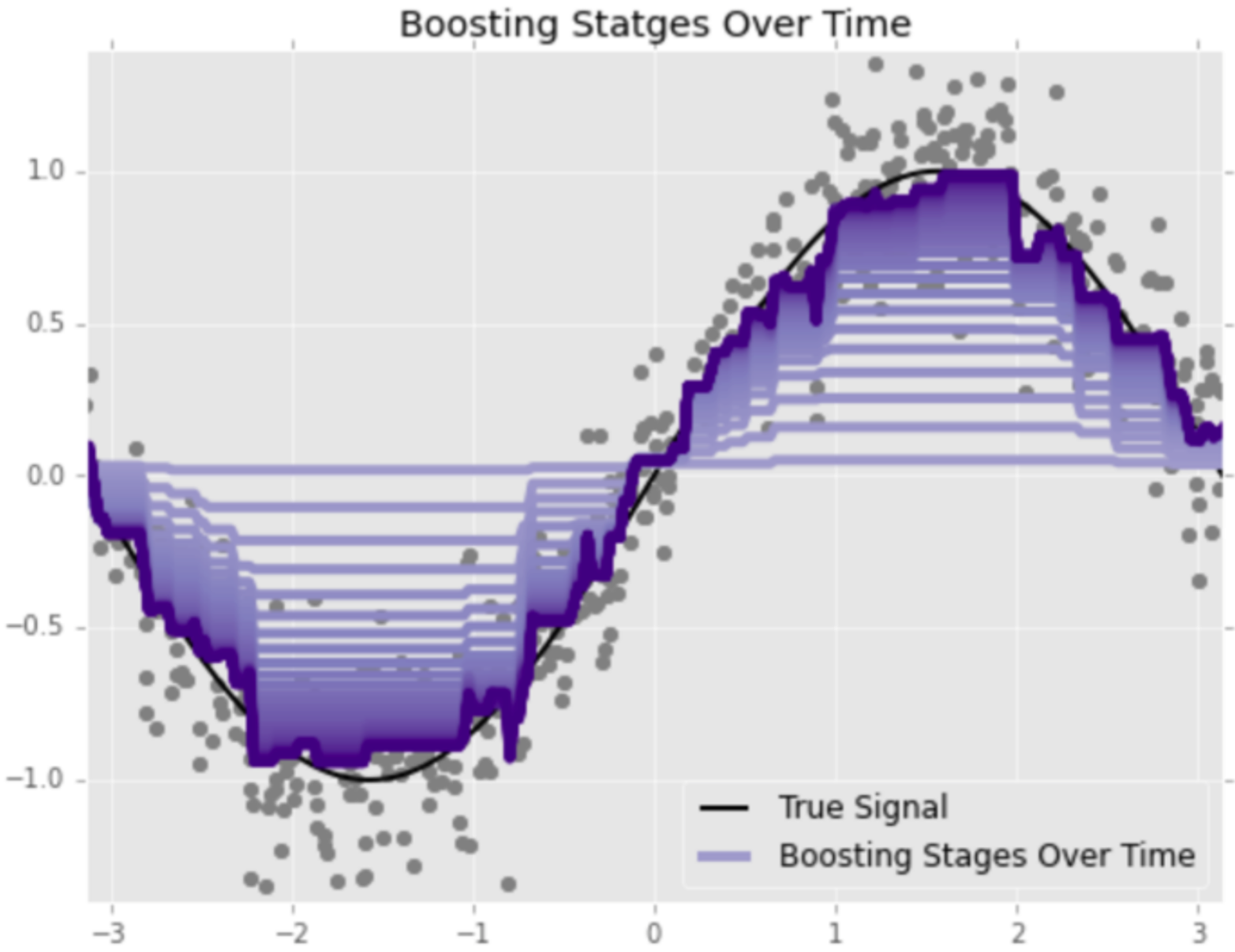
1. Set $\hat{f}^{(0)}(x_i) = 0$ and $\hat{\epsilon}_i^{(1)} = Y_i$
2. For $k = 1, \cdots m$
   - 2.1 Fit a tree $\hat{f}^{(k)}$ to $\hat{\epsilon}^{(k)}$ using features x
   - 2.2 Update the estimator $\hat{f}^{(k+1)} = f^{(k)} + \alpha_k f^{(k-1)}$
   - 2.3 Update the residuals $\hat{\epsilon}_i^{(k+1)} = \hat{\epsilon}_i^{(k)} - \alpha_k f^{(k)}(x_i)$
3. Return the boosted model $\hat{f}(x_i) = \sum_{k=1}^{m} \alpha_k f^{(k)}(x_i)$



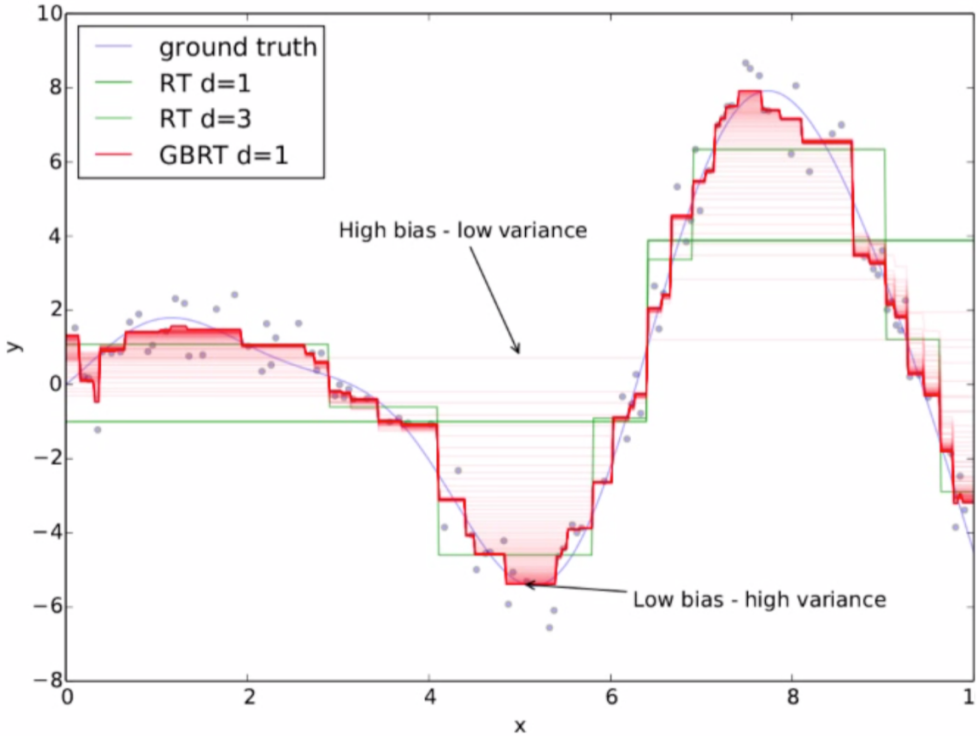
Fitting regressions on sequential residuals

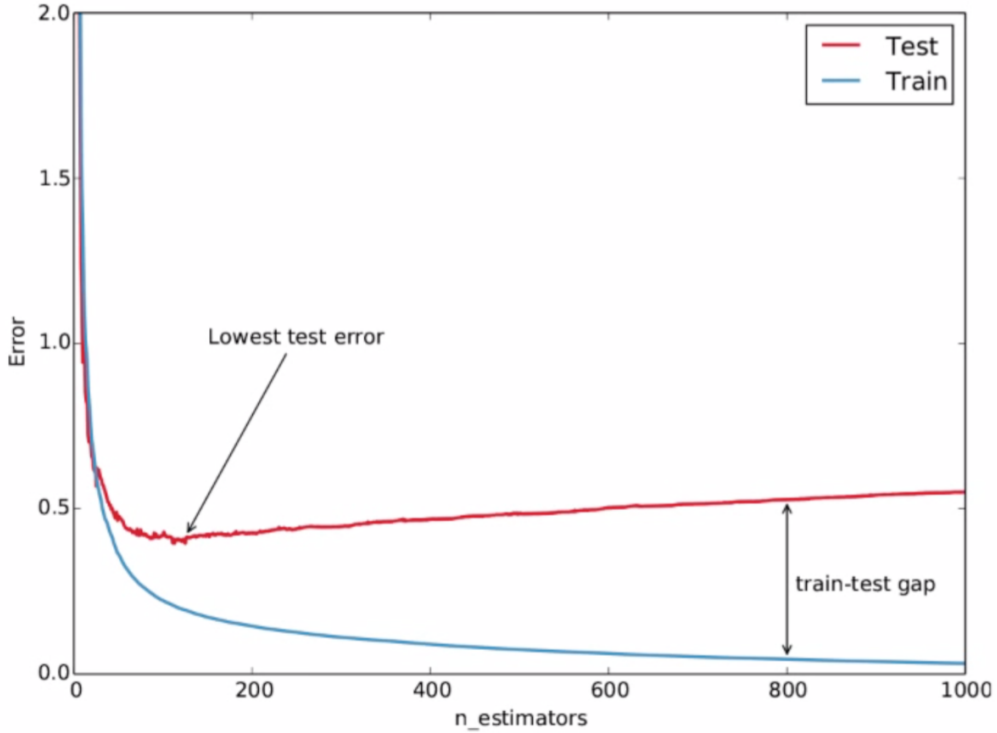


Fitting regressions on sequential residuals



Fitting regressions on sequential residuals





https://www.r-bloggers.com/an-attempt-to-understand-boosting-algorithms/

Loss Functions

- The *loss function* $L(Y_i, \hat{Y}_i)$ specifies prediction accuracy of $\hat{Y}_i$

Loss Functions

- The *loss function* $L(Y_i, \hat{Y}_i)$ specifies prediction accuracy of $\hat{Y}_i$
- The prediction $\hat{Y}_i$ can be changed

Loss Functions

- The *loss function* $L(Y_i, \hat{Y}_i)$ specifies prediction accuracy of $\hat{Y}_i$
- The prediction $\hat{Y}_i$ can be changed
- The gradient of the loss function with respect to $\hat{Y}_i$

$$\frac{\partial L(Y_i, \hat{Y}_i)}{\partial \hat{Y}_i}$$

gives the instantaneous Δ in $L(Y_i, \hat{Y}_i)$ at $\hat{Y}_i \longrightarrow \hat{Y}_i + \epsilon$

Loss Functions

- The *loss function* $L(Y_i, \hat{Y}_i)$ specifies prediction accuracy of $\hat{Y}_i$
- The prediction $\hat{Y}_i$ can be changed
- The gradient of the loss function with respect to $\hat{Y}_i$

$$\frac{\partial L(Y_i, \hat{Y}_i)}{\partial \hat{Y}_i}$$

gives the instantaneous Δ in $L(Y_i, \hat{Y}_i)$ at $\hat{Y}_i \longrightarrow \hat{Y}_i + \epsilon$

I.e., it tells how the loss function changes as $\hat{Y}_i$ increases

(this is a very simple idea with somewhat complex notation)

Gradient Decent

- But if we have partial derivatives we can use gradient decent

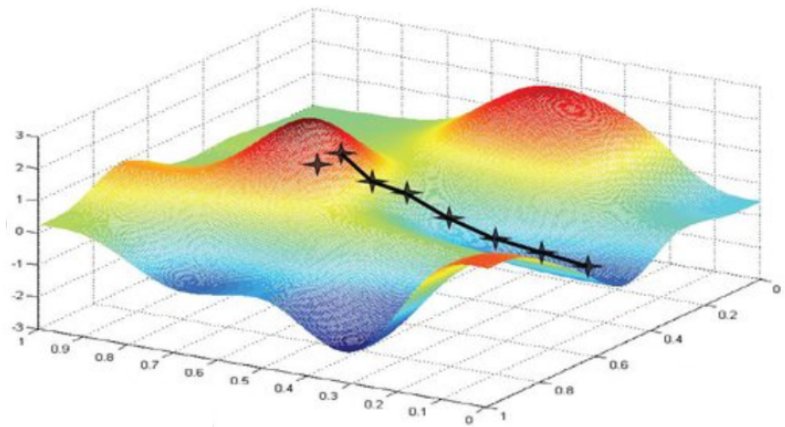
$$\nabla_{\hat{\boldsymbol{Y}}} L(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) = \begin{bmatrix} \frac{\partial L(Y_1, \hat{Y}_1)}{\partial \hat{Y}_1} \\ \frac{\partial L(Y_2, \hat{Y}_2)}{\partial \hat{Y}_2} \\ \vdots \\ \frac{\partial L(Y_i, \hat{Y}_i)}{\partial \hat{Y}_i} \\ \vdots \\ \frac{\partial L(Y_n, \hat{Y}_i)}{\partial \hat{Y}_n} \end{bmatrix}$$

to move in the direction of greatest decrease in loss

Gradient Decent

- But if we have partial derivatives we can use gradient decent

$$\nabla_{\hat{\boldsymbol{Y}}} L(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) = \begin{bmatrix} \frac{\partial L(Y_1, \hat{Y}_1)}{\partial \hat{Y}_1} \\ \frac{\partial L(Y_2, \hat{Y}_2)}{\partial \hat{Y}_2} \\ \vdots \\ \frac{\partial L(Y_i, \hat{Y}_i)}{\partial \hat{Y}_i} \\ \vdots \\ \frac{\partial L(Y_n, \hat{Y}_i)}{\partial \hat{Y}_n} \end{bmatrix}$$



The direction of maximal increase is the the gradient of a function

(The negative of the gradient is the direction of maximal decrease)

to move in the direction of greatest decrease in loss

Gradient Boosting

- If we take $\hat{Y}_i$ to be our current prediction $\phi_k(\boldsymbol{x}_i)$

Gradient Boosting

- If we take $\hat{Y}_i$ to be our current prediction $\phi_k(\boldsymbol{x}_i)$, the negative partial derivative of the loss function with respect to $\phi_k(\boldsymbol{x}_i)$

$$\Delta_i^k = -\frac{\partial L(Y_i, \phi_k(\boldsymbol{x}_i))}{\partial \phi_k(\boldsymbol{x}_i)}$$

Gradient Boosting

- If we take $\hat{Y}_i$ to be our current prediction $\phi_k(\boldsymbol{x}_i)$, the negative partial derivative of the loss function with respect to $\phi_k(\boldsymbol{x}_i)$

$$\Delta_i^k = -\frac{\partial L(Y_i, \phi_k(\boldsymbol{x}_i))}{\partial \phi_k(\boldsymbol{x}_i)}$$

  is the direction of greatest decrease in cost for $\hat{Y}_i$

Gradient Boosting

- If we take $\hat{Y}_i$ to be our current prediction $\phi_k(\boldsymbol{x}_i)$, the negative partial derivative of the loss function with respect to $\phi_k(\boldsymbol{x}_i)$

$$\Delta_i^k = -\frac{\partial L(Y_i, \phi_k(\boldsymbol{x}_i))}{\partial \phi_k(\boldsymbol{x}_i)}$$

  is the direction of greatest decrease in cost for $\hat{Y}_i$

$\Longrightarrow$ We should update $\phi_k(\boldsymbol{x}_i) \to \phi_{k+1}(\boldsymbol{x}_i)$ in that direction

Gradient Boosting

- If we take $\hat{Y}_i$ to be our current prediction $\phi_k(\boldsymbol{x}_i)$, the negative partial derivative of the loss function with respect to $\phi_k(\boldsymbol{x}_i)$

$$\Delta_i^k = -\frac{\partial L(Y_i, \phi_k(\boldsymbol{x}_i))}{\partial \phi_k(\boldsymbol{x}_i)}$$

  is the direction of greatest decrease in cost for $\hat{Y}_i$

$\Longrightarrow$ We should update $\phi_k(\boldsymbol{x}_i) \to \phi_{k+1}(\boldsymbol{x}_i)$ in that direction

- But $\hat{\boldsymbol{Y}} = \{\hat{Y}_1, \hat{Y}_2, \cdots \hat{Y}_n\}$ so we compromise between all Δ_i^k

$$\hat{W}(\cdot, \gamma_k) = \underset{W(\cdot,\gamma_k)}{\operatorname{argmin}} \sum_{i=1}^{n} D(\Delta_i^k, W(\boldsymbol{x}_i, \gamma_k))$$

  and set

$$\phi_{k+1}(\boldsymbol{x}) = \phi_k(\boldsymbol{x}) + \alpha_k \hat{W}(\boldsymbol{x}, \gamma_k)$$

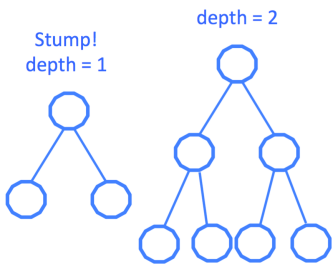
  where $\hat{W}(\cdot, \gamma_k)$ is a *weak learner* boosted at *learning rate* α_k

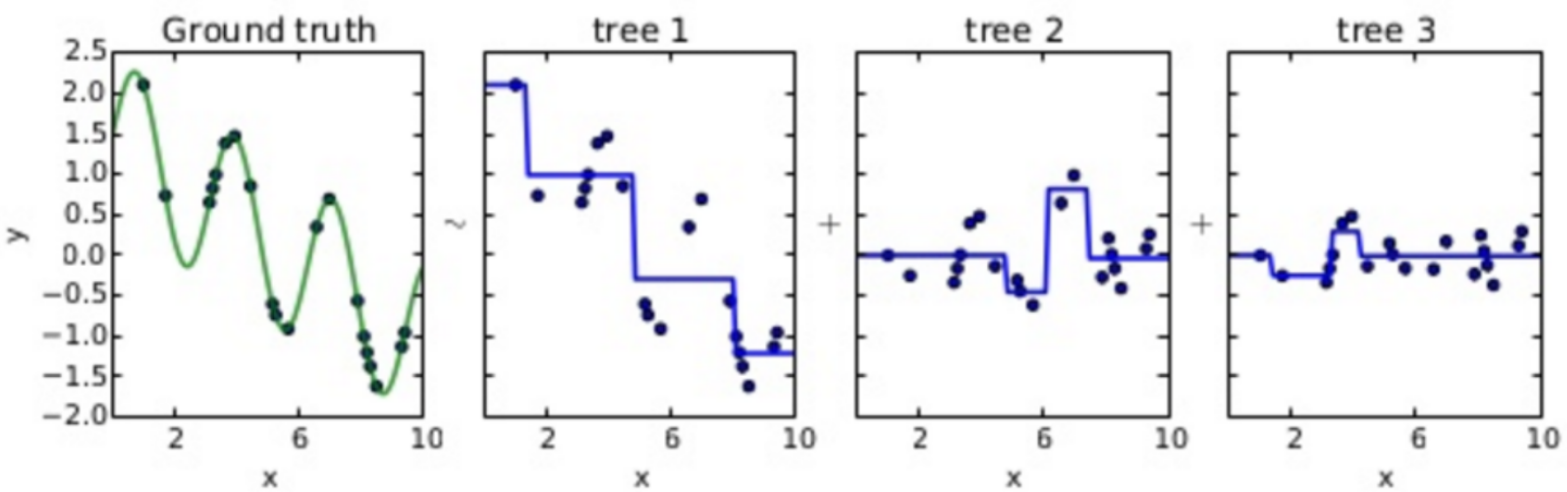
Fitting regressions on sequential residuals

- if we use squared error loss

$$L(Y_i, \phi_k(\boldsymbol{x}_i)) = \frac{1}{2}(Y_i - \phi_k(\boldsymbol{x}_i))^2$$

then the direction of the gradient is vector ϵ_i of residuals



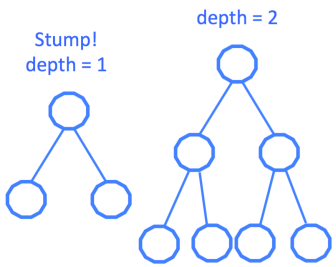


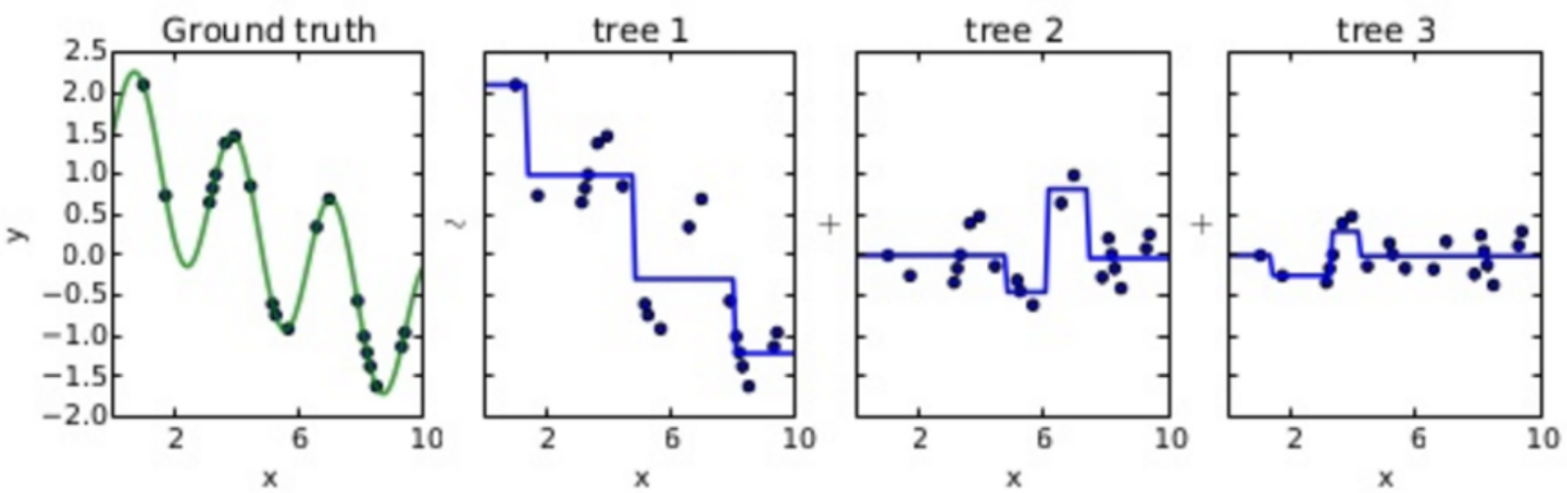
Fitting regressions on sequential residuals

- if we use squared error loss

$$L(Y_i, \phi_k(\boldsymbol{x}_i)) = \frac{1}{2}(Y_i - \phi_k(\boldsymbol{x}_i))^2$$

then the direction of the gradient is vector ϵ_i of residuals





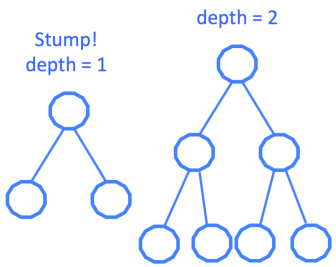
- Moving from $\phi_k(\boldsymbol{x}_i)$ to $\phi_k(\boldsymbol{x}_i) + \epsilon_i$ would be zero loss

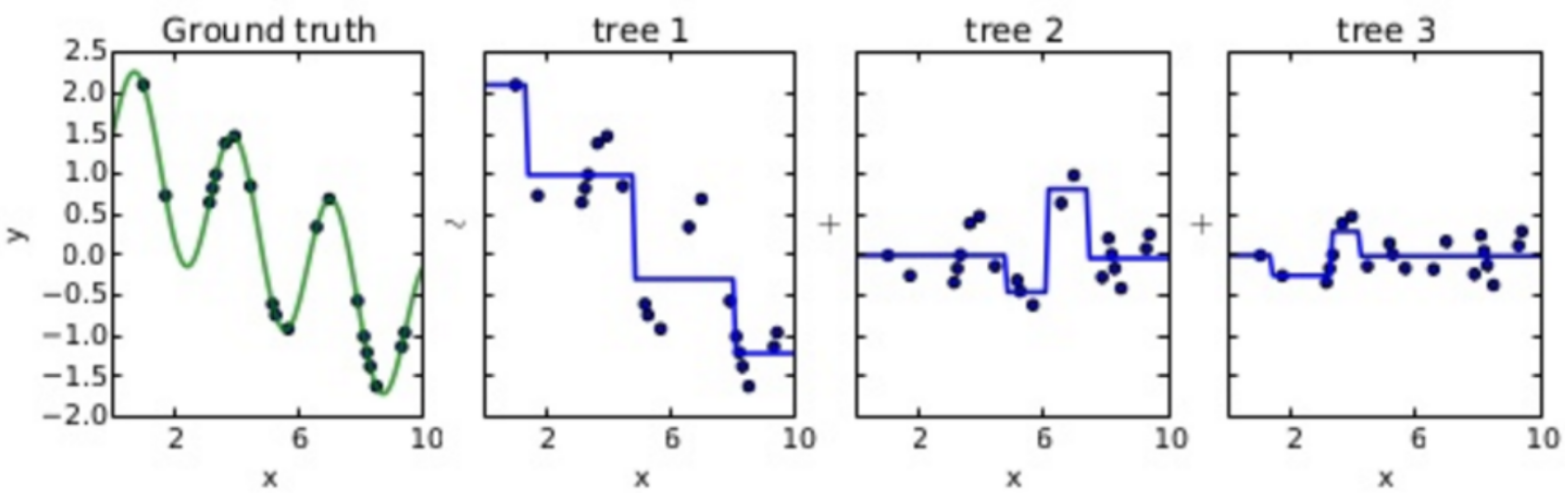
Fitting regressions on sequential residuals

- if we use squared error loss

$$L(Y_i, \phi_k(\boldsymbol{x}_i)) = \frac{1}{2}(Y_i - \phi_k(\boldsymbol{x}_i))^2$$

then the direction of the gradient is vector ϵ_i of residuals





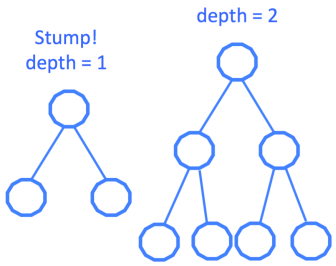
- Moving from $\phi_k(\boldsymbol{x}_i)$ to $\phi_k(\boldsymbol{x}_i) + \epsilon_i$ would be zero loss
- But we **don't** move exactly to $\phi_k(\boldsymbol{x}_i) + \epsilon_i$ because

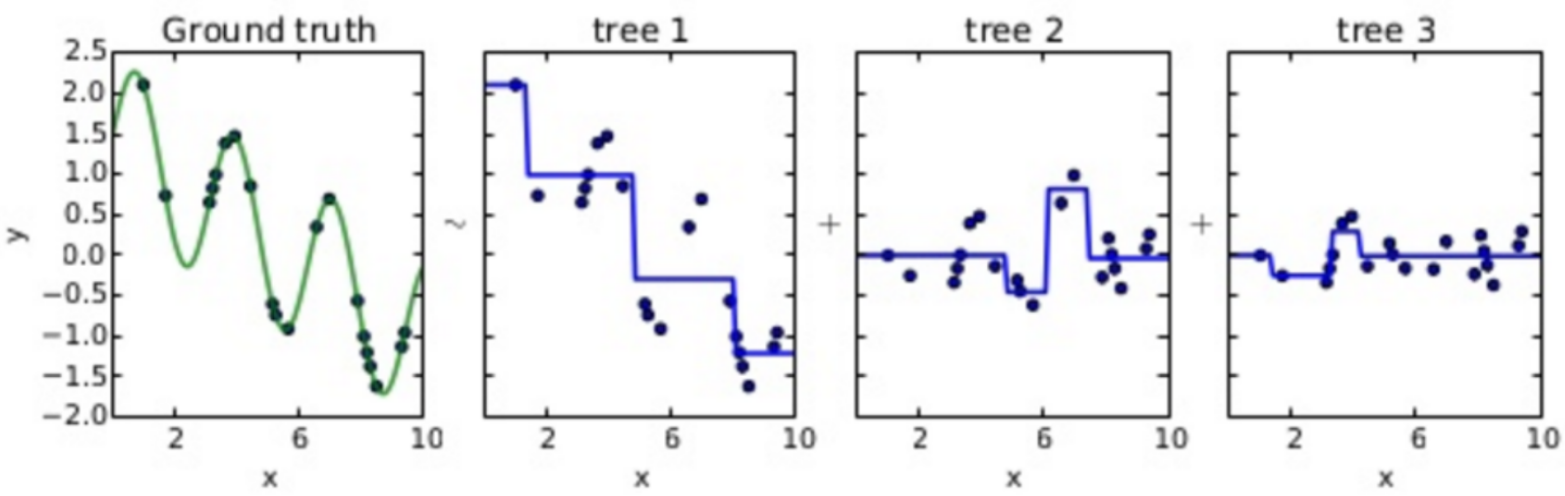
Fitting regressions on sequential residuals

- if we use squared error loss

$$L(Y_i, \phi_k(\boldsymbol{x}_i)) = \frac{1}{2}(Y_i - \phi_k(\boldsymbol{x}_i))^2$$

then the direction of the gradient is vector ϵ_i of residuals





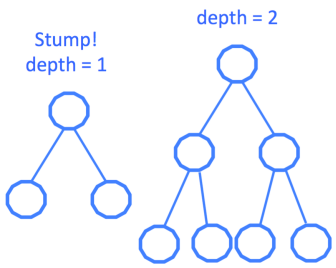
- Moving from $\phi_k(\boldsymbol{x}_i)$ to $\phi_k(\boldsymbol{x}_i) + \epsilon_i$ would be zero loss
- But we **don't** move exactly to $\phi_k(\boldsymbol{x}_i) + \epsilon_i$ because
  - $W(\boldsymbol{x}, \gamma_k)$ is a weak learner and probably can't, and

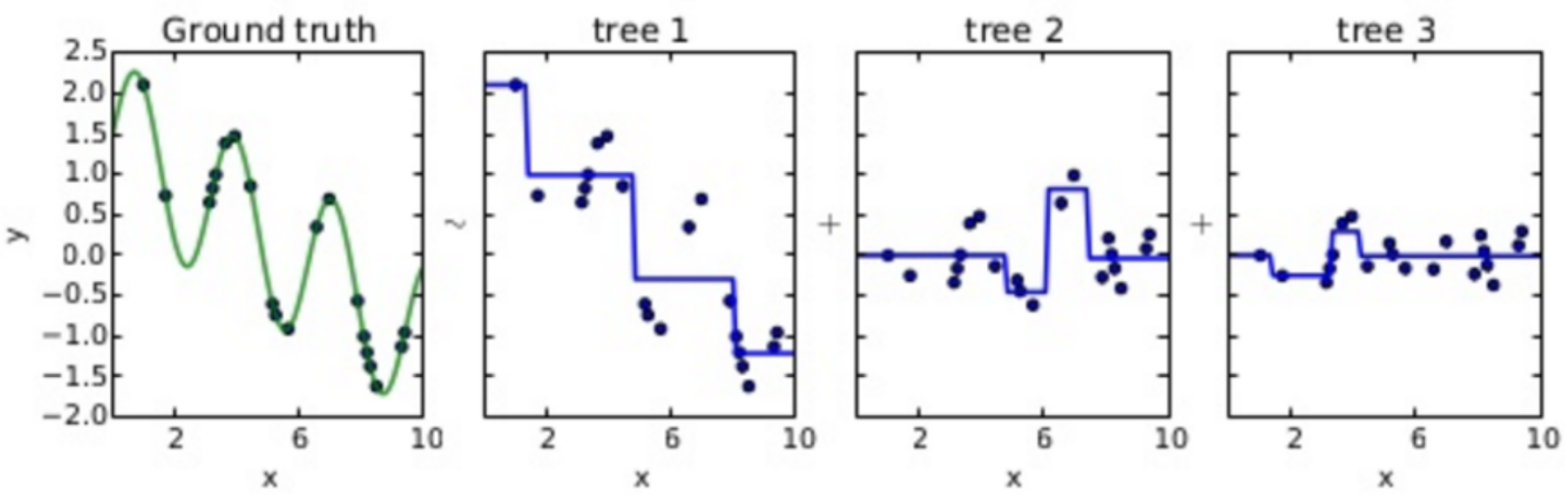
Fitting regressions on sequential residuals

- if we use squared error loss

$$L(Y_i, \phi_k(\boldsymbol{x}_i)) = \frac{1}{2}(Y_i - \phi_k(\boldsymbol{x}_i))^2$$

then the direction of the gradient is vector ϵ_i of residuals





- Moving from $\phi_k(\boldsymbol{x}_i)$ to $\phi_k(\boldsymbol{x}_i) + \epsilon_i$ would be zero loss
- But we **don't** move exactly to $\phi_k(\boldsymbol{x}_i) + \epsilon_i$ because
  - $W(\boldsymbol{x}, \gamma_k)$ is a weak learner and probably can't, and
  - we only move to $\phi_k(\boldsymbol{x}) + \alpha_k \hat{W}(\boldsymbol{x}, \gamma_k)$ at learning rate α_k

Some loss functions

Regression

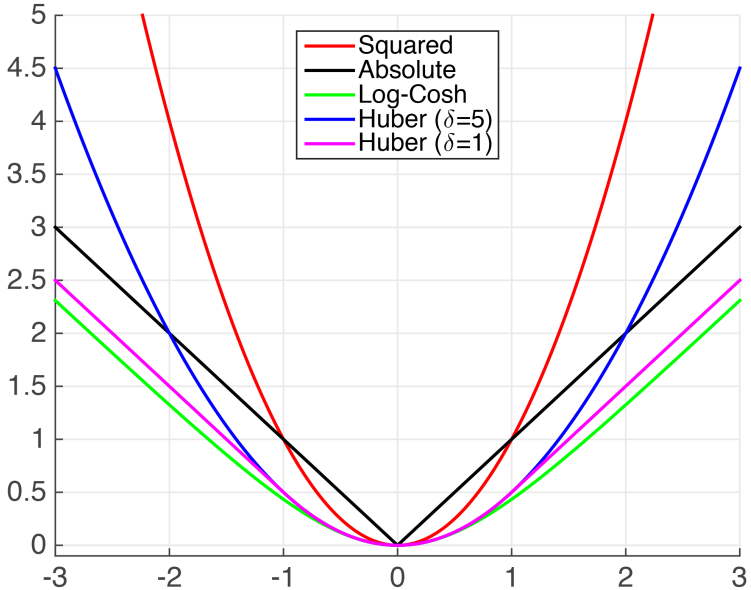
Squared Loss $\frac{1}{2}(y_i - \boldsymbol{x}_i^T\beta)^2$

Absolute Loss $|Y_i - \boldsymbol{x}_i^T\beta|$

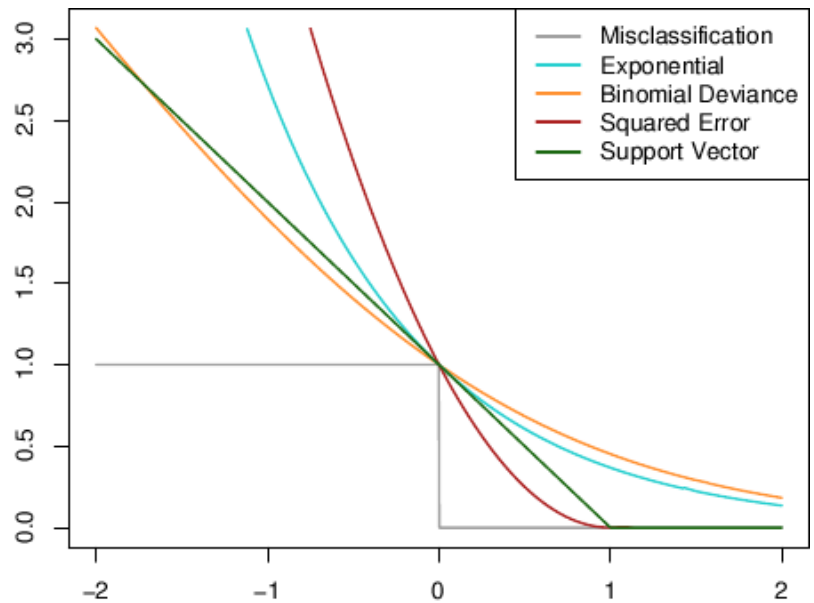
Classification

Log Loss $1 - \frac{1}{1+e^{-x_i^T\beta}}$ (Bernoulli Deviance/Logistic Reg.)

Exponential Loss $\exp(-y_i \cdot \boldsymbol{x}_i^T\beta)$ (Ada Boost)



$\boldsymbol{x}_i^T\beta$



$\boldsymbol{x}_i^T\beta$

AdaBoost (Adaptive Boosting)

$Y \in \{-1, 1\}$

1. Set weights $w_i^{(1)} = \frac{1}{n}$

AdaBoost (Adaptive Boosting)

$Y \in \{-1, 1\}$

1. Set weights $w_i^{(1)} = \frac{1}{n}$
2. For $k = 1, \cdots m$

AdaBoost (Adaptive Boosting)

$Y \in \{-1, 1\}$

1. Set weights $w_i^{(1)} = \frac{1}{n}$
2. For $k = 1, \cdots m$

   2.1 Fit $W(\cdot, \gamma_k)$ to the weighted data using weights $w_i^{(k)}$

AdaBoost (Adaptive Boosting)

$Y \in \{-1, 1\}$

1. Set weights $w_i^{(1)} = \frac{1}{n}$
2. For $k = 1, \cdots m$

   2.1 Fit $W(\cdot, \gamma_k)$ to the weighted data using weights $w_i^{(k)}$

   2.2 Compute error $\hat{\epsilon}_k = \frac{\sum_{Y_i \neq W(\boldsymbol{x}_i, \gamma_k)} w_i^{(k)}}{\sum w_i^{(k)}}$

AdaBoost (Adaptive Boosting)

$Y \in \{-1, 1\}$

1. Set weights $w_i^{(1)} = \frac{1}{n}$
2. For $k = 1, \cdots m$

   2.1 Fit $W(\cdot, \gamma_k)$ to the weighted data using weights $w_i^{(k)}$

   2.2 Compute error $\hat{\epsilon}_k = \frac{\sum_{Y_i \neq W(\boldsymbol{x}_i, \gamma_k)} w_i^{(k)}}{\sum w_i^{(k)}}$

   2.3 Compute $\alpha_k = log\left(\frac{1}{\hat{\epsilon}_k} - 1\right)$

AdaBoost (Adaptive Boosting)

$Y \in \{-1, 1\}$

1. Set weights $w_i^{(1)} = \frac{1}{n}$
2. For $k = 1, \cdots m$
   - 2.1 Fit $W(\cdot, \gamma_k)$ to the weighted data using weights $w_i^{(k)}$
   - 2.2 Compute error $\hat{\epsilon}_k = \frac{\sum_{Y_i \neq W(x_i, \gamma_k)} w_i^{(k)}}{\sum w_i^{(k)}}$
   - 2.3 Compute $\alpha_k = log\left(\frac{1}{\hat{\epsilon}_k} - 1\right)$
   - 2.4 Set $w_i^{(k+1)} = w_i^{(k)} e^{\alpha_k 1_{[Y_i \neq W(x_i, \gamma_k)]}}$

AdaBoost (Adaptive Boosting)

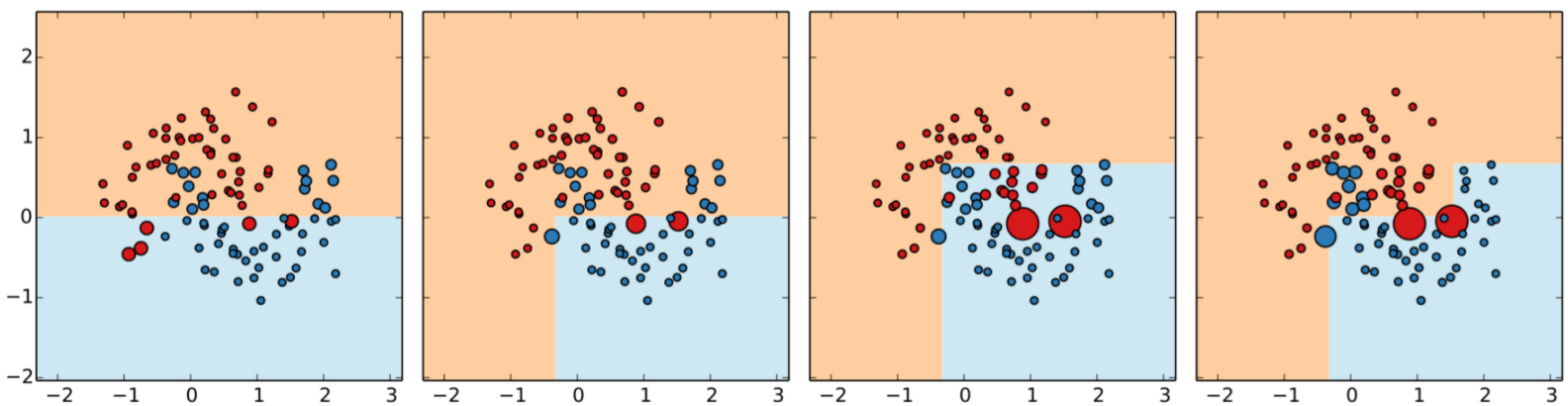
$Y \in \{-1, 1\}$

1. Set weights $w_i^{(1)} = \frac{1}{n}$
2. For $k = 1, \cdots m$
   - 2.1 Fit $W(\cdot, \gamma_k)$ to the weighted data using weights $w_i^{(k)}$
   - 2.2 Compute error $\hat{\epsilon}_k = \frac{\sum_{Y_i \neq W(\boldsymbol{x}_i, \gamma_k)} w_i^{(k)}}{\sum w_i^{(k)}}$
   - 2.3 Compute $\alpha_k = log\left(\frac{1}{\hat{\epsilon}_k} - 1\right)$
   - 2.4 Set $w_i^{(k+1)} = w_i^{(k)} e^{\alpha_k 1_{[Y_i \neq W(\boldsymbol{x}_i, \gamma_k)]}}$
3. Return $G(\boldsymbol{x}_i) = \text{sign}\left(\sum_{k=1}^{m} \alpha_k W(\boldsymbol{x}_i, \gamma_k)\right)$

AdaBoost (Adaptive Boosting)

$Y \in \{-1, 1\}$

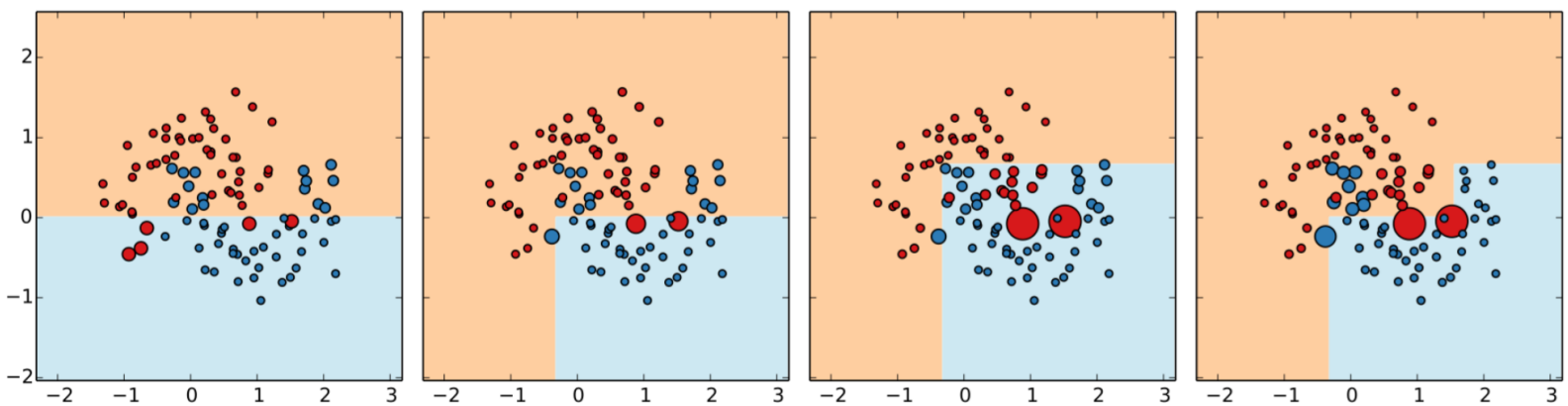
1. Set weights $w_i^{(1)} = \frac{1}{n}$
2. For $k = 1, \cdots m$
   - 2.1 Fit $W(\cdot, \gamma_k)$ to the weighted data using weights $w_i^{(k)}$
   - 2.2 Compute error $\hat{\epsilon}_k = \frac{\sum_{Y_i \neq W(\boldsymbol{x}_i, \gamma_k)} w_i^{(k)}}{\sum w_i^{(k)}}$
   - 2.3 Compute $\alpha_k = log\left(\frac{1}{\hat{\epsilon}_k} - 1\right)$
   - 2.4 Set $w_i^{(k+1)} = w_i^{(k)} e^{\alpha_k 1_{[Y_i \neq W(\boldsymbol{x}_i, \gamma_k)]}}$
3. Return $G(\boldsymbol{x}_i) = \text{sign}\left(\sum_{k=1}^{m} \alpha_k W(\boldsymbol{x}_i, \gamma_k)\right)$



AdaBoost (Adaptive Boosting)

$Y \in \{-1, 1\}$

1. Set weights $w_i^{(1)} = \frac{1}{n}$
2. For $k = 1, \cdots m$
   - 2.1 Fit $W(\cdot, \gamma_k)$ to the weighted data using weights $w_i^{(k)}$
   - 2.2 Compute error $\hat{\epsilon}_k = \frac{\sum_{Y_i \neq W(\boldsymbol{x}_i, \gamma_k)} w_i^{(k)}}{\sum w_i^{(k)}}$
   - 2.3 Compute $\alpha_k = log\left(\frac{1}{\hat{\epsilon}_k} - 1\right)$
   - 2.4 Set $w_i^{(k+1)} = w_i^{(k)} e^{\alpha_k 1_{[Y_i \neq W(\boldsymbol{x}_i, \gamma_k)]}}$
3. Return $G(\boldsymbol{x}_i) = \text{sign}\left(\sum_{k=1}^{m} \alpha_k W(\boldsymbol{x}_i, \gamma_k)\right)$



*Sean Sall's slides have an appendix showing that AdaBoost is exponential loss

XGBoost (eXtreme Gradient Boosting)

- Percentiles binned features: faster/more efficient splitting
- Handles missing data
- Smart memory management/out-of-core computation

`http://xgboost.readthedocs.io`

- Knobs
- Get Started (Python Install)
- Packages $>$ Python $>$ Python API Reference
  - Core Data Structure
  - Learning API
  - Scikit-Learn API