



NATIONAL RESEARCH  
UNIVERSITY



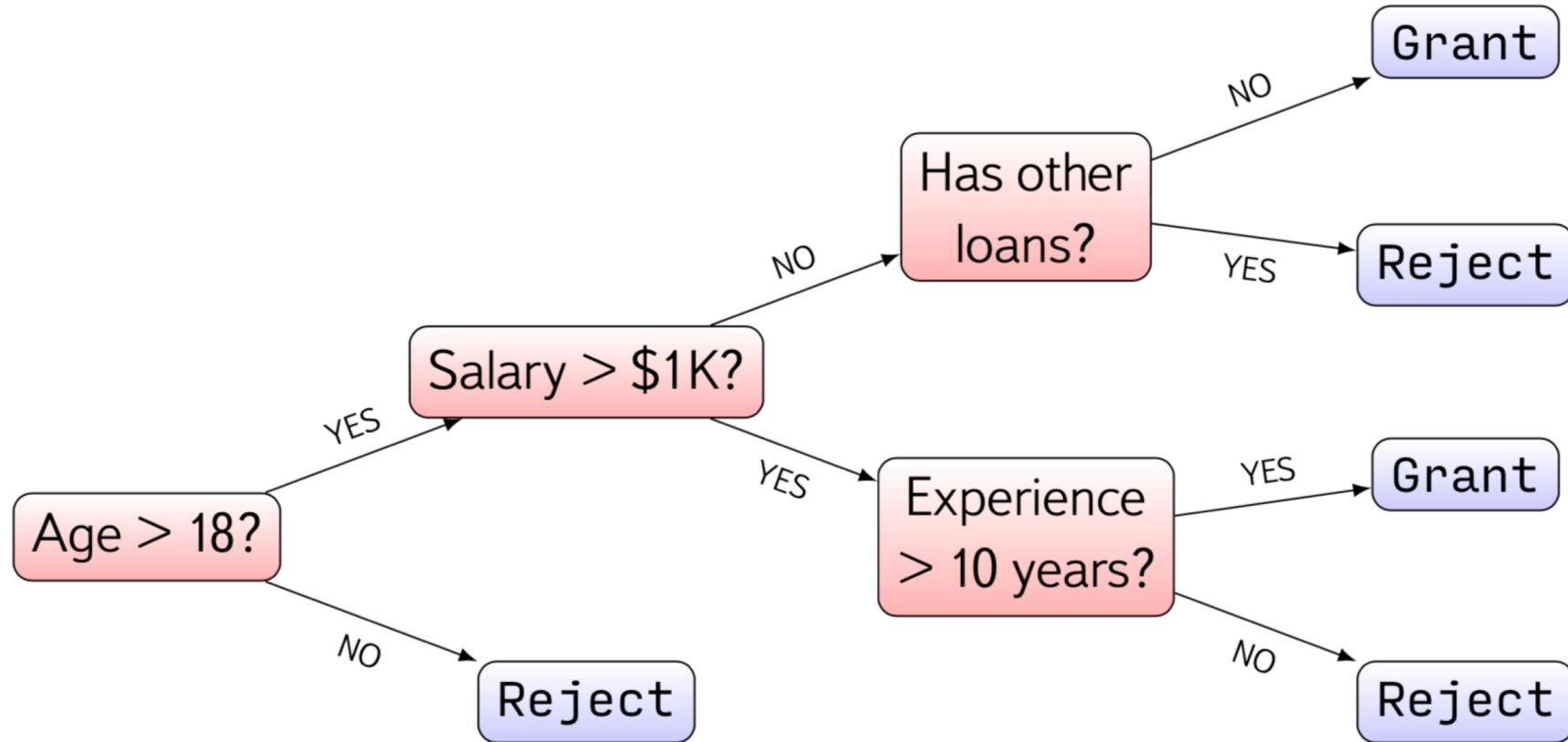
# Decision trees, Bagging and Random Forests, Stacking, Gradient Boosting Models

Sergey Sviridov

\* Slides are courtesy of Andrey Ustyuzhanin and Alexey Artemov

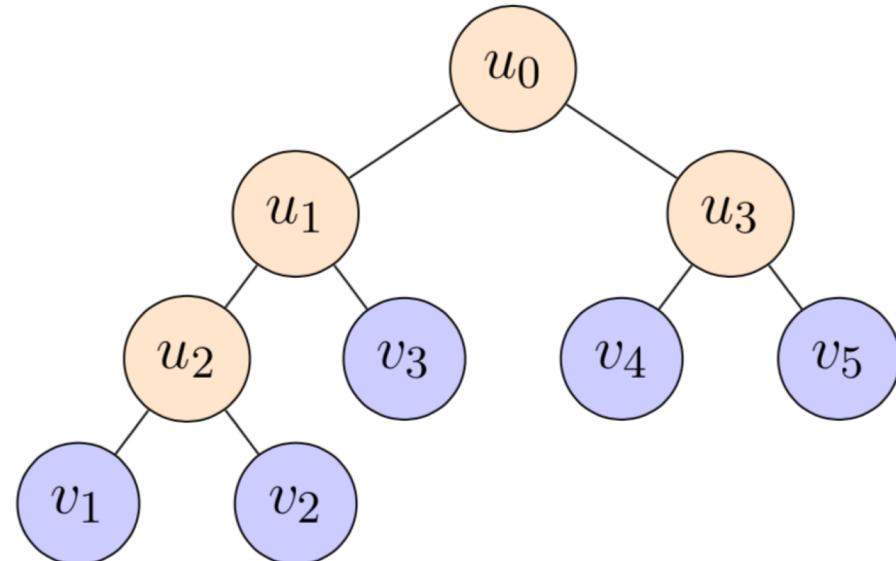
# Decision Trees

# Decision making at a bank



# Decision tree formalism

- › Decision tree is a binary tree  $V$
- › Internal nodes  $u \in V$ : predicates  
 $\beta_u : \mathbb{X} \rightarrow \{0, 1\}$
- › Leafs  $v \in V$ : predictions  $x$
- › Algorithm  $h(\mathbf{x})$  starts at  $u = u_0$ 
  - › Compute  $b = \beta_u(\mathbf{x})$
  - › If  $b = 0$ ,  $u \leftarrow \text{LeftChild}(u)$
  - › If  $b = 1$ ,  $u \leftarrow \text{RightChild}(u)$
  - › If  $u$  is a leaf, return  $b$
- › In practice:  $\beta_u(\mathbf{x}; j, t) = [\mathbf{x}_j < t]$



# Greedy tree learning for binary classification

› Input: training set  $X^\ell = \{(\mathbf{x}_i, y_i)\}_{i=1}^\ell$

1. Greedily split  $X^\ell$  into  $R_1$  and  $R_2$ :

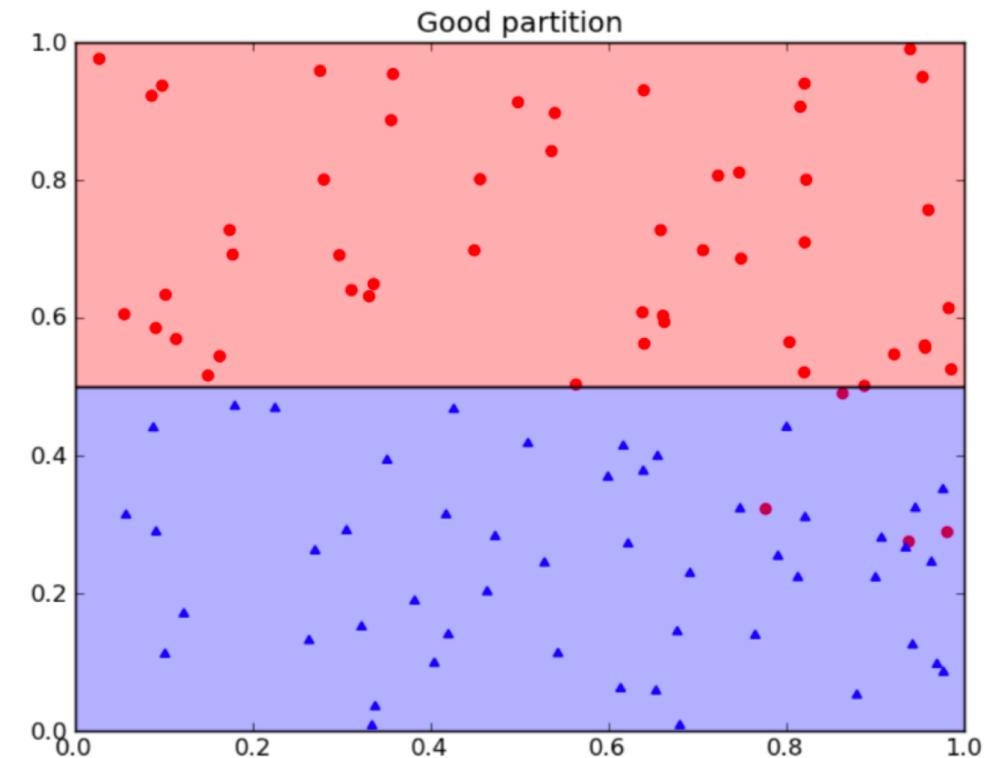
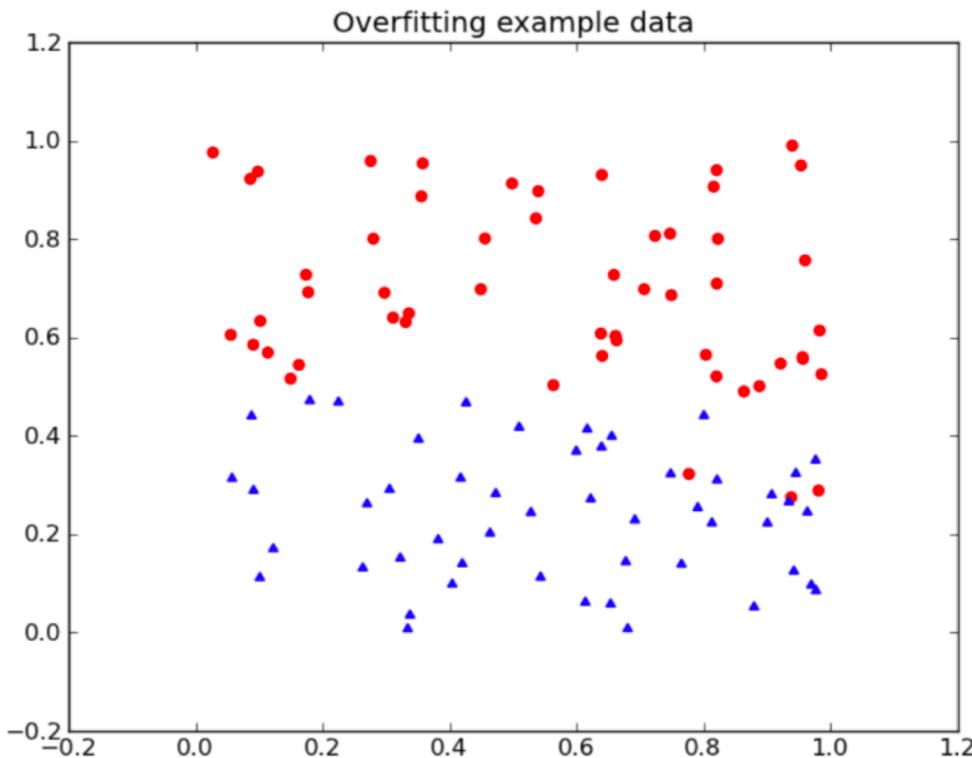
$$R_1(j, t) = \{\mathbf{x} \in X^\ell | \mathbf{x}_j < t\}, \quad R_2(j, t) = \{\mathbf{x} \in X^\ell | \mathbf{x}_j > t\}$$

optimizing a given loss:  $Q(X^\ell, j, t) \rightarrow \min_{(j,t)}$

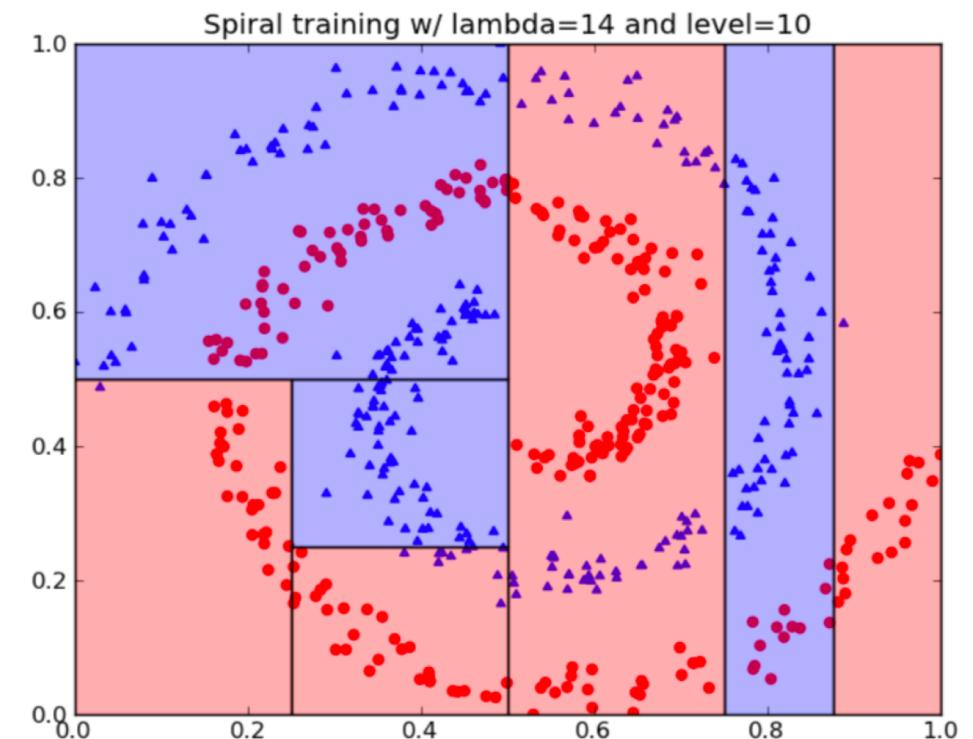
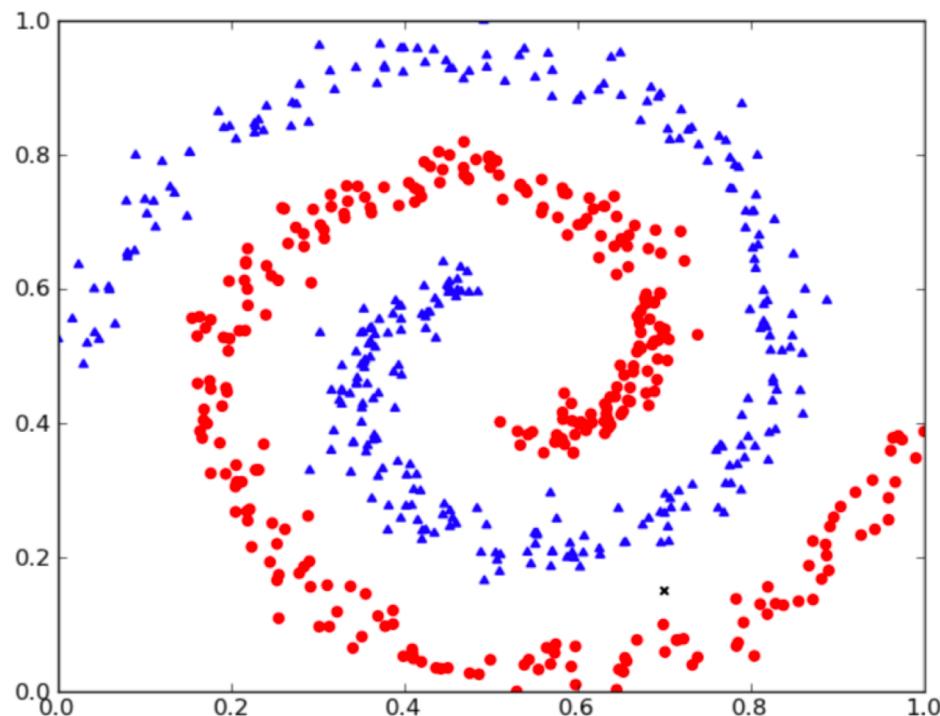
2. Create internal node  $u$  corresponding to the predicate  $[\mathbf{x}_j < t]$
3. If a stopping criterion is satisfied for  $u$ ,  
declare it a leaf, setting some  $c_u \in \mathbb{Y}$  as leaf prediction
4. If not, repeat 1–2 for  $R_1(j, t)$  and  $R_2(j, t)$

› Output: a decision tree  $V$

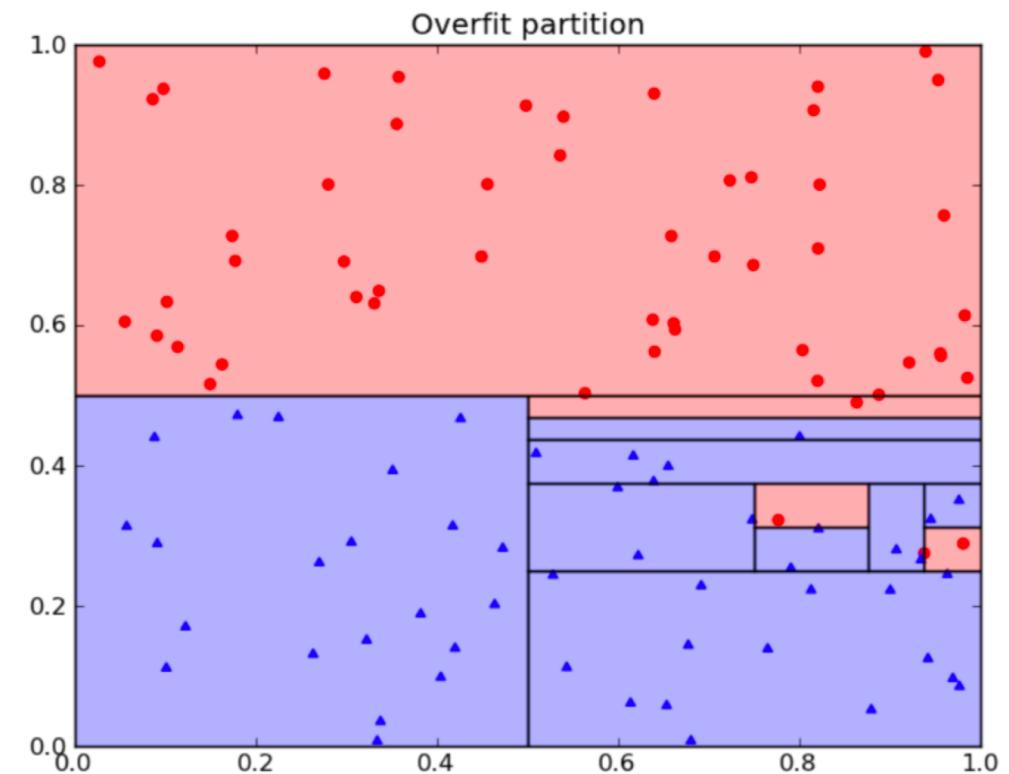
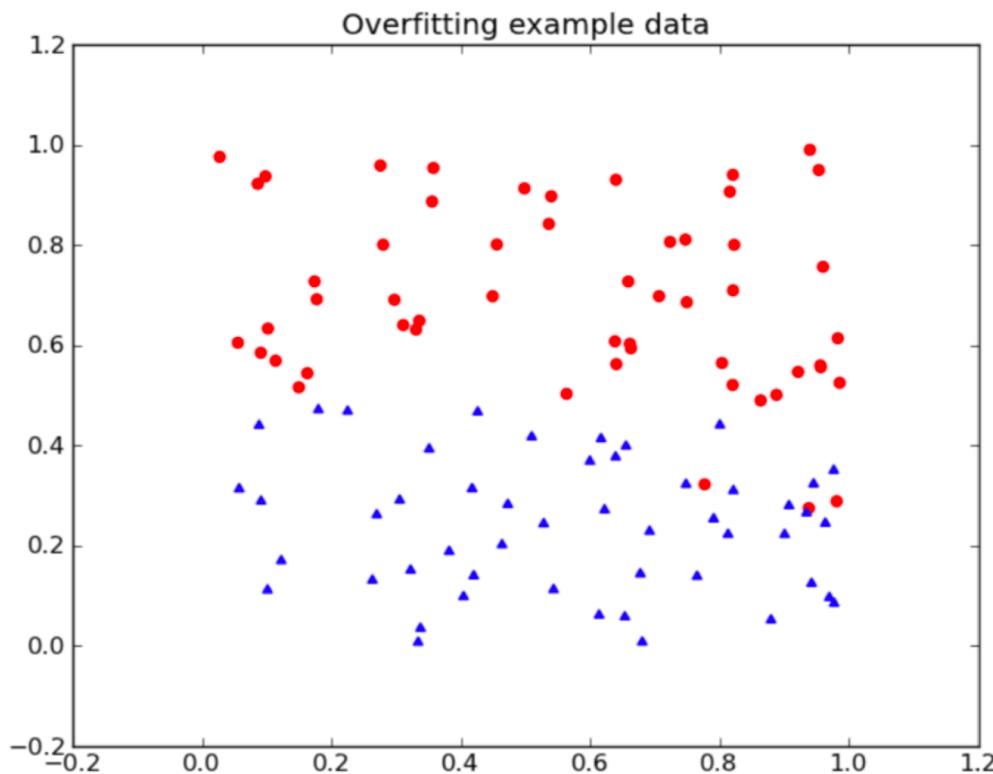
# Greedy tree learning for binary classification



# Greedy tree learning for binary classification



# With decision trees, overfitting is extra-easy!



# Design choices for learning a decision tree classifier

- › Type of predicate in internal nodes
- › The loss function  $Q(X^\ell, j, t)$
- › The stopping criterion
- › Hacks: missing values, pruning, etc.
  
- › CART, C4.5, ID3

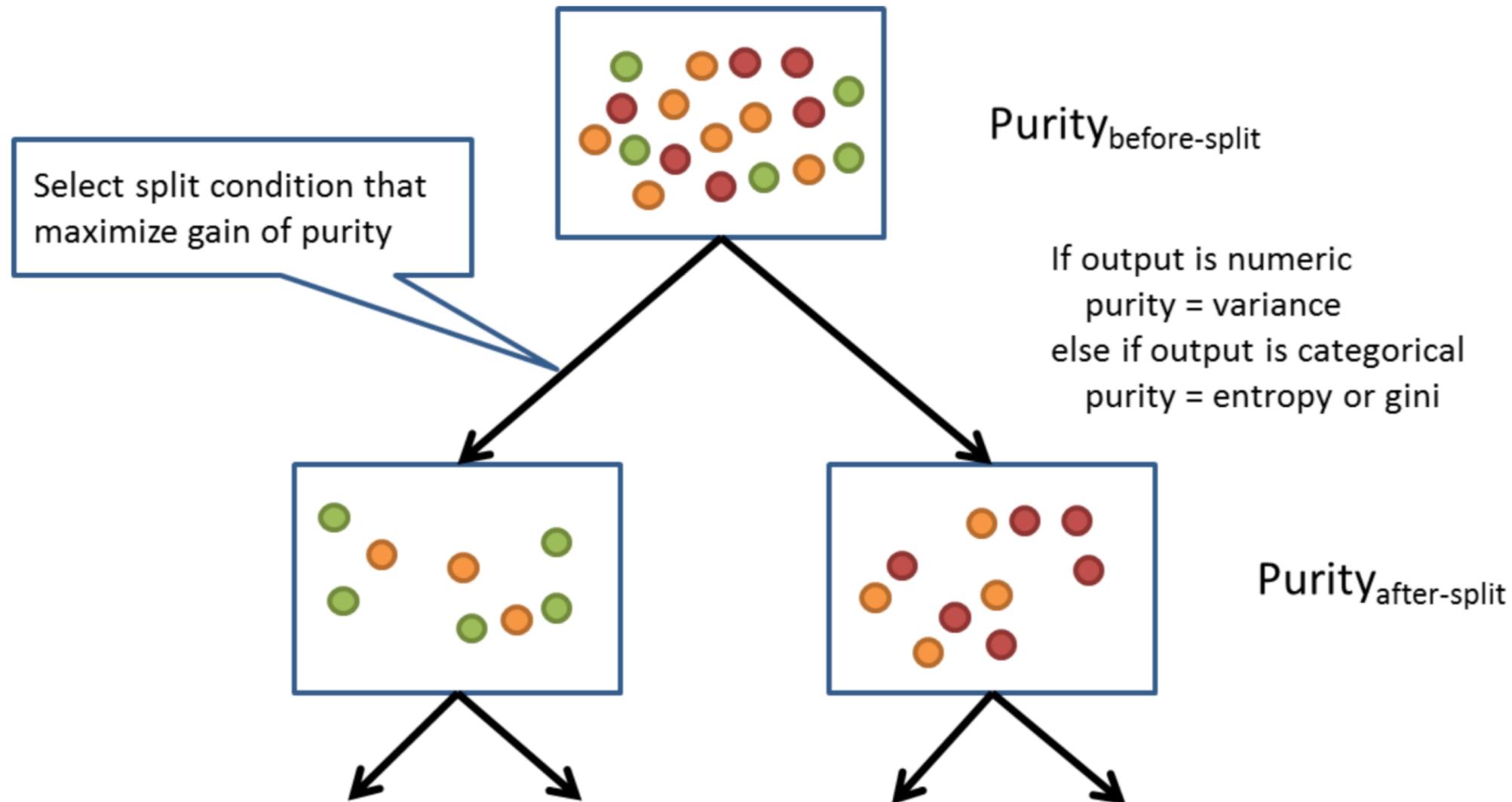
# The loss function $Q(X^\ell, j, t)$

- ›  $R_m$ : the subset of  $X^\ell$  at step  $m$
- › With the current split, let  $R_l \subseteq R_m$  go left and  $R_r \subseteq R_m$  go right
- › Choose predicate to optimize

$$Q(R_m, j, t) = H(R_m) - \frac{|R_l|}{|R_m|}H(R_l) - \frac{|R_r|}{|R_m|}H(R_r) \rightarrow \max$$

- ›  $H(R)$ : impurity criterion
- › Generally  $H(R) = \min_{c \in \mathbb{Y}} \frac{1}{|R|} \sum_{(\mathbf{x}_i, y_i) \in R} L(y_i, c)$

# The idea: maximize purity



# Examples of information criteria

- › Regression:

- ›  $H(R) = \min_{c \in \mathbb{Y}} |R|^{-1} \sum_{(\mathbf{x}_i, y_i) \in R} (y_i - c)^2$

- › Sum of squared residuals minimized by  $c = |R|^{-1} \sum_{(\mathbf{x}_j, y_j) \in R} y_j$
- › Impurity  $\equiv$  variance of the target

- › Classification:

- › Let  $p_k = |R|^{-1} \sum_{(\mathbf{x}_i, y_i) \in R} [y_i = k]$  (share of  $y_i$ 's equal to  $k$ )

- › Miss rate:  $H(R) = \min_{c \in \mathbb{Y}} |R|^{-1} \sum_{(\mathbf{x}_i, y_i) \in R} [y_i \neq c]$

Minimizing miss rate  $1 - p_{k_*}$ ,

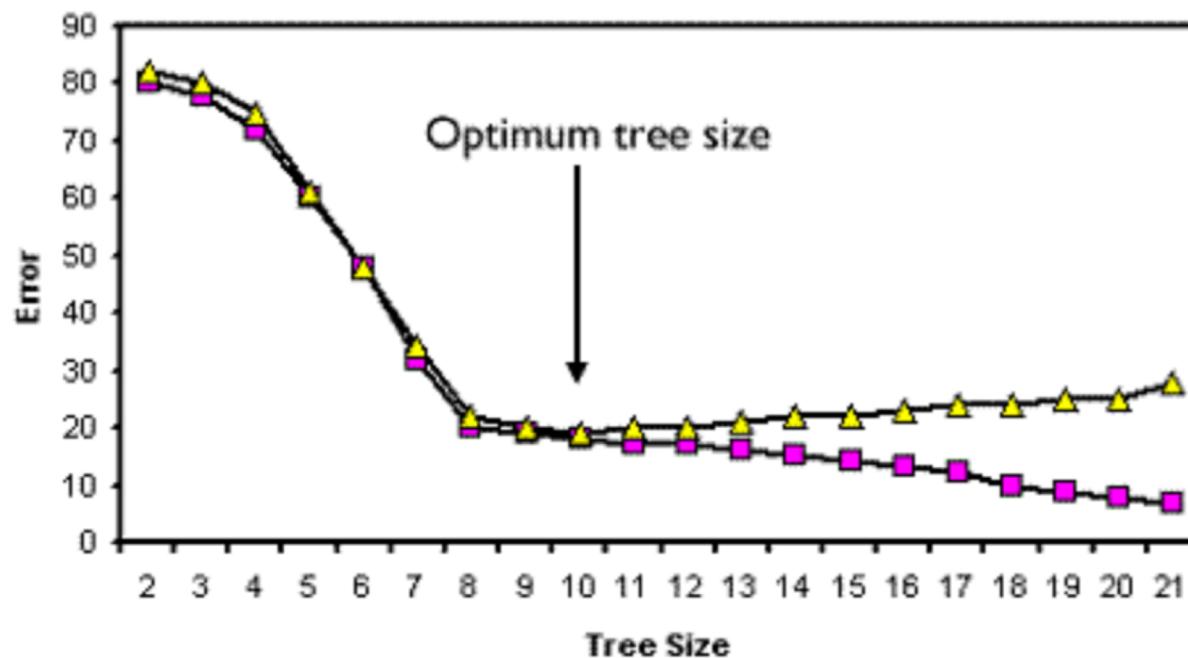
Gini index  $\sum_{k=1}^K p_k(1 - p_k)$ ,

Cross-entropy  $-\sum_{k=1}^K p_k \log p_k$

# Stopping rules for decision tree learning

- › Significantly impacts learning performance
- › Multiple choices available:
  - › Maximum tree depth
  - › Minimum number of objects in leaf
  - › Maximum number of leafs in tree
  - › Stop if all objects fall into same leaf
  - › Constrain quality improvement
    - (stop when improvement gains drop below  $s\%$ )
- › Typically selected via exhaustive search and cross-validation

# Decision tree pruning

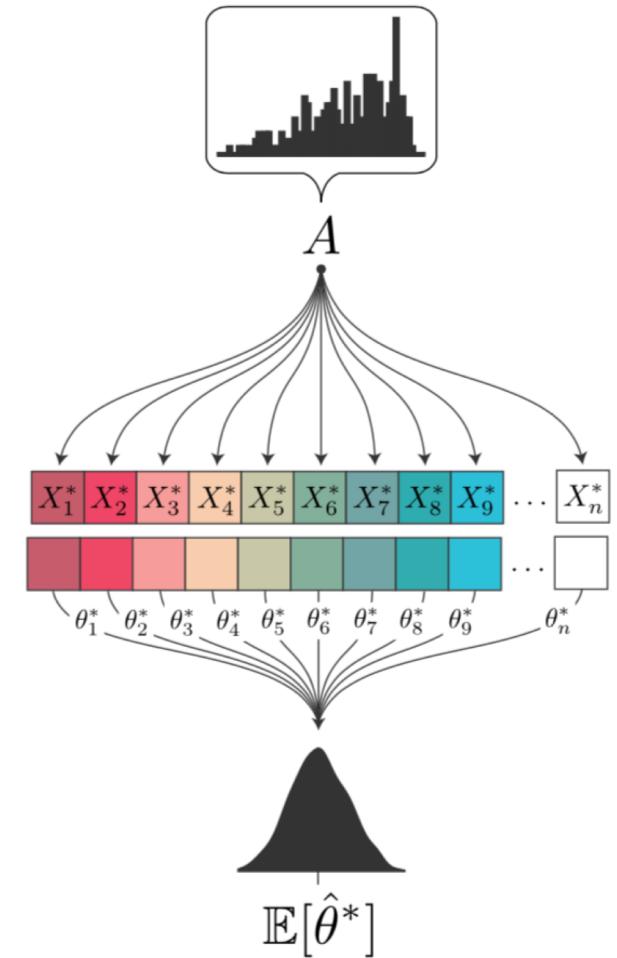


- › Learn a large tree (effectively overfit the training set)
- › Detect overfitting via  $K$ -fold cross-validation
- › Optimize structure by removing least important nodes

# Bagging and Random Forest

# The bootstrapping procedure

- › Input: a sample  $X^\ell = \{(x_i, y_i)\}_{i=1}^\ell$
- › **Bootstrapping:** generate new samples  $X_1^m$  of  $(x_i, y_i)$  drawn from  $X^\ell$  uniformly at random with replacement (replicated  $(x_i, y_i)$  possible!)
- › **Ensemble learning idea:**
  1. Generate  $N$  bootstrapped samples  $X_1^m, \dots, X_N^m$
  2. Learn  $N$  hypotheses  $h_1, \dots, h_N$
  3. Average predictions to obtain 
$$h(x) = \frac{1}{N} \sum_{i=1}^N h_i(x)$$
  4. Profit!



# Bagging: bootstrap aggregation (+ demo)

- › Input: a sample

$$X^\ell = \{(x_i, y_i)\}_{i=1}^\ell$$

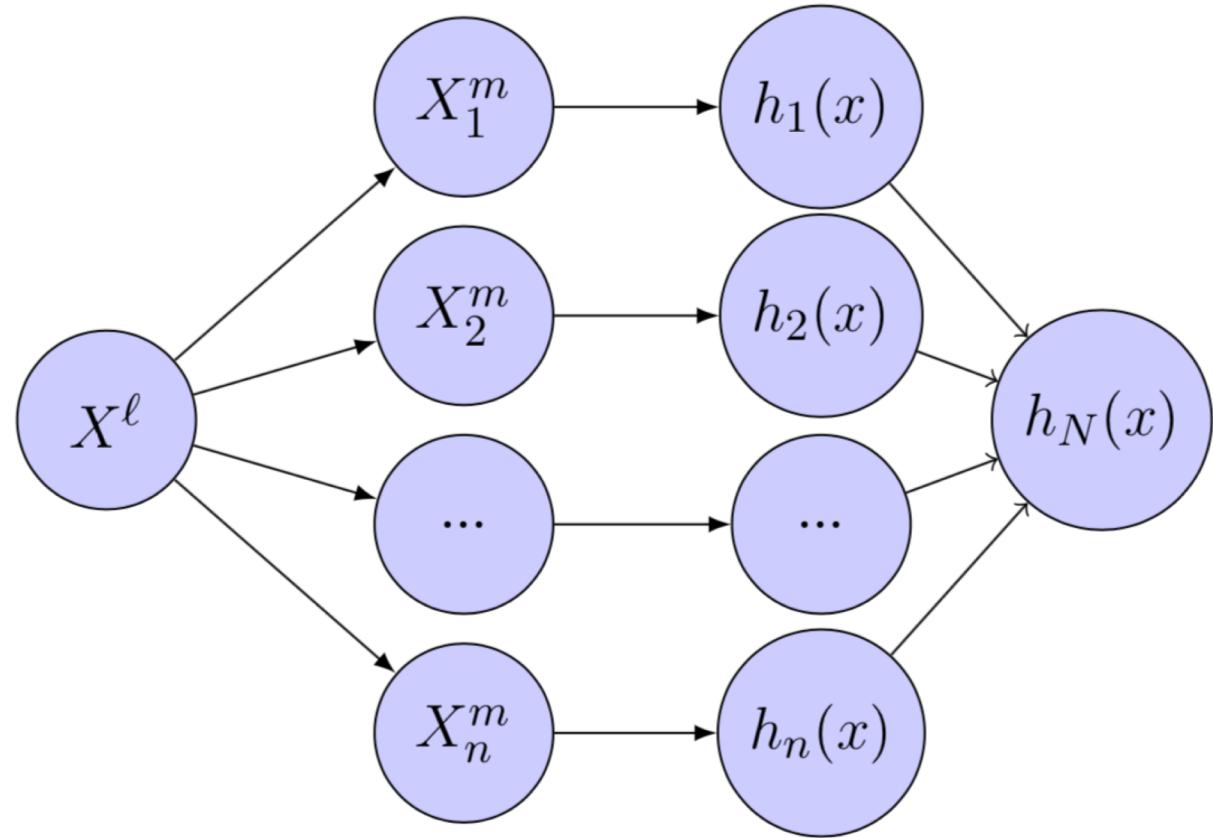
- › Weak learners via bootstrapping

$$\tilde{\mu}(X^\ell) = \mu(\tilde{X}^\ell)$$

- › Ensemble average

$$h_N(x) = \frac{1}{N} \sum_{i=1}^N h_i(x) =$$

$$= \frac{1}{N} \sum_{i=1}^N \tilde{\mu}(X^\ell)(x)$$



# The Random Forest algorithm

- › Bagging over (weak) decision trees
- › Reduce error via averaging over instances and features
- › Input: a sample  $X^\ell = \{(\mathbf{x}_i, y_i)\}_{i=1}^\ell$ , where  $\mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{Y}$
- › The algorithm iterates for  $i = 1, \dots, N$ :
  1. Pick  $p$  random features out of  $d$
  2. Bootstrap a sample  $X_i^m = \{(\mathbf{x}_i, y_i)\}_{i=1}^\ell$  where  $\mathbf{x}_i \in \mathbb{R}^p, y_i \in \mathbb{Y}$
  3. Learn a decision tree  $h_i(\mathbf{x})$  using bootstrapped  $X_i^m$
  4. Stop when leafs in  $h_i$  contain less than  $n_{\min}$  instances

$$\mathbf{x}_i \in \{\text{A, B, C}\}$$

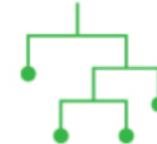
$$X^\ell = \{(\mathbf{x}_i, y_i)\}_{i=1}^5$$

Bootstrap  $X_i^m, i \in \{1, 2, 3, 4\}$

Learn Tree $_i(\mathbf{x})$  using  $X_i^m$

Tree 1

[1, 1, 2, 4, 5]  
[A, B]



Tree 2

[2, 1, 3, 4, 5]  
[A, C]



Tree 3

[2, 1, 3, 4, 5]  
[B, C]

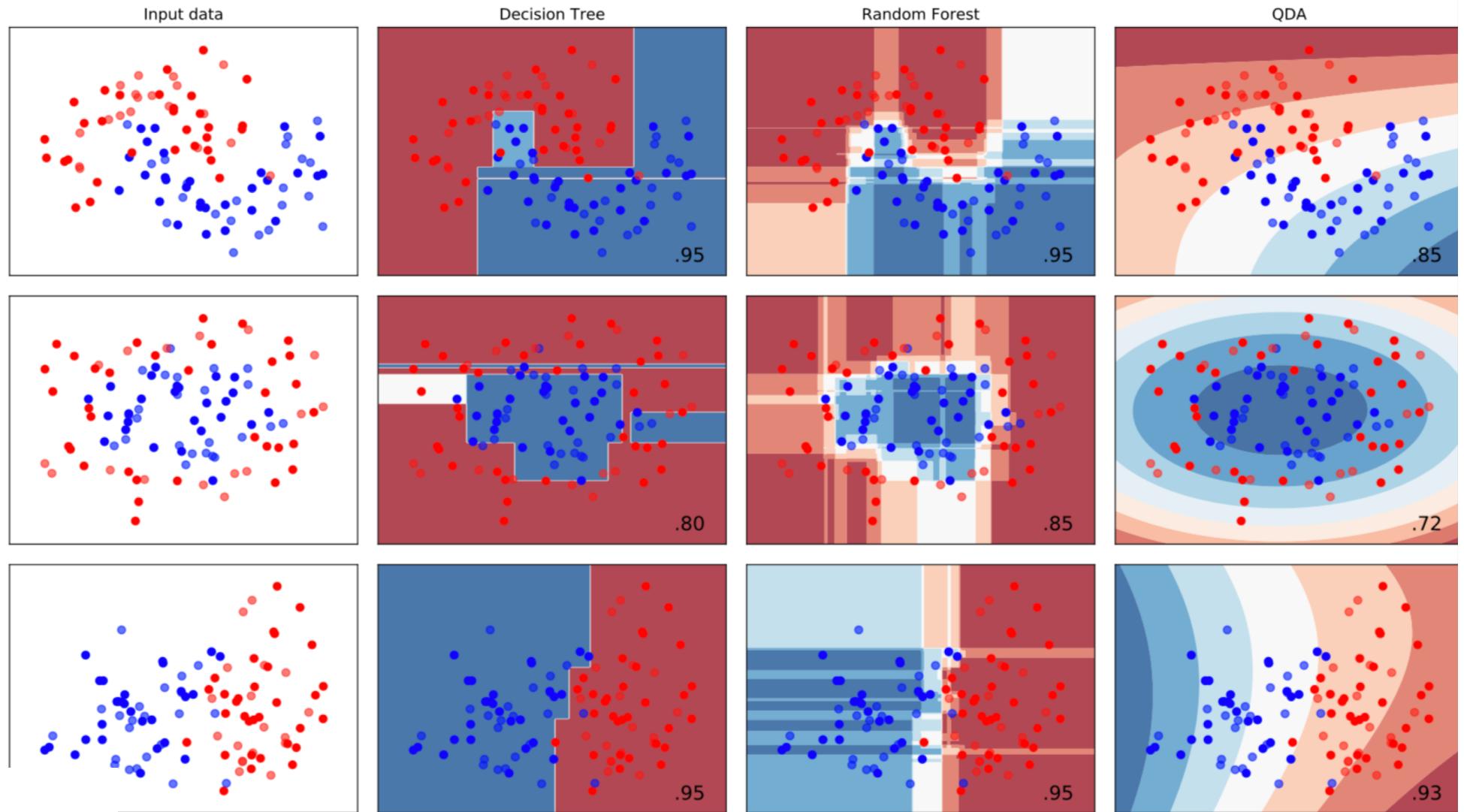


Tree  $N$

[1, 1, 3, 3, 4]  
[A, C]



# Random Forest: synthetic examples



# A bit of Learning Theory

# Bias-variance decomposition

- › Input: the training set  $X^\ell = \{(x_i, y_i)\}_{i=1}^\ell$
- › Learning method  $\mu : (\mathbb{X} \times \mathbb{Y})^\ell \rightarrow \mathbb{H}$
- › Can evaluate quality using average (over possible samples  $X^\ell$ )

$$\begin{aligned} Q(\mu) &= \mathbb{E}_{X^\ell} \left[ \mathbb{E}_{x,y} \left[ (y - \mu(X^\ell)(x))^2 \right] \right] = \\ &= \int_{(\mathbb{X} \times \mathbb{Y})^\ell} \int_{\mathbb{X} \times \mathbb{Y}} (y - \mu(X^\ell)(x))^2 p(x, y) \prod_{i=1}^{\ell} p(x_i, y_i) dx dy dx_i dy_i \end{aligned}$$

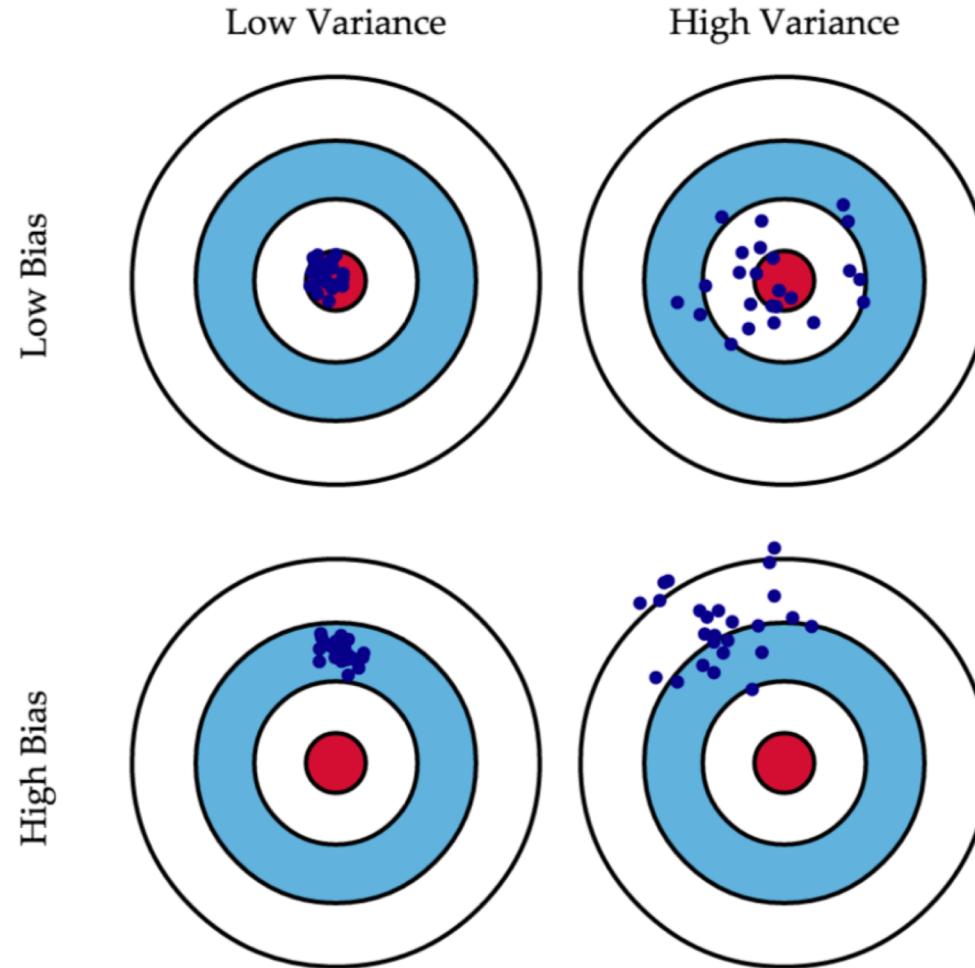
# Bias-variance decomposition

- › We arrive at the famous bias-variance(-noise) decomposition

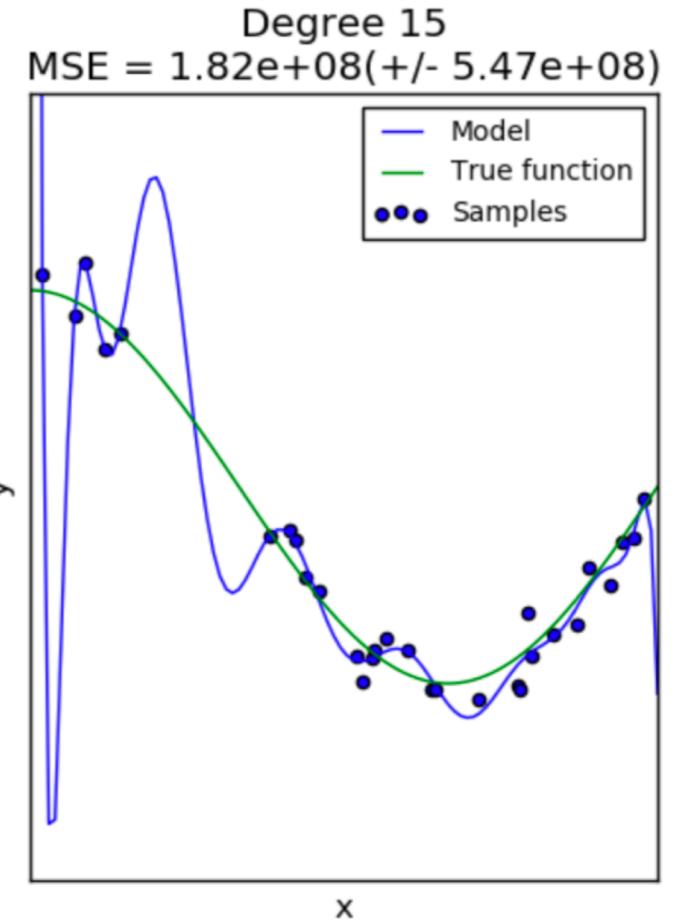
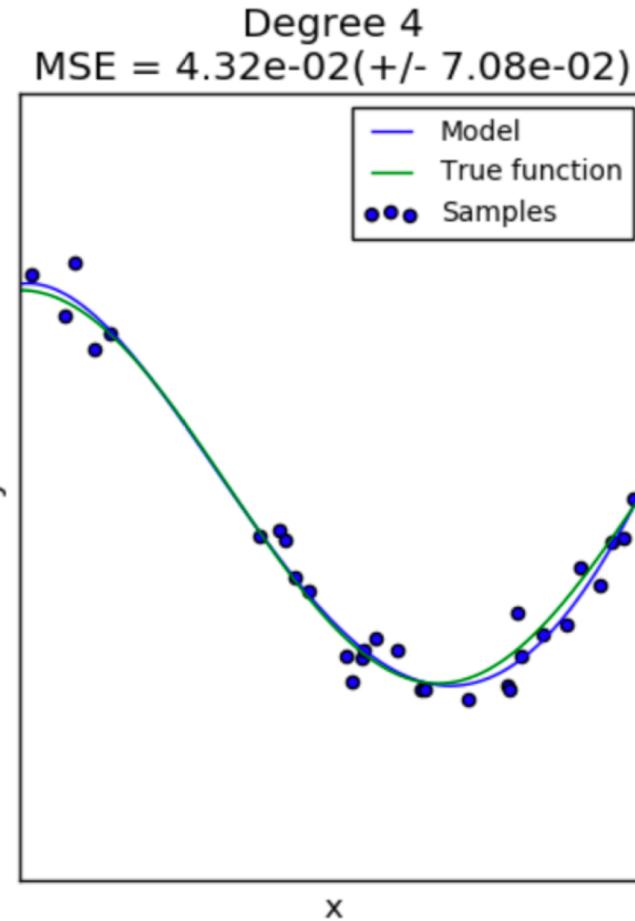
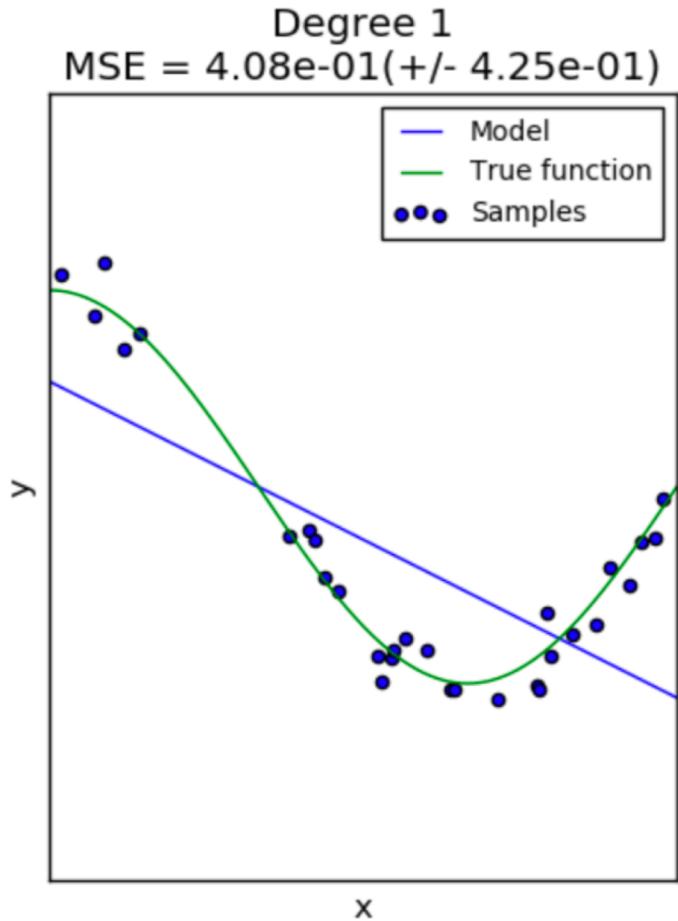
$$Q(\mu) = \underbrace{\mathbb{E}_{x,y} \left[ (y - \mathbb{E}[y | x])^2 \right]}_{\text{noise}} + \underbrace{\mathbb{E}_x \left[ (\mathbb{E}_{X^\ell} [\mu(X^\ell)] - \mathbb{E}[y | x])^2 \right]}_{\text{bias}} + \underbrace{\mathbb{E}_x \left[ \mathbb{E}_{X^\ell} \left[ (\mu(X^\ell) - \mathbb{E}_{X^\ell} [\mu(X^\ell)])^2 \right] \right]}_{\text{variance}}$$

- › **Noise term:** an error of an ideal learner (nobody can do better!)
- › **Bias term:** learner's approximation of the ideal algorithm
  - › The more complex the learning algorithm, the lower the bias
- › **Variance term:** sensitivity to sample replacement
  - › Simple algorithms have lower variance

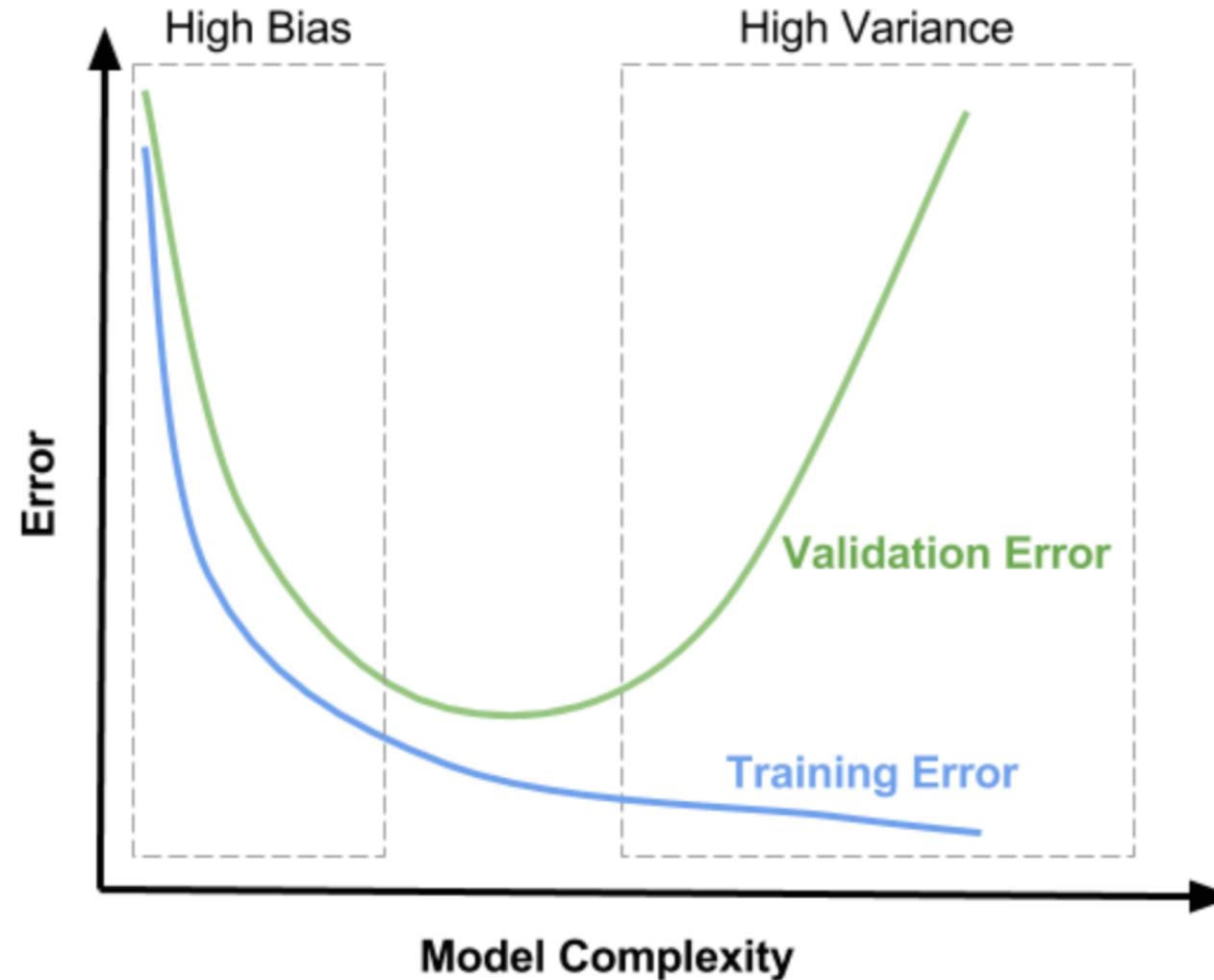
# Bias-variance decomposition



# Polynomial fits of different degrees



# Bias-variance tradeoff



# An application: statistical analysis of Bagging

- › Bias: not made any worse by bagging multiple hypotheses

$$\begin{aligned}\mathbb{E}_{x,y} \left[ \left( \mathbb{E}_{X^\ell} \left[ \frac{1}{N} \sum_{n=1}^N \tilde{\mu}(X^\ell)(x) \right] - \mathbb{E}[y | x] \right)^2 \right] &= \\ \mathbb{E}_{x,y} \left[ \left( \frac{1}{N} \sum_{n=1}^N \mathbb{E}_X^\ell [\tilde{\mu}(X^\ell)(x)] - \mathbb{E}[y | x] \right)^2 \right] &= \\ \mathbb{E}_{x,y} \left[ \left( \mathbb{E}_{X^\ell} [\tilde{\mu}(X^\ell)(x)] - \mathbb{E}[y | x] \right)^2 \right]\end{aligned}$$

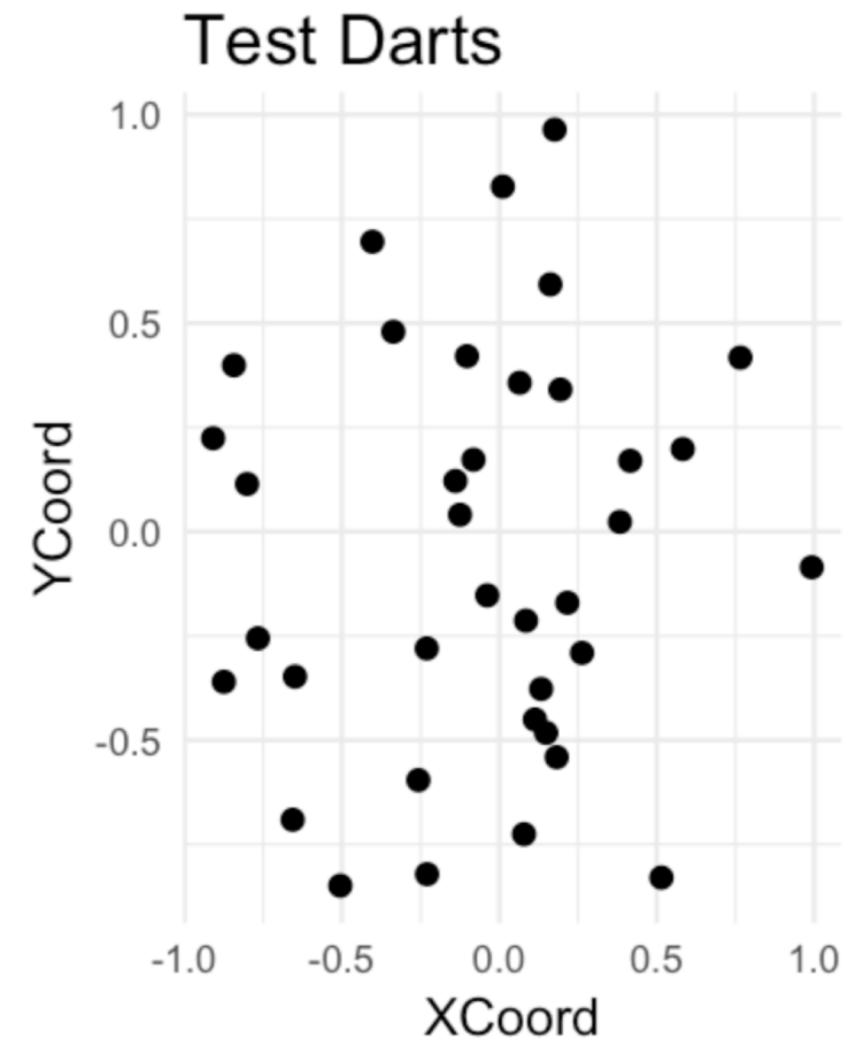
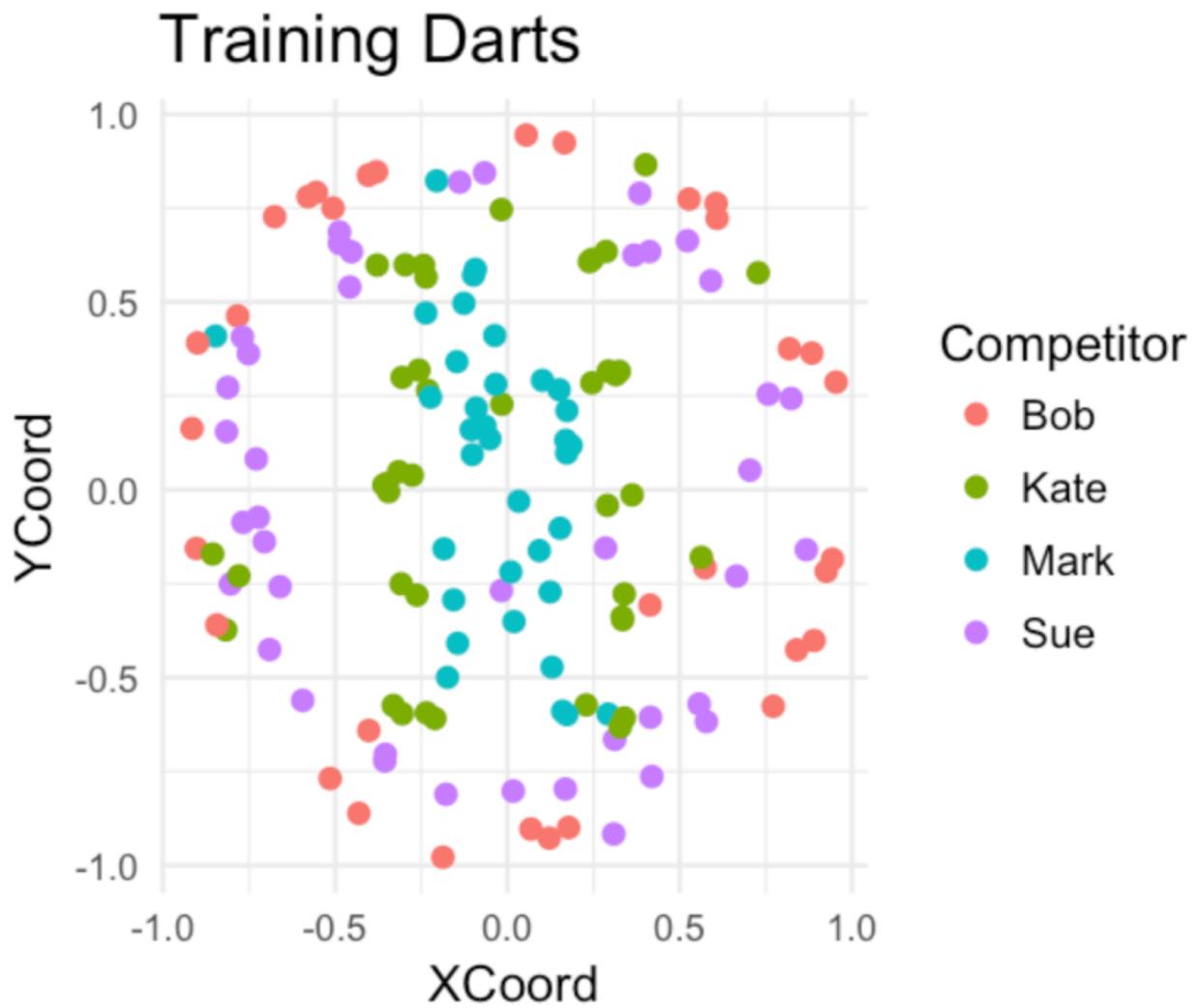
# An application: statistical analysis of Bagging

- › Variance:  $N$  times lower for uncorrelated hypotheses,  
yet not as much an improvement for highly correlated

$$\begin{aligned} \mathbb{E}_{x,y} \left[ \mathbb{E}_{X^\ell} \left[ \left( \frac{1}{N} \sum_{n=1}^N \tilde{\mu}(X^\ell)(x) - \mathbb{E}_{X^\ell} \left[ \frac{1}{N} \sum_{n=1}^N \tilde{\mu}(X^\ell)(x) \right] \right)^2 \right] \right] &= \\ &= \frac{1}{N} \mathbb{E}_{x,y} \left[ \mathbb{E}_{X^\ell} \left[ \left( \tilde{\mu}(X^\ell)(x) - \mathbb{E}_{X^\ell} [\tilde{\mu}(X^\ell)(x)] \right)^2 \right] \right] + \\ &\quad + \frac{N(N-1)}{N^2} \mathbb{E}_{x,y} \left[ \mathbb{E}_{X^\ell} \left[ \left( \tilde{\mu}(X^\ell)(x) - \mathbb{E}_{X^\ell} [\tilde{\mu}(X^\ell)(x)] \right) \times \right. \right. \\ &\quad \left. \left. \times \left( \tilde{\mu}(X^\ell)(x) - \mathbb{E}_{X^\ell} [\tilde{\mu}(X^\ell)(x)] \right) \right] \right] \end{aligned}$$

# Stacking

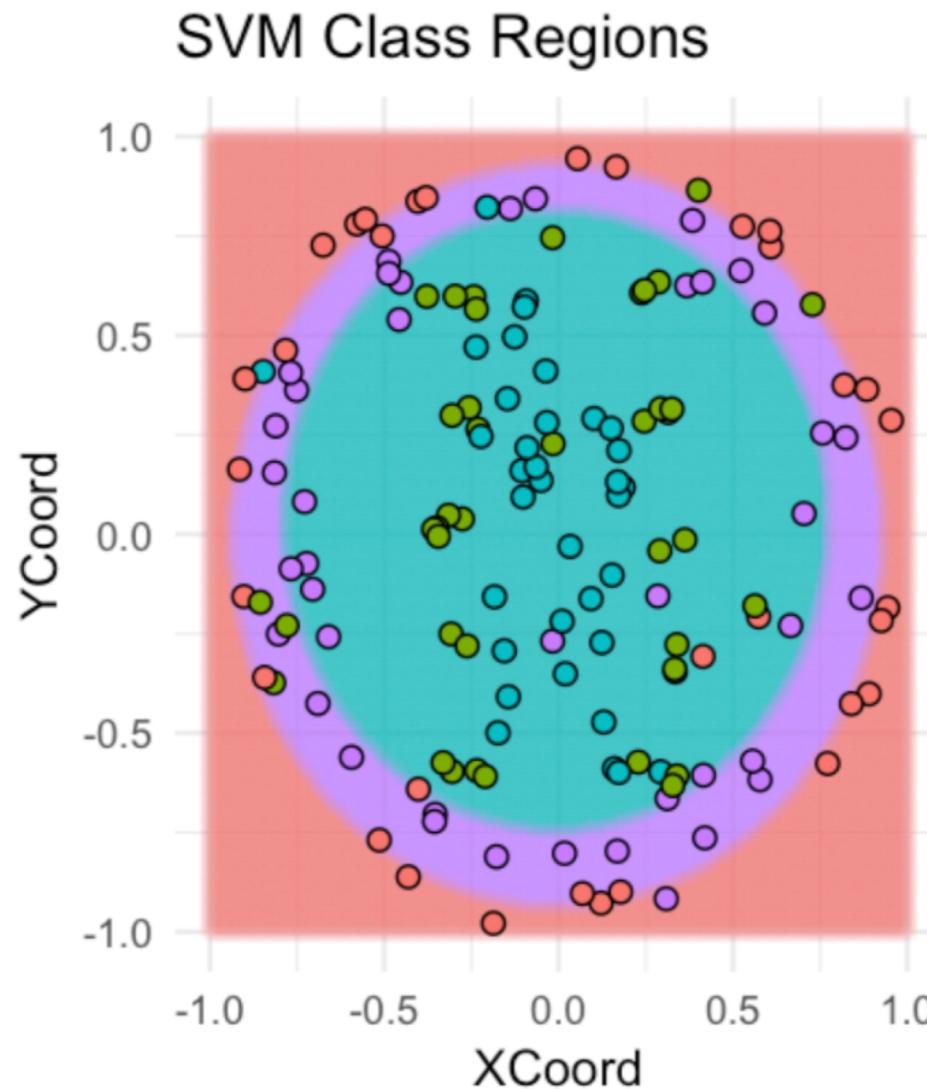
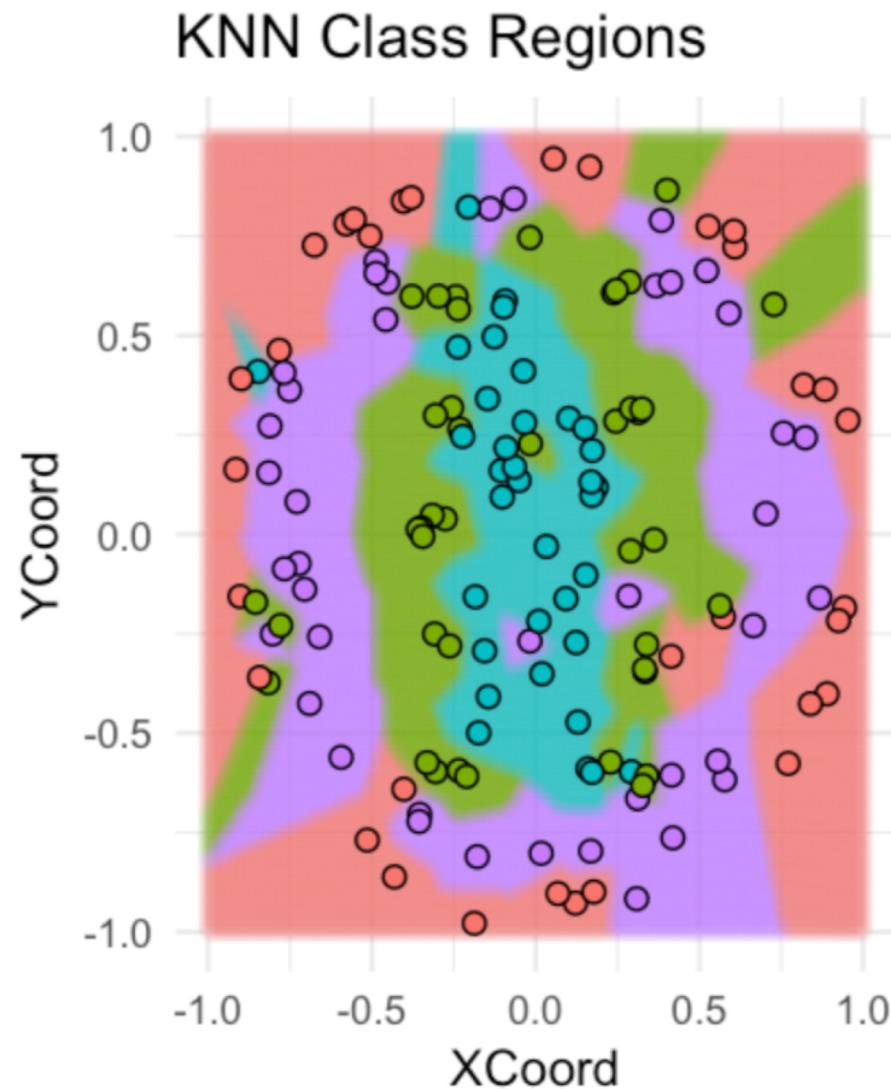
# Stacking motivation: the game of Darts



# Base model training

- › Select  $k$  nearest neighbours as base model 1
  - › Fit base model 1 in the most fancy way possible  
(grid search for optimal  $k$  using  $K$ -fold cross-validation, etc.)
  - ›  $k$ -NN accuracy on Test Darts: 70%
- 
- › Select Support Vector Machine as base model 2
  - › Fit base model 2 in the most fancy way possible  
(different penalizations, grid search for optimal kernel width using  $K$ -fold cross-validation, etc.)
  - › SVM accuracy on Test Darts: 78%

# Results for base models



Competitor

- Bob
- Kate
- Mark
- Sue

# Stacking base models

1. Partition `train` into 5 folds
2. Create `train_meta/test_meta`: same row/fold IDs as in `train/test`, empty M1/M2
3. For each  $\text{Fold}_i \in \{\text{Fold}_1, \dots, \text{Fold}_5\}$ 
  - 3.1 Combine the other 4 folds for training  $\rightarrow \text{Fold}_{-i}$
  - 3.2 Fit each base model to  $\text{Fold}_{-i}$ , predict on  $\text{Fold}_i$ , save predictions to M1/M2 in `train_meta`
4. Fit each base model to `train`, predict on `test`, save predictions to M1/M2 in `test_meta`
5. Fit stacking model S (LinearRegression) to `train_meta`, using M1/M2 as features
6. Use the stacked model S to make final predictions on `test_meta`

# Results for base models



# Gradient Boosting

# Boosting for regression

- › Consider a regression problem  $\frac{1}{2} \sum_{i=1}^{\ell} (h(x_i) - y_i)^2 \rightarrow \min_h$
- › Search for solution in the form of weak learner composition  $a_N(x) = \sum_{n=1}^N h_n(x)$  with weak learners  $h_n \in \mathbb{H}$
- › The **boosting approach**: add weak learners greedily
  1. Start with a “trivial” weak learner  $h_0(x) = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i$
  2. At step  $N$ , compute the residuals

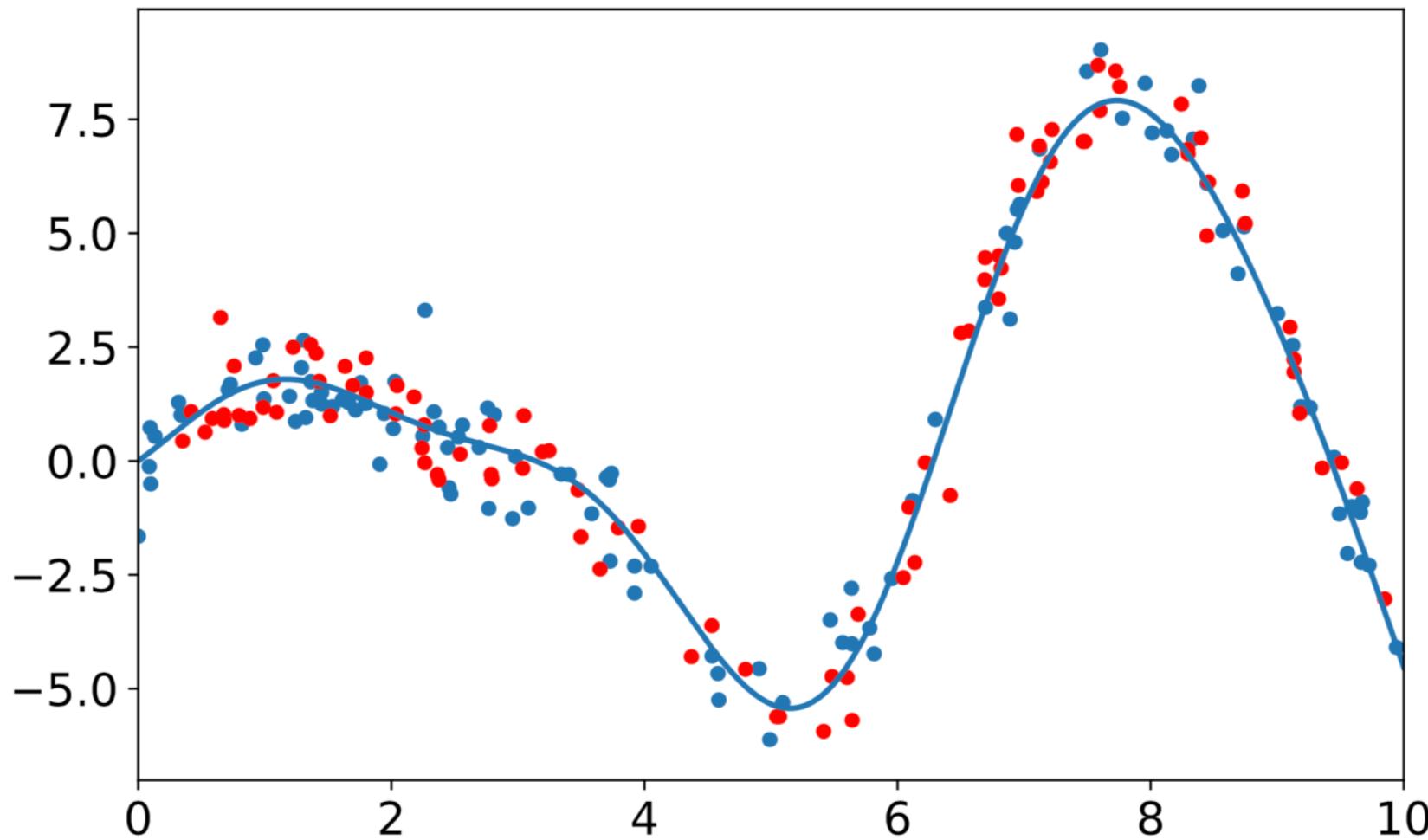
$$s_i^{(N)} = y_i - \sum_{n=1}^{N-1} h_n(x_i) = y_i - a_{N-1}(x_i), \quad i = 1, \dots, \ell$$

3. Learn the next weak algorithm using

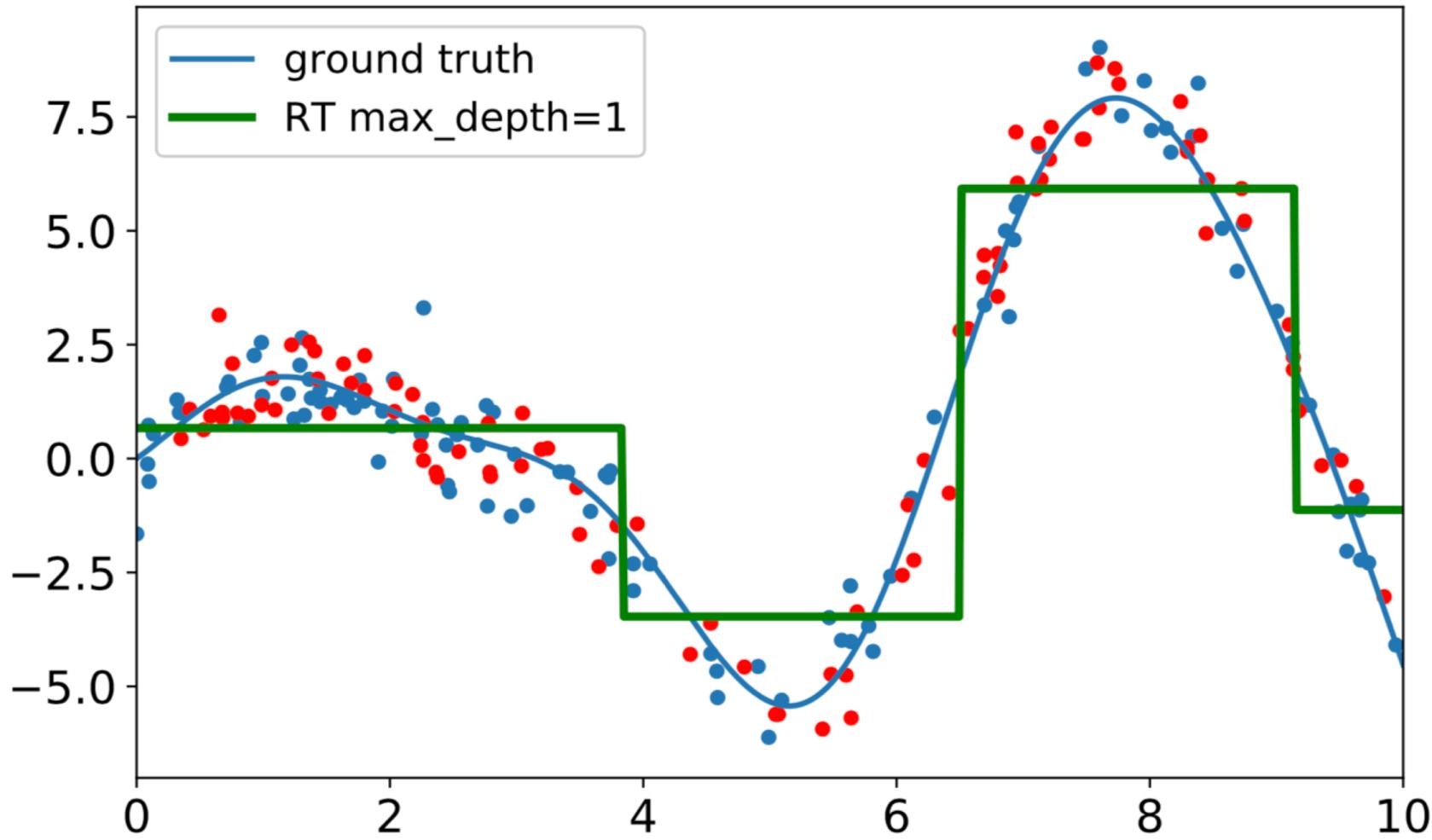
$$a_N(x) := \arg \min_{h \in \mathbb{H}} \frac{1}{2} \sum_{i=1}^{\ell} (h(x_i) - s_i^{(N)})^2$$

(this implementation may be found in, e.g., **scikit-learn**)

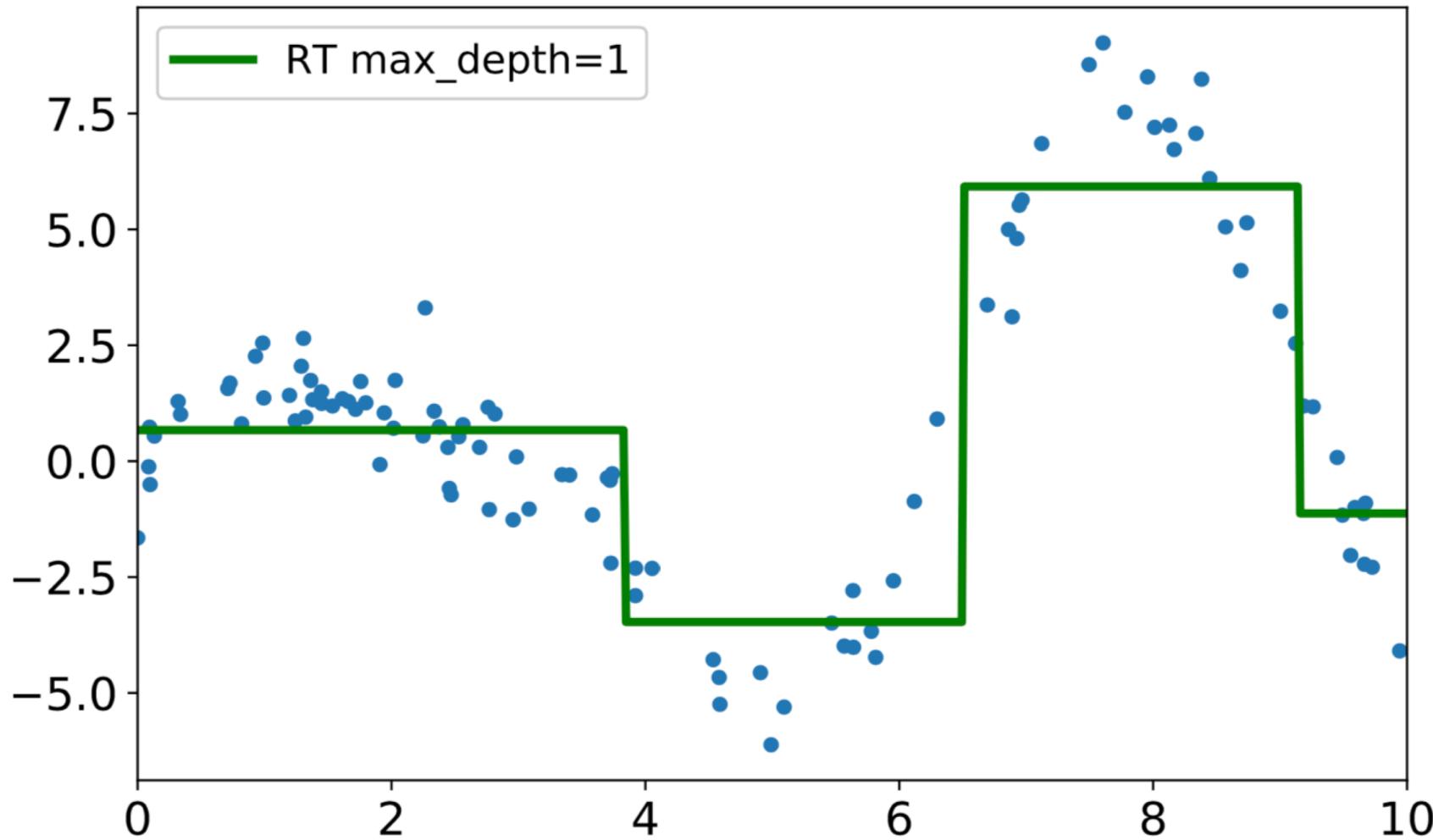
# Boosting: an example regression problem



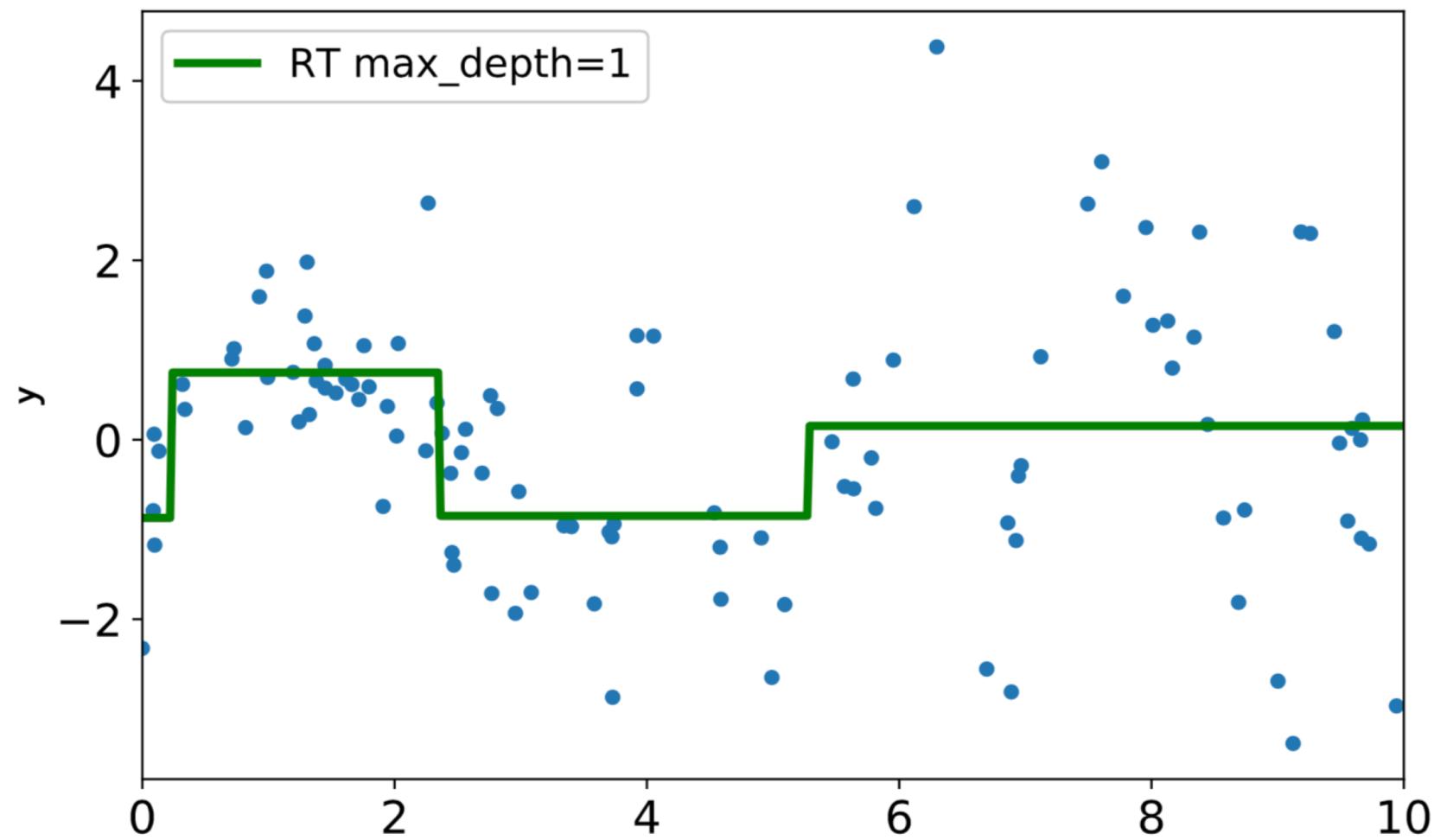
# Boosting: an example regression problem



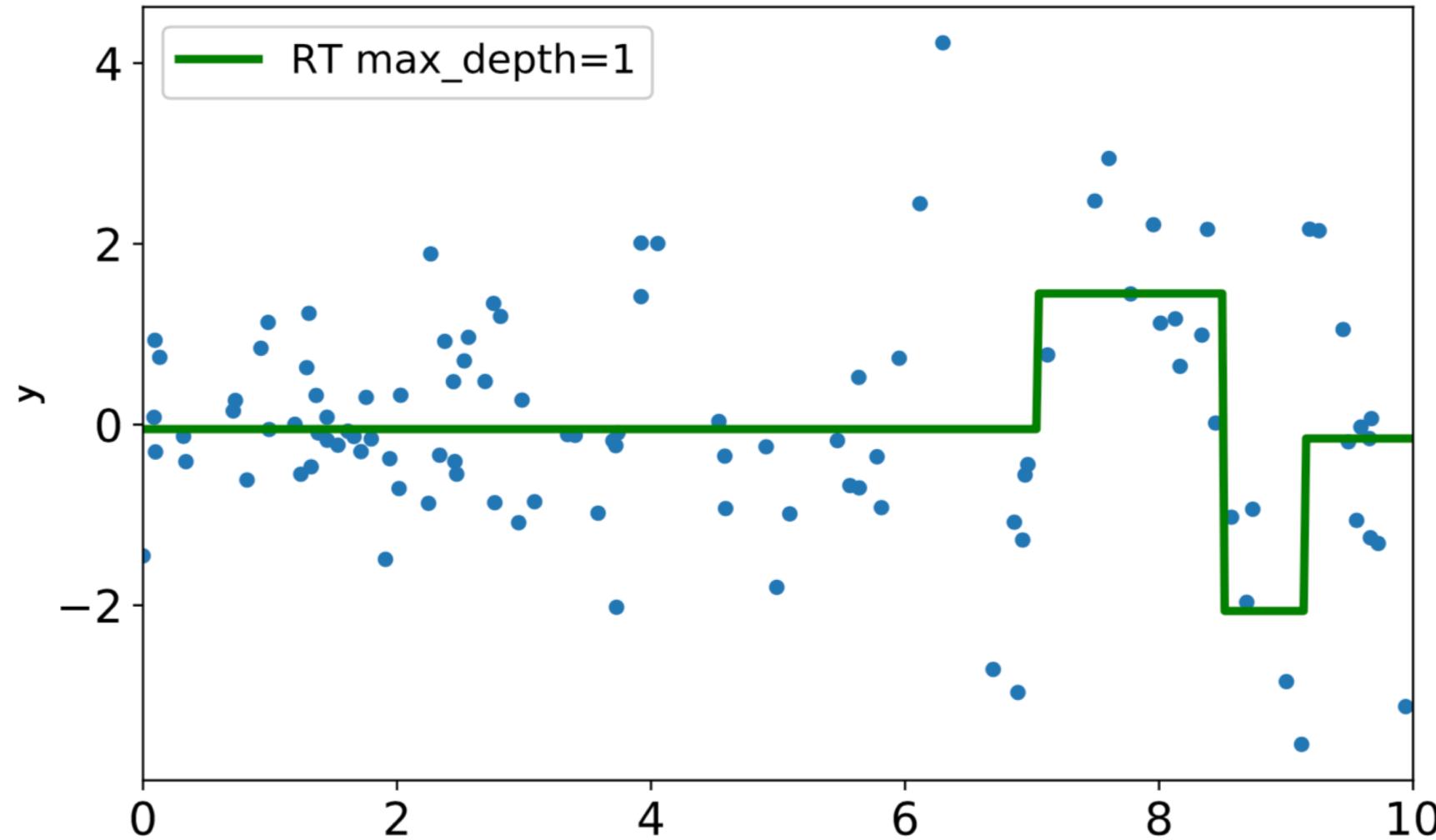
# Boosting: an example regression problem



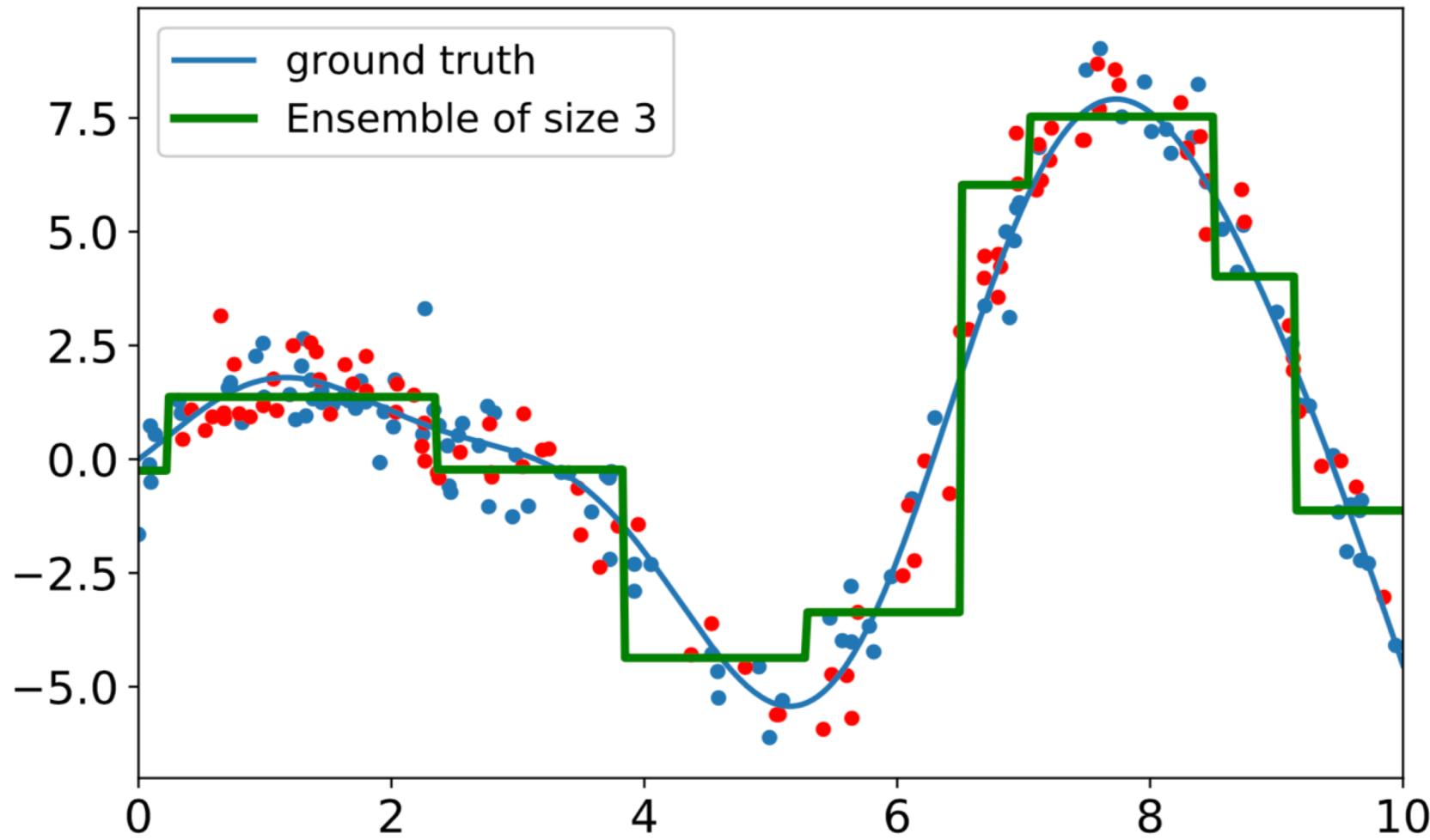
# Boosting: an example regression problem



# Boosting: an example regression problem



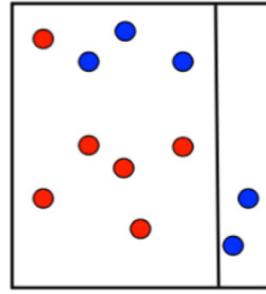
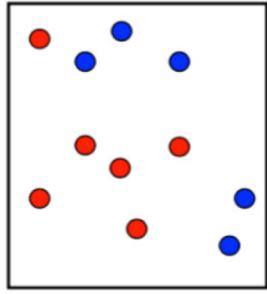
# Boosting: an example regression problem



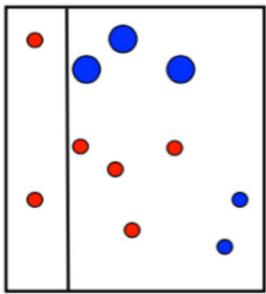
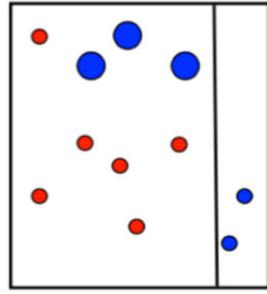
# Adaptive boosting for classification

- › Training set  $X^\ell = \{(\mathbf{x}_i, y_i)\}_{i=1}^\ell, y_i \in \{-1, +1\}$
- › Search for solution in the form of a **weighted** sum  $a_N(\mathbf{x}) = \sum_{n=1}^N \gamma_n h_n(\mathbf{x})$  with weak learners  $h_n \in \mathbb{H}$
- › At step  $N$ , extend  $a$  with  $h_N$ :  $a_N(\mathbf{x}_i) = a_{N-1}(\mathbf{x}_i) + \gamma_N h_N(\mathbf{x}_i)$ .  
How do we choose  $h_N$  and its weight  $\gamma_N$ ?
- › Measure fit quality using **exponential loss**  $Q(a_N, X^\ell) = \sum_{i=1}^\ell \exp\{-y_i a_N(\mathbf{x}_i)\}$
- › Derivation yields  $h_N = \arg \min_h \sum_{i=1}^\ell (h(\mathbf{x}_i) - w_i y_i)^2$   
with **weights**  $w_i = \exp\{-y_i a_{N-1}(\mathbf{x}_i)\}$
- › Weak learner weight: minimize  $\gamma_N = \arg \min_\gamma Q(a_{N-1}(\mathbf{x}_i) + \gamma h_N(\mathbf{x}_i), X^\ell)$

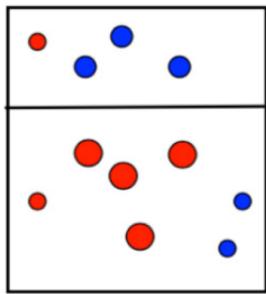
# Adaptive boosting for classification



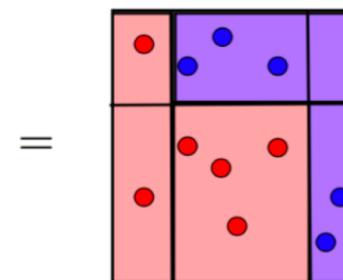
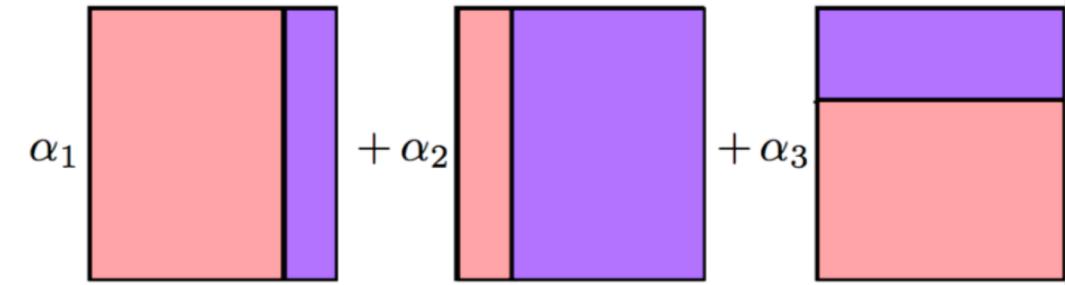
$t = 1$



$t = 2$



$t = 3$



# Gradient boosting: motivation

- › With  $a_{N-1}(\mathbf{x})$  already built, how to find the next  $\gamma_N$  and  $h_N$  if

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(\mathbf{x}_i) + \gamma h(\mathbf{x}_i)) \rightarrow \min_{\gamma, h}$$

- › Recall: functions decrease in the direction of negative gradient
- › View  $L(y, z)$  as a function of  $z$  ( $= a_N(\mathbf{x}_i)$ ), execute gradient descent on  $z$
- › Search for such  $s_1, \dots, s_\ell$  that

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(\mathbf{x}_i) + s_i) \rightarrow \min_{s_1, \dots, s_\ell}$$

- › Choose  $s_i = - \left. \frac{\partial L(y_i, z)}{\partial z} \right|_{z=a_{N-1}(\mathbf{x}_i)}$ , approximate  $s_i$ 's by  $h_N(\mathbf{x}_i)$

# The Gradient Boosting Machine [Friedman, 2001]

- › Input:
  - › Training set  $X^\ell = \{(\mathbf{x}_i, y_i)\}_{i=1}^\ell$
  - › Number of boosting iterations  $N$
  - › Loss function  $Q(y, z)$  with its gradient  $\frac{\partial Q}{\partial z}$
  - › A family  $\mathbb{H} = \{h(\mathbf{x})\}$  of weak learners and their associated learning procedures
  - › Additional hyperparameters of weak learners (tree depth, etc.)
- › Initialize GBM  $h_0(\mathbf{x})$  using some simple rule (zero, most popular class, etc.)
- › Execute boosting iterations  $t = 1, \dots, N$  (see next slide)
- › Compose the final GBM learner:  $a_N(\mathbf{x}) = \sum_{t=0}^N \gamma_t h_t(\mathbf{x})$

# The Gradient Boosting Machine [Friedman, 2001]

At every iteration:

1. Compute **pseudo-residuals**:  $s_i = - \left. \frac{\partial Q(y_i, z)}{\partial z} \right|_{z=a_{N-1}(\mathbf{x}_i)}, i = 1, \dots, \ell$
2. Learn  $h_N(\mathbf{x}_i)$  by regressing onto  $s_1, \dots, s_\ell$ :

$$h_N(x) = \arg \min_{h \in \mathbb{H}} \sum_{i=1}^{\ell} (h(\mathbf{x}_i) - s_i)^2$$

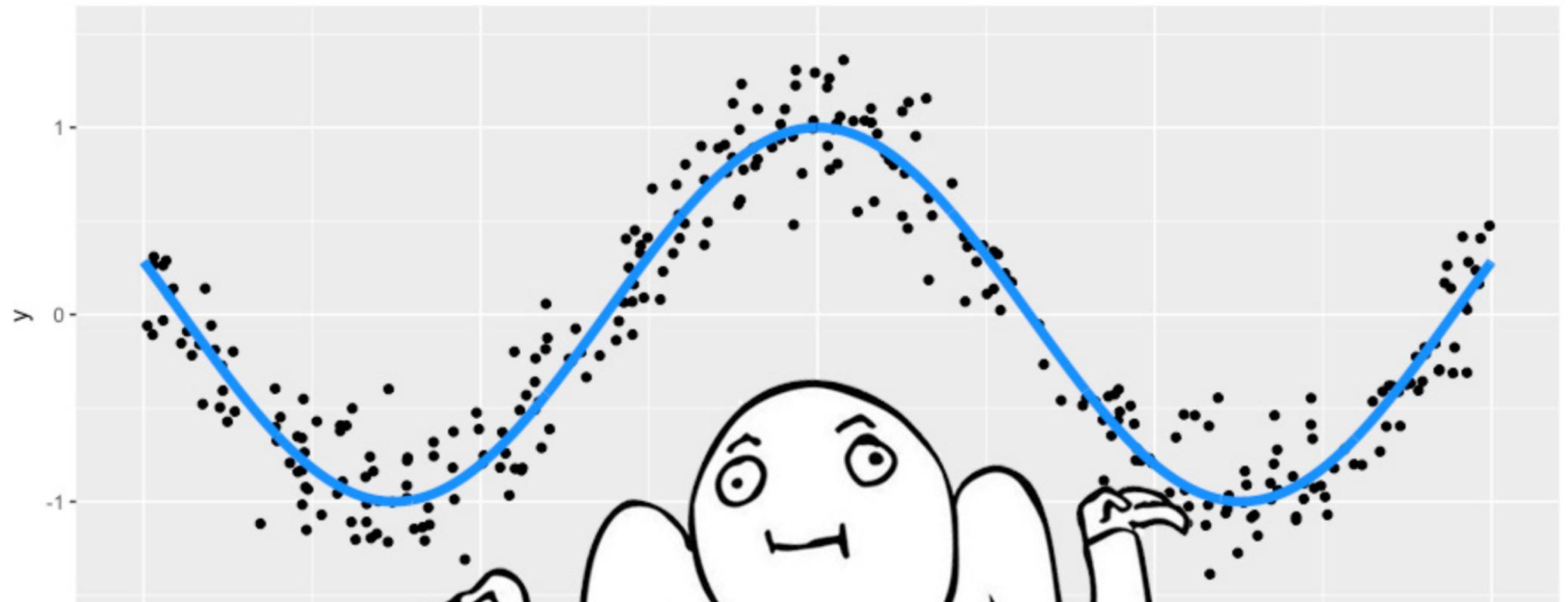
3. Find the optimal  $\gamma_N$  using plain gradient descent:

$$\gamma_N = \arg \min_{\gamma \in \mathbb{R}} \sum_{i=1}^{\ell} Q(y_i, a_{N-1}(\mathbf{x}_i) + \gamma h_N(\mathbf{x}_i))$$

4. Update the GBM by  $a_N(\mathbf{x}_i) \leftarrow a_{N-1}(\mathbf{x}) + \gamma_N h_N(\mathbf{x})$

# GBM: an example regression problem

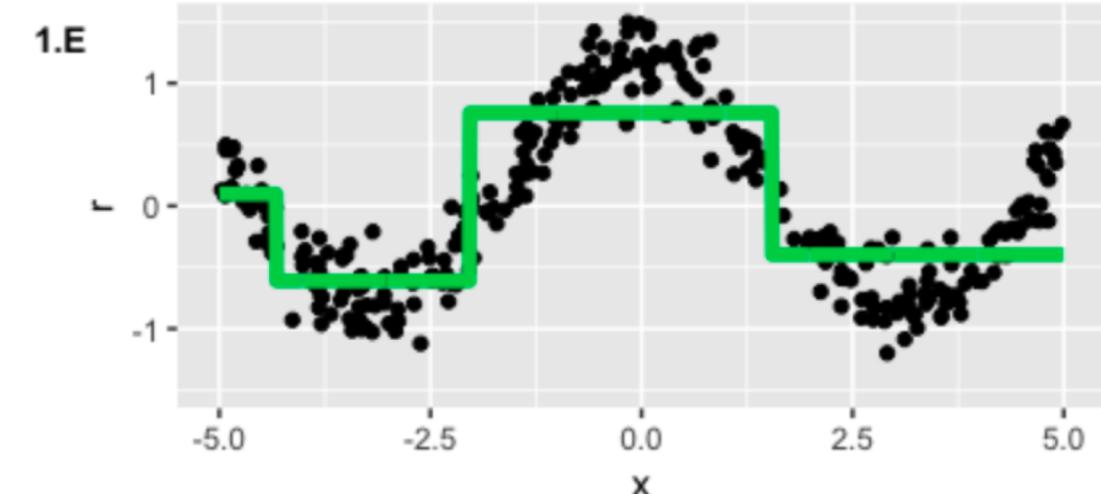
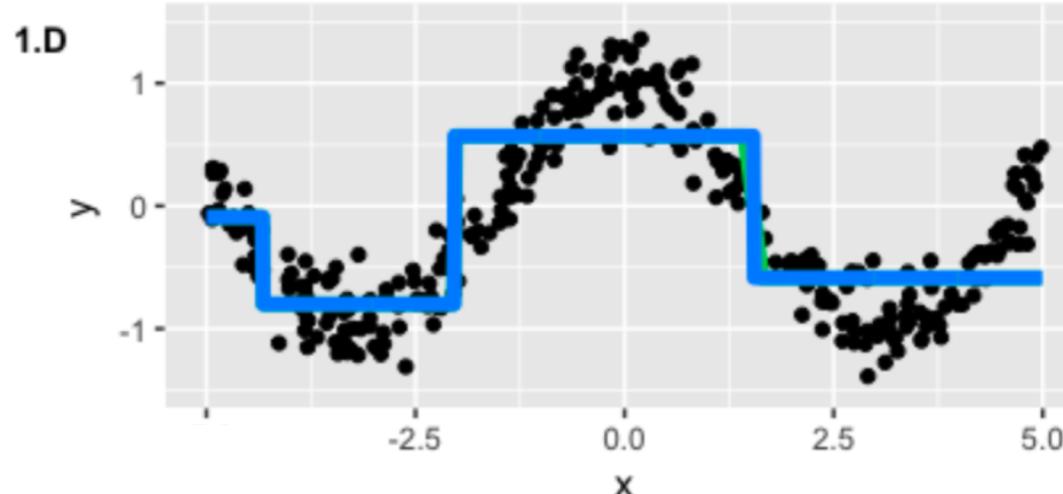
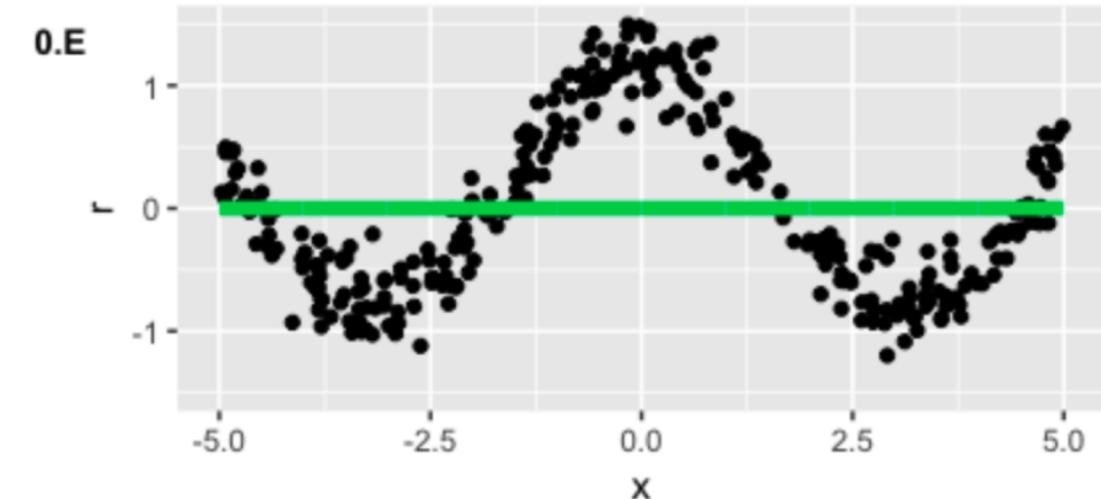
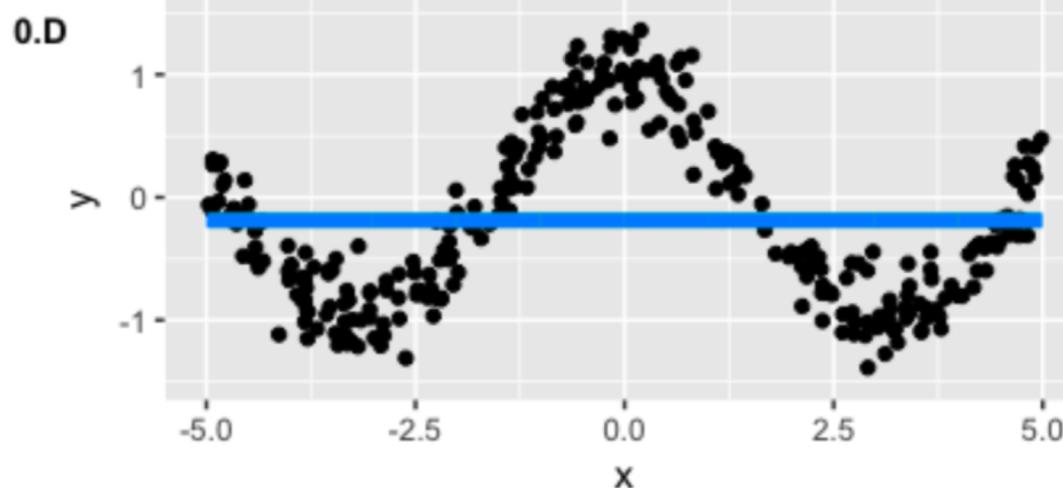
- › Consider a training set for a  $X^{300} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{300}$  where  $x_i \in [-5, 5]$ ,  $y_i = \cos(x_i) + \varepsilon_i$ ,  $\varepsilon_i \sim \mathcal{N}(0, 1/5)$



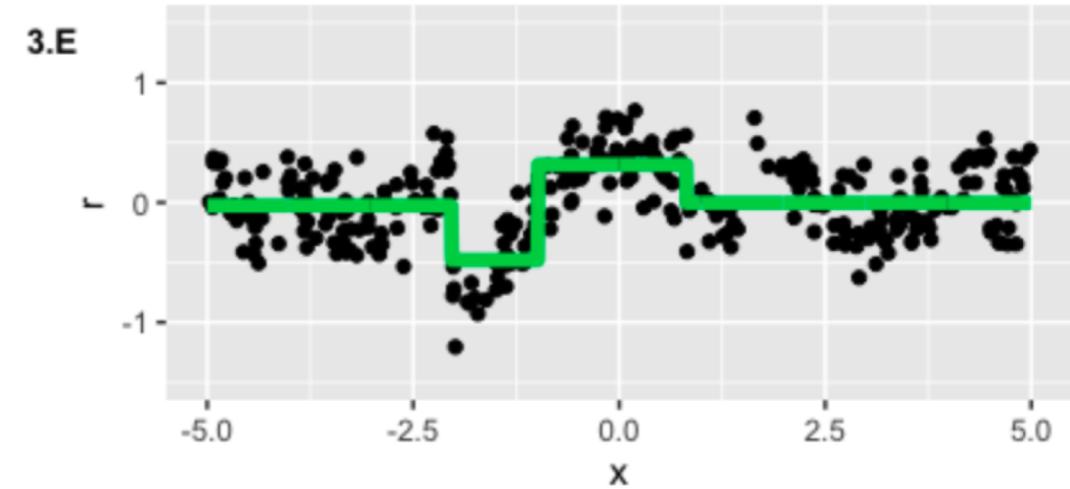
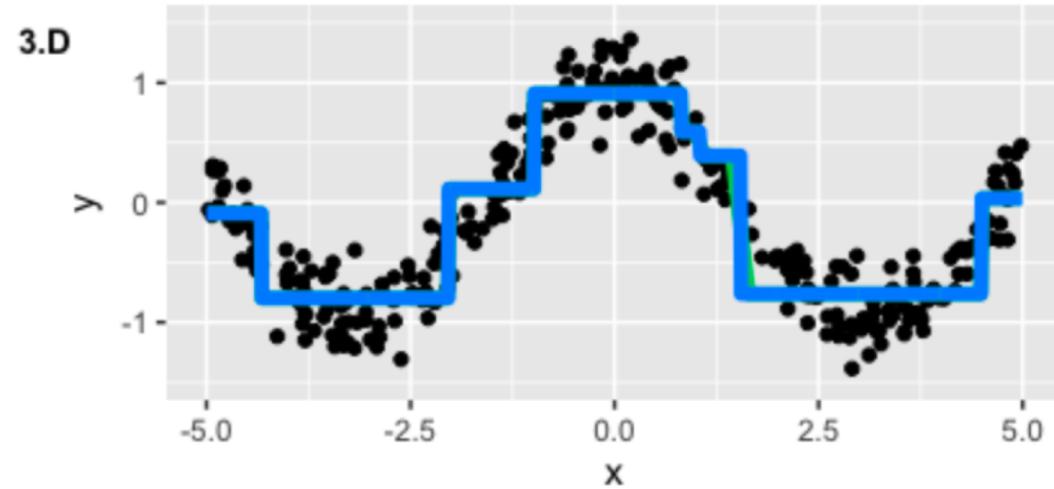
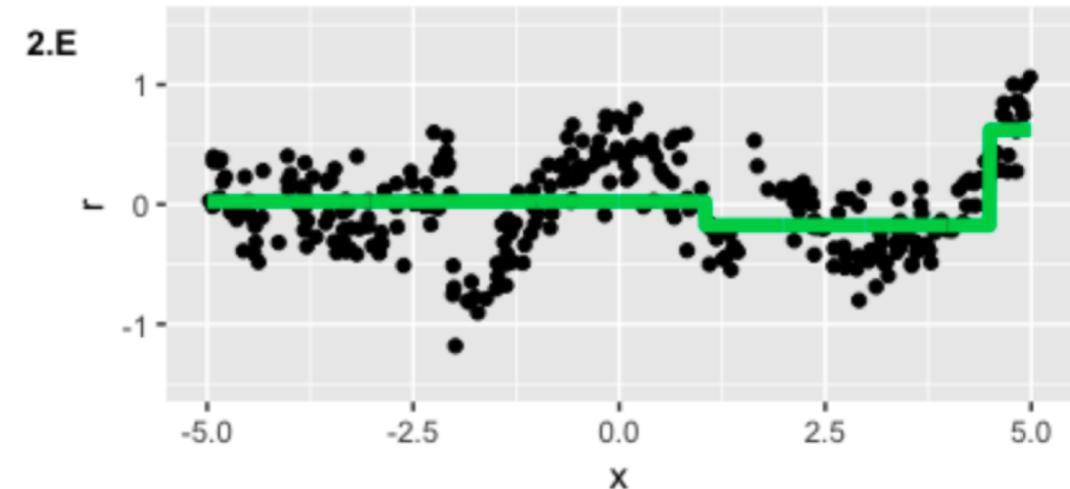
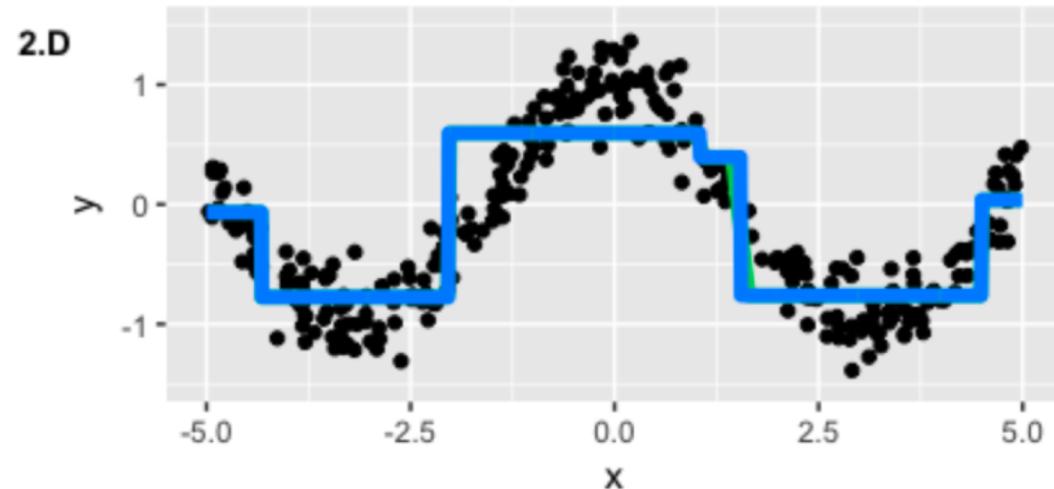
# GBM: an example regression problem

- › Pick  $N = 3$  boosting iterations
- › Quadratic loss  $Q(y, z) = (y - z)^2$
- › Gradient of the quadratic loss  $\frac{\partial Q(y_i, z)}{\partial z} = (y - z)$  is just residuals
- › Pick decision trees as weak learners  $h_i(\mathbf{x})$
- › Set 2 as the maximum depth for decision trees

# GBM: an example regression problem

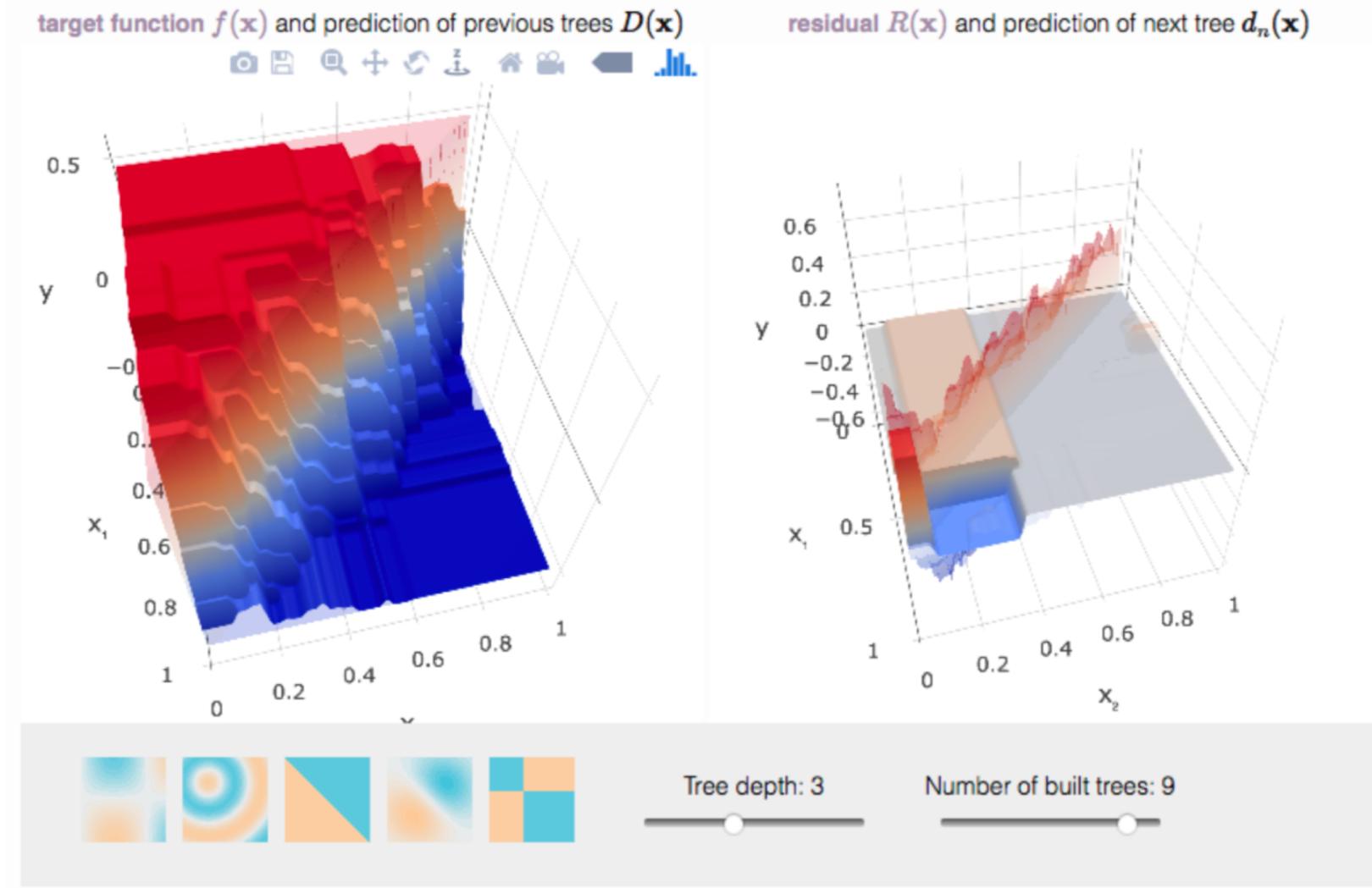


# GBM: an example regression problem



# GBM: an interactive demo

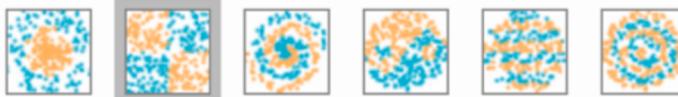
[http://arogozhnikov.github.io/2016/06/24/gradient\\_boosting\\_explained.html](http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html)



# GBM: an interactive demo

[http://arogozhnikov.github.io/2016/07/05/gradient\\_boosting\\_playground.html](http://arogozhnikov.github.io/2016/07/05/gradient_boosting_playground.html)

Dataset to classify:



Prediction:

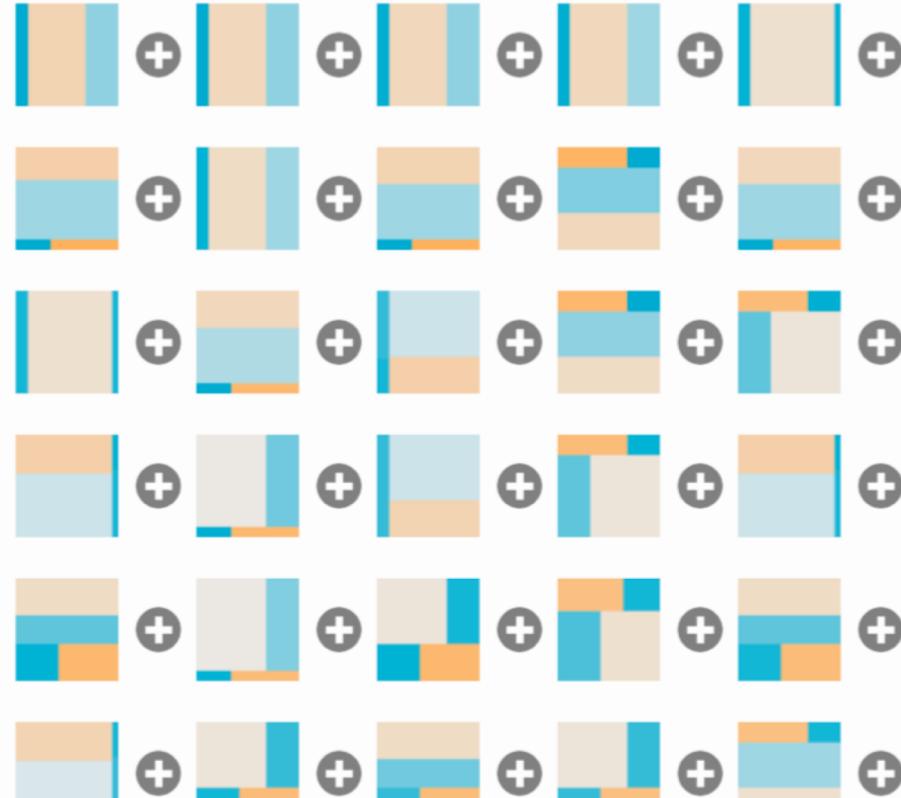


↑  
predictions of GB (all 50 trees)

train loss: 0.266

test loss: 0.312

Decision functions of first 30 trees



# GBM: regularization via shrinkage

- › For **too simple weak learners**, the negative gradient is approximated badly  $\implies$  random walk in space of samples
- › For **too complex weak learners**, a few boosting steps may be enough for overfitting
- › Shrinkage: make shorter steps using a learning rate  $\eta \in (0, 1]$

$$a_N(\mathbf{x}_i) \leftarrow a_{N-1}(\mathbf{x}) + \eta \gamma_N h_N(\mathbf{x})$$

(effectively distrust gradient direction estimated via weak learners)

# GBM: shrinkage animated

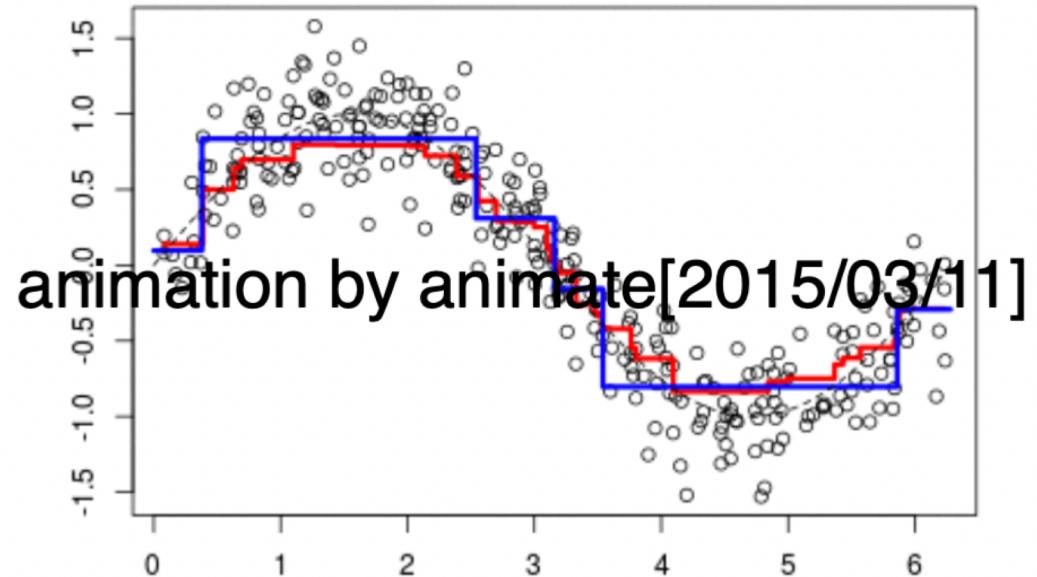


Figure: High shrinkage

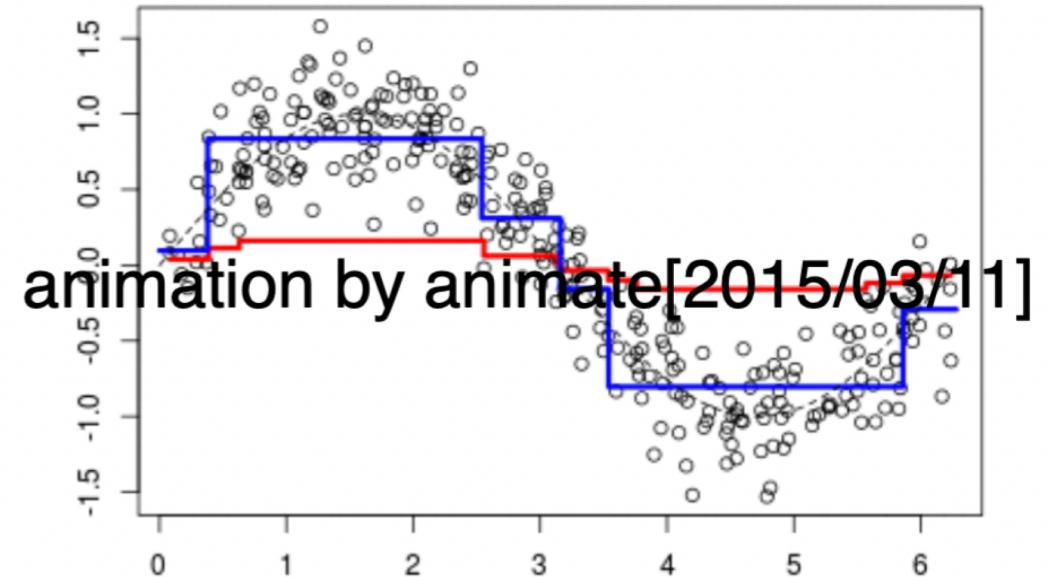
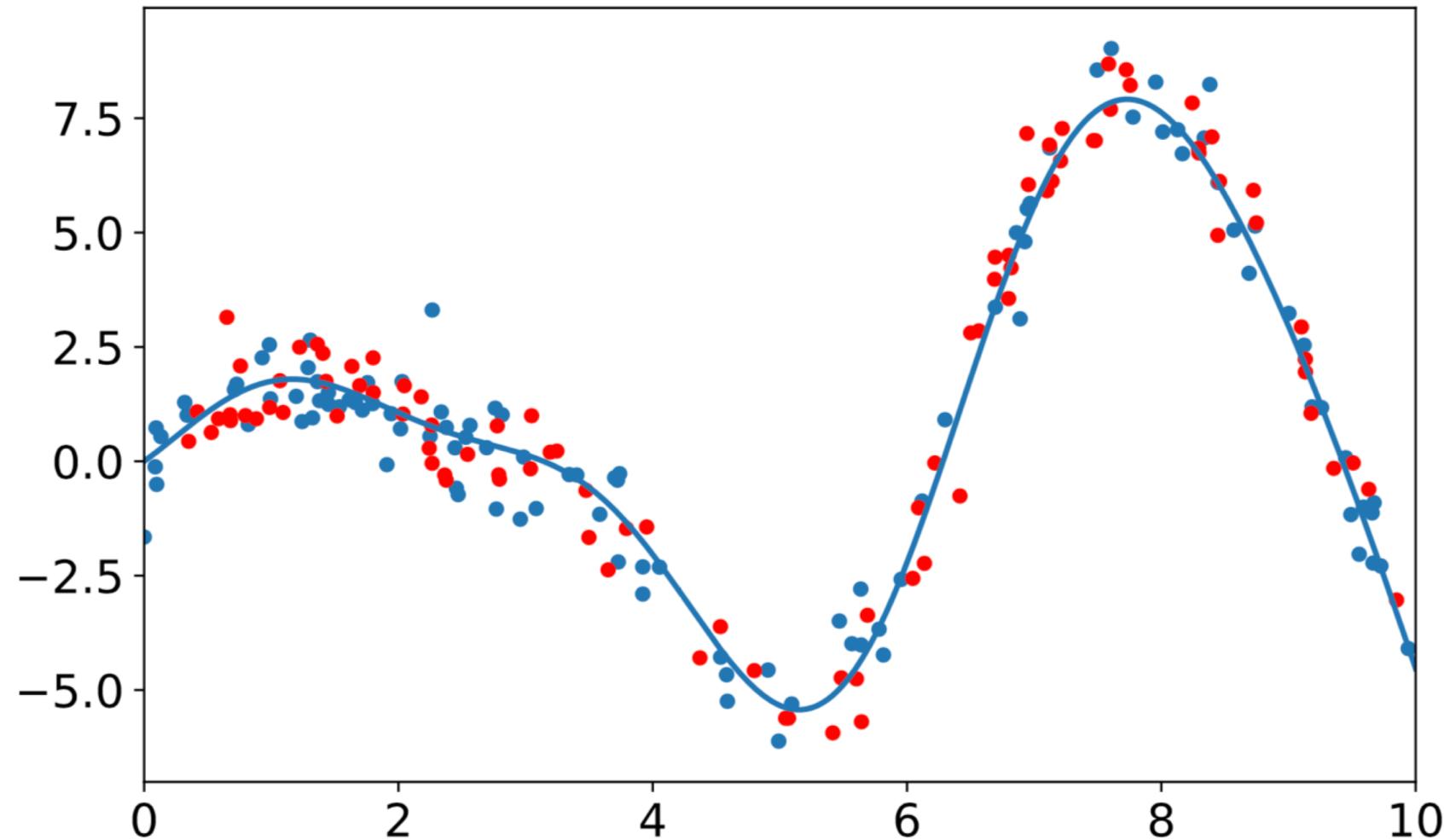
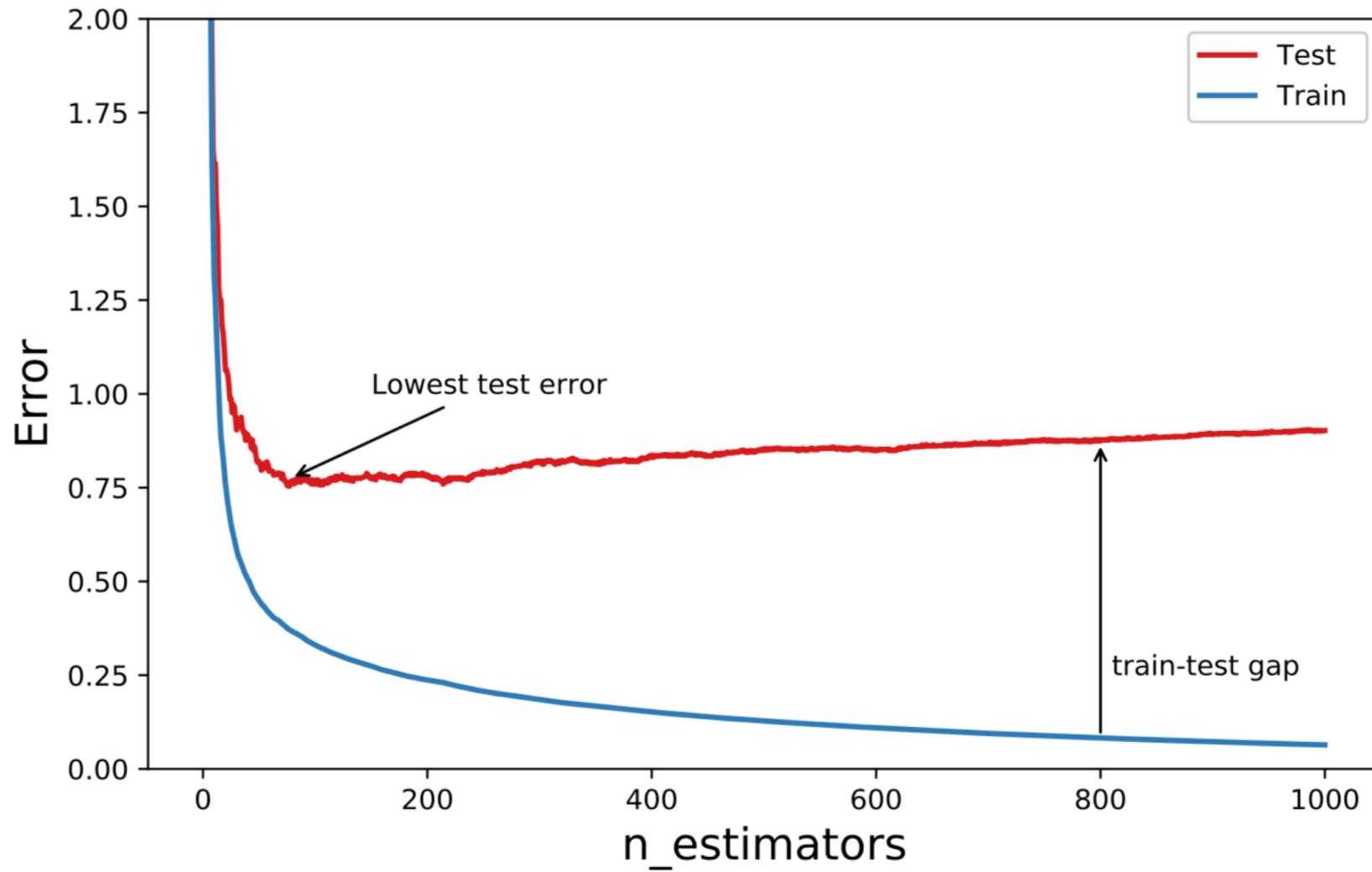


Figure: Low shrinkage

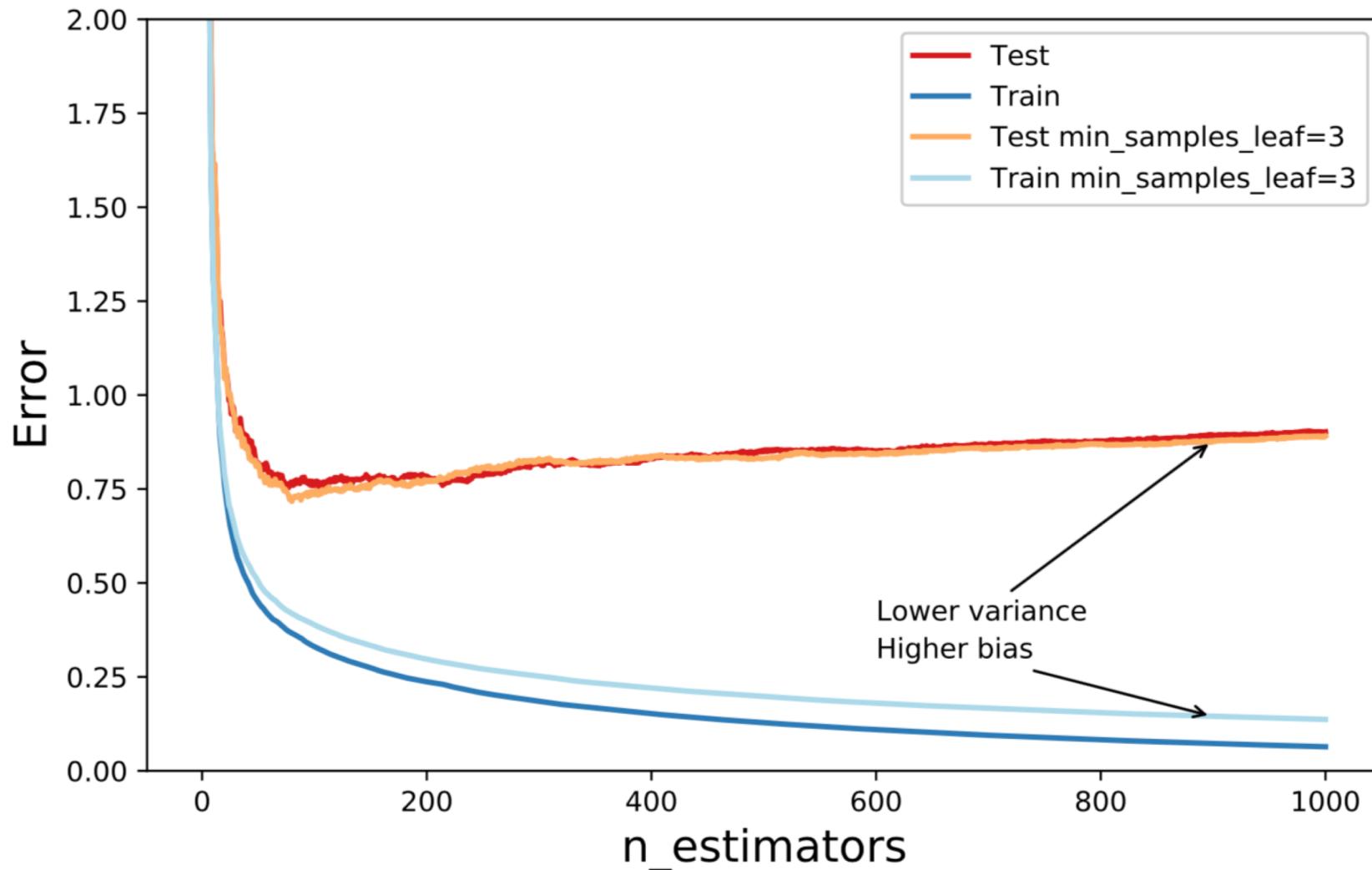
# GBM: regularization approaches



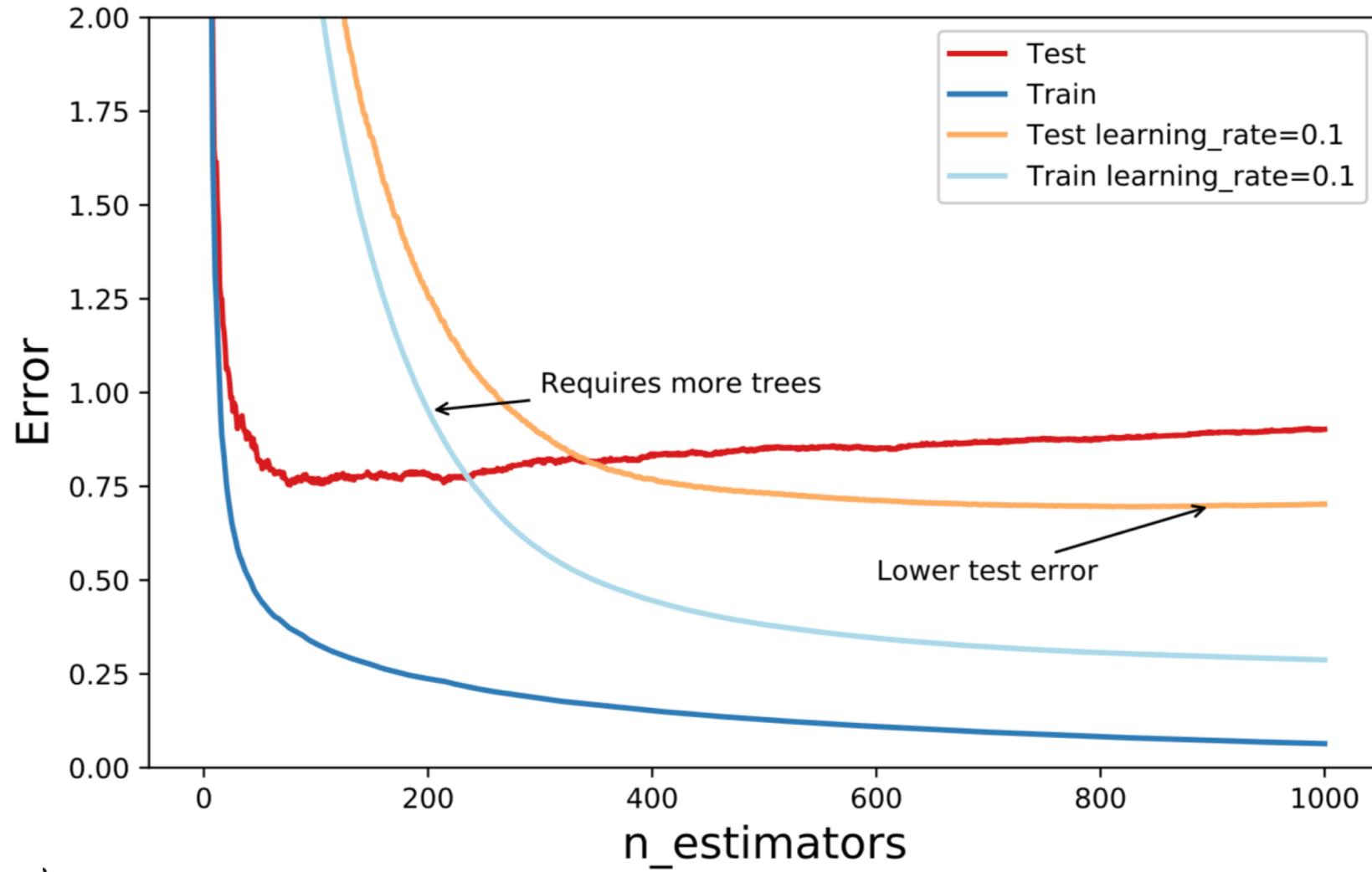
# GBM: regularization approaches



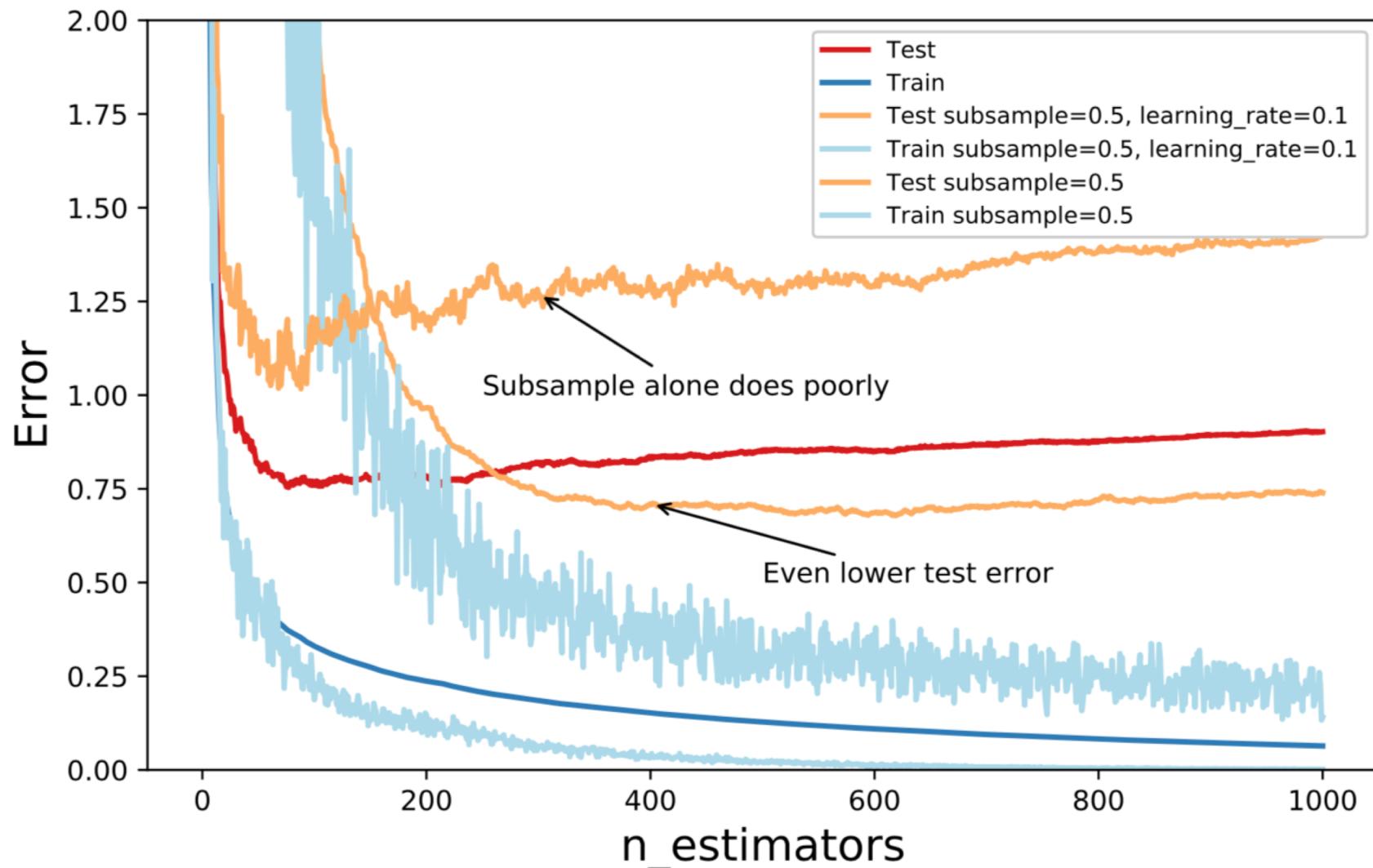
# GBM: regularization approaches



# GBM: regularization approaches



# GBM: regularization approaches



# XGBoost

# Extreme Gradient Boosting

1. Approximate the descent direction constructed using **second order derivatives**

$$\sum_{i=1}^{\ell} \left( -s_i h(\mathbf{x}_i) + \frac{1}{2} t_i h^2(\mathbf{x}_i) \right) \rightarrow \min_h, \quad t_i = \left. \frac{\partial^2}{\partial z^2} L(y_i, z) \right|_{a_{N-1}(\mathbf{x}_i)}$$

2. Penalize **large leaf counts**  $J$  and **large leaf coefficient norm**  $\|b\|_2^2 = \sum_{j=1}^J b_j^2$

$$\sum_{i=1}^{\ell} \left( -s_i h(x_i) + \frac{1}{2} t_i h^2(x_i) \right) + \gamma J + \frac{\lambda}{2} \sum_{j=1}^J b_j^2 \rightarrow \min_h$$

where  $b(\mathbf{x}) = \sum_{j=1}^J b_j [\mathbf{x} \in R_j]$

# Extreme Gradient Boosting

3. Choose split  $[\mathbf{x}_j < t]$  at node  $R$  to maximize

$$Q = H(R) - H(R_\ell) - H(R_r) \rightarrow \max,$$

where the impurity criterion

$$H(R) = -\frac{1}{2} \left( \sum_{(t_i, s_i) \in R} s_j \right)^2 / \left( \sum_{(t_i, s_i) \in R} t_j + \lambda \right) + \gamma$$

4. The stopping rule: declare the node a leaf if even the best split gives negative  $Q$

<https://xgboost.readthedocs.io/en/latest/model.html>

# Conclusion

- › **Boosting:** a general meta-algorithm aimed at composing a strong hypothesis from multiple weak hypotheses
- › Boosting can be applied for arbitrary losses, arbitrary problems (regression, classification) and over arbitrary weak learners
- › The **Gradient Boosting Machine:** a general approach to boosting adding weak learners that approximate gradient of the loss function
- › **AdaBoost:** gradient boosting with an exponential loss function resulting in reweighting training instances when adding weak learners
- › **XGBoost:** gradient boosting with second order optimization, penalized loss and particular choice of impurity criterion