Statistical Machine Learning (BE4M33SSU)
Lecture 13: Ensembling

Jan Drchal

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Overview

Topics covered in the lecture:

- Ensemble Methods
- Bias-Variance Decomposition
- Bagging
- Random Forests
- Boosting and Gradient Boosting
- Gradient Boosted Trees

Ensemble Methods

- Inspired in *Wisdom of the crowd*
  - (weighted) averaging or taking majority vote
  - cancelling effect of noise of individual opinions,
  - examples: politics, trial by jury (vs. trial by judge), sports (figure skating, gymnastics), Wikipedia, Quora, Stack Overflow, . . .
- Learning and aggregating multiple predictors
- Ensemble may be built using single or different types of predictors

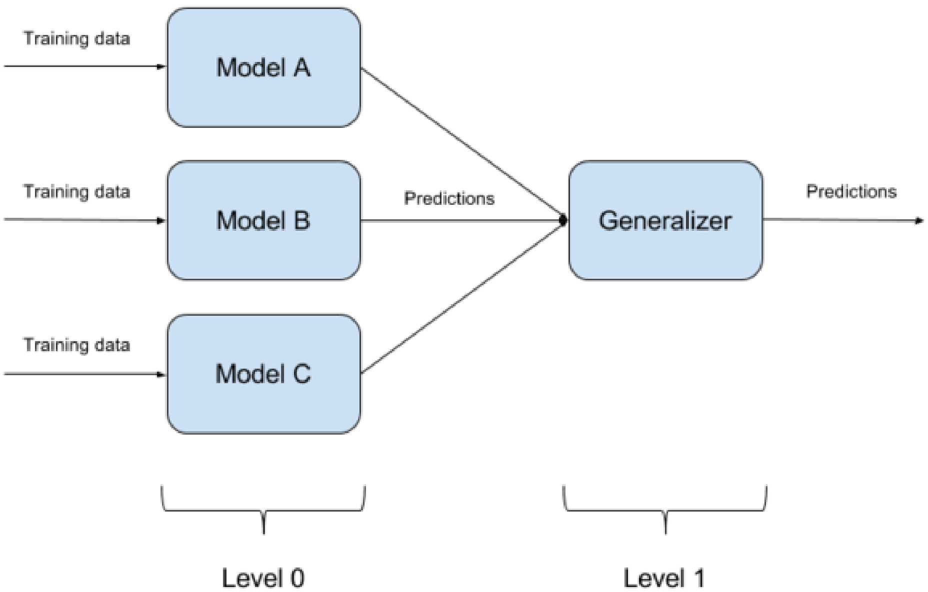
Wikimedia Commons

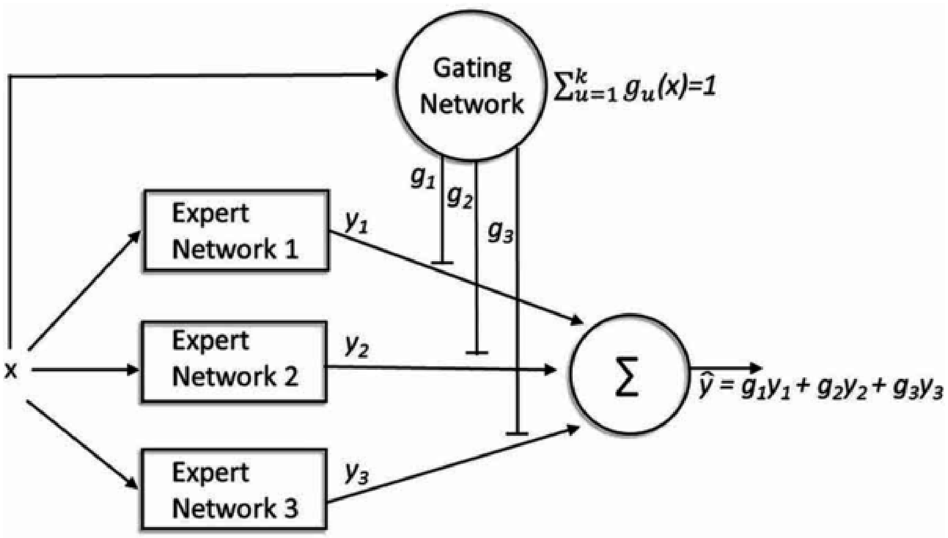
Ensembling Approaches

- Bagging (Bootstrap AGGregatING):
  - sample different training sets from the original training set
  - train *high variance low bias* predictors based on these sets and average them
  - exploits independence between predictors
- Boosting:
  - sequentially train *low variance high bias* predictors
  - subsequent predictors learn to fix the mistakes of the previous ones
  - exploits dependence between learners

Stacking and Mixture of Experts

- Combine *base-learners* with *meta-learner*





https://www.commonlounge.com/discussion/9331c0d004704e89bd4d1da08fd7c7bc

Prediction Problem: Expected Risk and Error Decomposition

Expected risk for data generated by $p(x, y)$:

$$R(h) = \mathbb{E}_{(x,y)\sim p}\Big[\ell(y, h(x))\Big]$$

- The best attainable (Bayes) risk is $R^* = \inf_{h \in \mathcal{Y}^{\mathcal{X}}} R(h)$
- The best predictor in $\mathcal{H}$ is $h_{\mathcal{H}} \in \operatorname{Argmin}_{h \in \mathcal{H}} R(h)$
- The predictor $h_m = A(\mathcal{T}^m)$ learned from $\mathcal{T}^m$ has risk $R(h_m)$

Excess error measures deviation of the learned predictor from the best one:

$$\underbrace{\Big(R(h_m) - R^*\Big)}_{\text{excess error}} = \underbrace{\Big(R(h_m) - R(h_{\mathcal{H}})\Big)}_{\text{estimation error}} + \underbrace{\Big(R(h_{\mathcal{H}}) - R^*\Big)}_{\text{approximation error}}$$

Risk Averaged over Datasets

- How will our predictor behave when sampling different training sets?
- We can define the errors considering average over models constructed using all possible datasets $\mathcal{T}^m$, i.e., $\mathbb{E}_{\mathcal{T}^m}\Big[R(h_m)\Big]$
- The errors can be redefined as:

$$\underbrace{\left(\mathbb{E}_{\mathcal{T}^m}\Big[R(h_m)\Big] - R^*\right)}_{\text{excess error}} = \underbrace{\left(\mathbb{E}_{\mathcal{T}^m}\Big[R(h_m)\Big] - R(h_{\mathcal{H}})\right)}_{\text{estimation error}} + \underbrace{\left(R(h_{\mathcal{H}}) - R^*\right)}_{\text{approximation error}}$$

Predictors Averaged over Datasets

- Let us also define a model averaged over all possible datasets:

$$g_m(x) = \mathbb{E}_{\mathcal{T}^m}\Big[h_m(x)\Big]$$

- Unlike individual h_m models, g_m has an access to the whole $p(x, y)$
- Note: in general $g_m \neq h_{\mathcal{H}}$ due to training algorithm A involved in h_m.
- Also: g_m can't be actually evaluated for infinite number of $\mathcal{T}^m$ datasets

Bias-Variance Decomposition for Regression

- Consider a regression problem with data generated as follows:

$$y = h^*(x) + \epsilon$$

where ϵ is noise: $\mathbb{E}[\epsilon] = 0$ and $\mathrm{Var}(\epsilon) = \sigma^2$, e.g., $\epsilon \sim \mathcal{N}(0, \sigma^2)$

- Use squared loss:

$$\ell(y, h(x)) = \Big(h(x) - y\Big)^2$$

- The optimal predictor $h^*(x)$ has a nonzero risk (for $\sigma^2 > 0$):

$$R^* = \mathbb{E}_{x,y}\left[\Big(h^*(x) - y\Big)^2\right] = \mathbb{E}_{\epsilon}\left[\epsilon^2\right] = \mathrm{Var}(\epsilon) = \sigma^2$$

Bias-Variance Decomposition for Regression 2

- The expected risk for h_m can be decomposed:

$$\mathbb{E}_{\mathcal{T}^m}\Big[R(h_m)\Big] = \mathbb{E}_{x,y,\mathcal{T}^m}\Big[\Big(h_m(x) - y\Big)^2\Big]$$

$$= \cdots$$

$$= \underbrace{\mathbb{E}_{x,\mathcal{T}^m}\Big[\Big(h_m(x) - g_m(x)\Big)^2\Big]}_{\text{variance}} +$$

$$+ \underbrace{\mathbb{E}_{x}\Big[\Big(g_m(x) - h^*(x)\Big)^2\Big]}_{\text{bias}^2} + \underbrace{\sigma^2}_{\text{noise}}$$

- The error splits into three terms
  - **variance**: difference of h_m from the averaged predictor g_m,
  - **bias**^{2}: difference of the averaged predictor g_m from the optimal one,
  - **noise**: irreducible determined by data

Excess Error vs. Bias and Variance

- The excess error is defined as:

$$\mathbb{E}_{\mathcal{T}^m}\Big[R(h_m)\Big] - R^*$$

- As $R^* = \sigma^2$ we get:

$$\mathbb{E}_{\mathcal{T}^m}\Big[R(h_m)\Big] - R^* = \underbrace{\mathbb{E}_x\Big[\Big(g_m(x) - h^*(x)\Big)^2\Big]}_{\text{bias}^2} + \underbrace{\mathbb{E}_{x,\mathcal{T}^m}\Big[\Big(h_m(x) - g_m(x)\Big)^2\Big]}_{\text{variance}}$$

- Compare
  - **bias**^{2} vs. approximation error,
  - **variance** vs. estimation error
  - averaged model g_m vs. best predictor $h_{\mathcal{H}}$

Derivation of the Bias-Variance Decomposition

$$
\begin{aligned}
\mathbb{E}_{\mathcal{T}^m}\Big[R(h_m)\Big] &= \mathbb{E}_{x,y,\mathcal{T}^m}\left[\Big(h_m(x) - y\Big)^2\right] \\
&= \mathbb{E}_{x,y,\mathcal{T}^m}\left[\Big(h_m(x) - g_m(x) + g_m(x) - y\Big)^2\right] \\
&= \mathbb{E}_{x,y,\mathcal{T}^m}\left[\Big(h_m(x) - g_m(x)\Big)^2 + \Big(g_m(x) - y\Big)^2\right. \\
&\qquad \left. + 2\Big(h_m(x) - g_m(x)\Big)\Big(g_m(x) - y\Big)\right] \\
&= \mathbb{E}_{x,\mathcal{T}^m}\left[\Big(h_m(x) - g_m(x)\Big)^2\right] + \mathbb{E}_{x,y}\left[\Big(g_m(x) - y\Big)^2\right] \\
&\qquad + \mathbb{E}_{x,y}\left[2\Big(\underbrace{\mathbb{E}_{\mathcal{T}^m}\Big[h_m(x)\Big]}_{g_m(x)} - g_m(x)\Big)\Big(g_m(x) - y\Big)\right]
\end{aligned}
$$

Derivation of the Bias-Variance Decomposition 2

We get:

$$\mathbb{E}_{\mathcal{T}^m}\Big[R(h_m)\Big] = \underbrace{\mathbb{E}_{x,\mathcal{T}^m}\Big[\Big(h_m(x) - g_m(x)\Big)^2\Big]}_{\text{variance}} + \mathbb{E}_{x,y}\Big[\Big(g_m(x) - y\Big)^2\Big]$$

$$= \mathrm{Var}_{x,\mathcal{T}^m}\Big(h_m(x)\Big) + \mathbb{E}_{x,y}\Big[\Big(g_m(x) - y\Big)^2\Big]$$

Note that the second term does not depend on $\mathcal{T}^m$.

Derivation of the Bias-Variance Decomposition 3

Let us continue with the second term:

$$
\begin{aligned}
\mathbb{E}_{x,y}\left[\Big(g_m(x) - y\Big)^2\right] &= \mathbb{E}_{x,\epsilon}\left[\Big(g_m(x) - h^*(x) - \epsilon\Big)^2\right] \\
&= \mathbb{E}_{x,\epsilon}\left[\Big(g_m(x) - h^*(x)\Big)^2 + \epsilon^2 - 2\epsilon\Big(g_m(x) - h^*(x)\Big)\right] \\
&= \mathbb{E}_{x}\left[\Big(g_m(x) - h^*(x)\Big)^2\right] + \mathbb{E}_{\epsilon}\left[\epsilon^2\right] \\
&\quad \underbrace{-2\mathbb{E}_{x,\epsilon}\left[\epsilon\Big(g_m(x) - h^*(x)\Big)\right]}_{=0} \\
&= \underbrace{\mathbb{E}_{x}\left[\Big(g_m(x) - h^*(x)\Big)^2\right]}_{\text{bias}^2} + \underbrace{\sigma^2}_{\text{noise}}
\end{aligned}
$$

Pointwise Bias-Variance

We can express the bias and variance as function of x by not integrating over in expected values

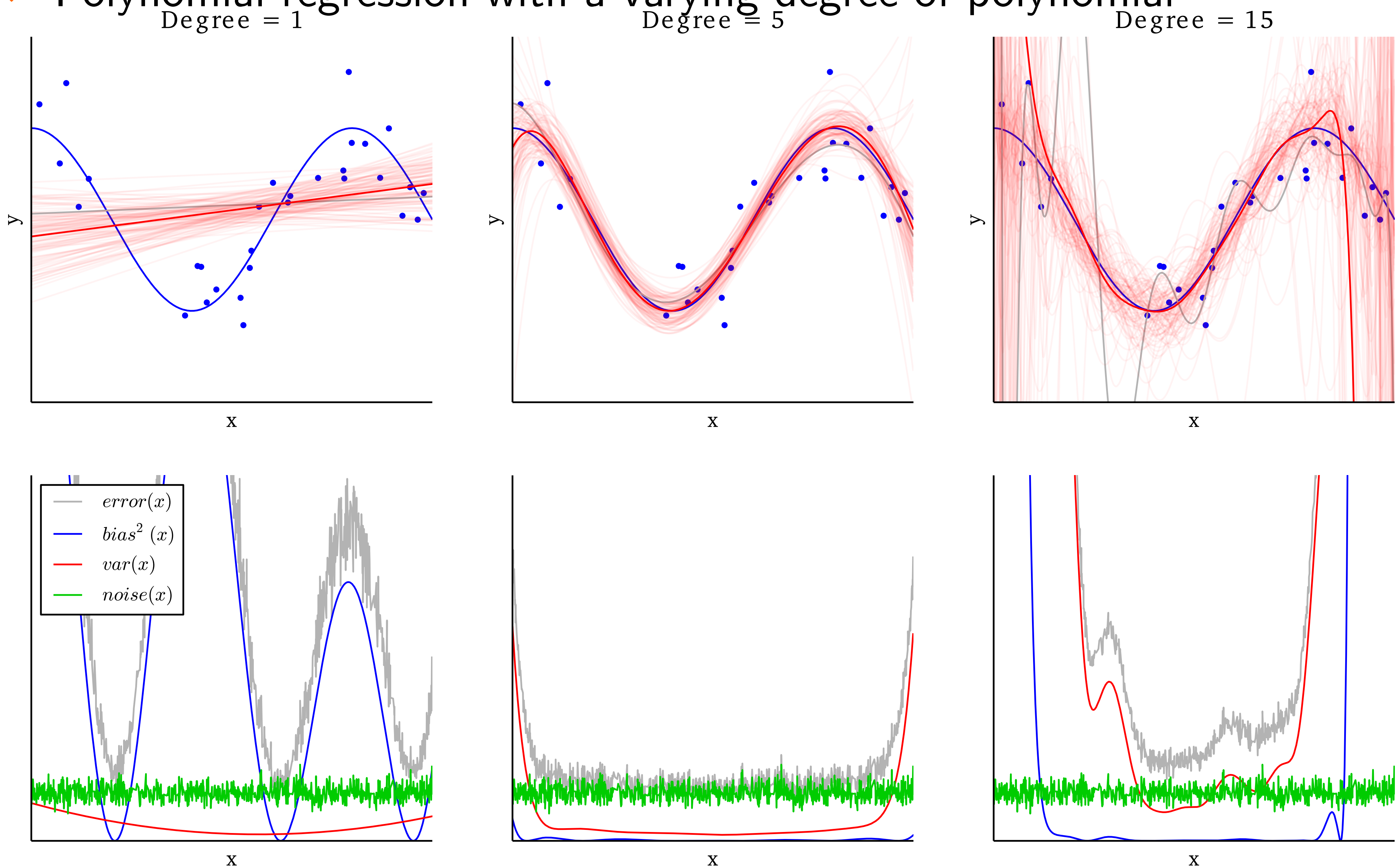
$$\mathbb{E}_{y|x,\mathcal{T}^m}\Big[\ell(y, h_m(x))\Big] = \mathbb{E}_{y|x,\mathcal{T}^m}\Big[\Big(h_m(x) - y\Big)^2\Big]$$

$$= \underbrace{\mathrm{Var}_{\mathcal{T}^m}\Big(h_m(x)\Big)}_{\text{variance(x)}} +$$

$$+ \underbrace{\Big(g_m(x) - h^*(x)\Big)^2}_{\text{bias(x)}^2} + \underbrace{\sigma(x)^2}_{\text{noise}}$$

Bias-Variance: Example

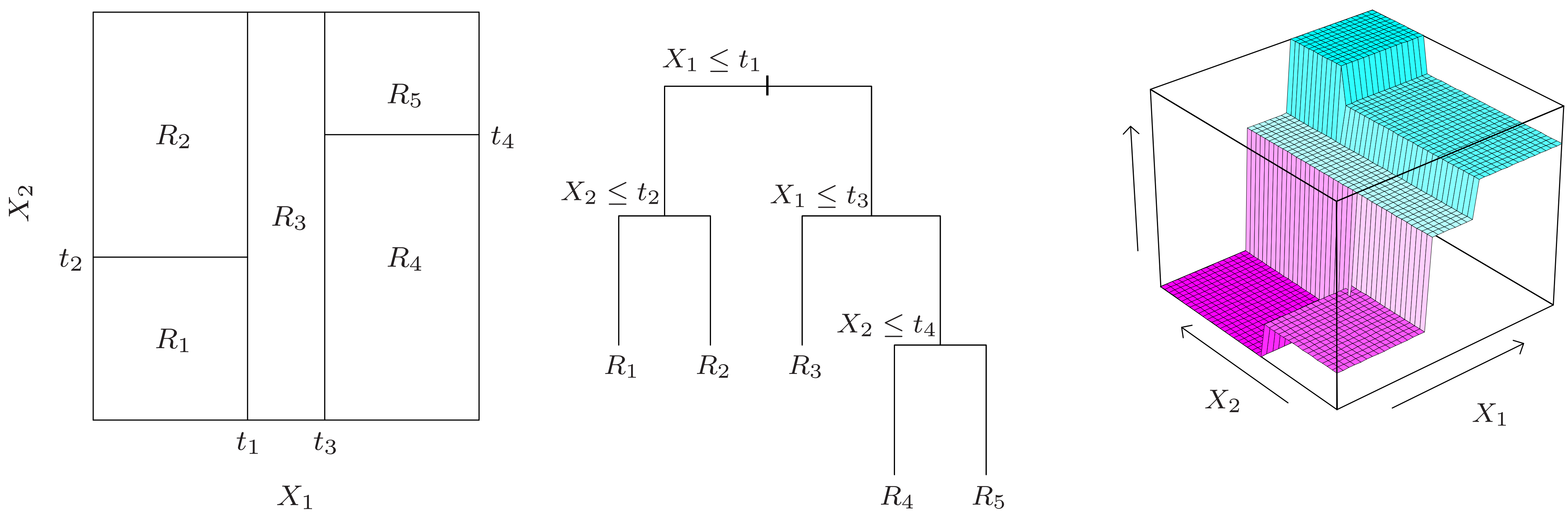
- Polynomial regression with a varying degree of polynomial



Gilles Louppe: *Understanding Random Forests: From Theory to Practice*, 2014

Decision/Regression Trees

- Nodes at the same level correspond to mutually exclusive subsets of the original training data as well as mutually exclusive subsets of the input space $\mathcal{X}$
- Inner node further splits its subset



Hastie et al.: *The Elements of Statistical Learning*, 2009

Decision/Regression Trees (contd.)

- Training set: $\mathcal{T}^m = \{(\boldsymbol{x}_i, y_i) \mid i = 1, \ldots, m\}$, $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$
- Input space split into regions defined in leaves: R_r, $r \in \{1, \ldots, M\}$
- We can model *region responses* by constants c_r, $r \in \{1, \ldots, M\}$ but other possibilities, e.g., linear regression are possible
- Prediction:

$$h(\boldsymbol{x}) = \sum_{r=1}^{M} c_r [\![\boldsymbol{x} \in R_r]\!]$$

- For sum of squares *loss function* $\sum_{i=1}^{m} (y_i - h(\boldsymbol{x}_i))^2$ we set the responses to be the averages over regions:

$$\hat{c}_r = \frac{1}{|S_r|} \sum_{(\boldsymbol{x}_i, y_i) \in S_r} y_i \qquad \text{(see seminar)}$$

where $S_r = \{(\boldsymbol{x}_i, y_i) : (\boldsymbol{x}_i, y_i) \in \mathcal{T}^m \wedge \boldsymbol{x}_i \in R_r\}$

Greedy Learning of Decision/Regression Trees

- How many distinct decision trees with p Boolean attributes for binary classification?
  - at least as many as boolean functions of p attributes
  - $=$ number of distinct truth tables with 2^p rows: 2^{2^p}
  - For 6 Boolean attributes at least 18,446,744,073,709,551,616 trees!
- Learning is NP-complete: [Hyafil and Rivest 1976]
- We need heuristics $\Rightarrow$ **greedy approach**
- Recursively choose the "most important" attribute to find a small tree consistent with the training data
- Split points:
  - **nominal attribute**: try all possibilities
  - **ordinal/continuous attribute**: try attribute values based on all training data samples or their subset

Regression Trees: Which Attribute to Split?

- The "most important" attribute for regression trees would be the one, for which the split reduces the loss (sum of squared errors) by the greatest amount
- We have:

$$h(\boldsymbol{x}) = \sum_{r=1}^{M} c_r [\![\boldsymbol{x} \in R_r]\!]$$

- Consider splitting attribute j and split point s, we split an original region R into a pair of half-planes for an ordinal (e.g., continuous) attribute:

$$R_L(j,s) = \{\boldsymbol{x} | \boldsymbol{x} \in R \wedge x_j \leq s\} \text{ and } R_R(j,s) = \{\boldsymbol{x} | \boldsymbol{x} \in R \wedge x_j > s\}$$

similarly for a nominal attribute:

$$R_L(j,s) = \{\boldsymbol{x} | \boldsymbol{x} \in R \wedge x_j = s\} \text{ and } R_R(j,s) = \{\boldsymbol{x} | \boldsymbol{x} \in R \wedge x_j \neq s\}$$

- Denote the corresponding subsets of of $\mathcal{T}^m$ as S_L and S_R

Regression Trees: Which Attribute to Split? (contd.)

- We seek for an attribute j and a split point s which minimize the total *impurity* (=loss, risk):

$$\min_{c_L} \sum_{(\boldsymbol{x}_i, y_i) \in S_L(j,s)} (y_i - c_L)^2 + \min_{c_R} \sum_{(\boldsymbol{x}_i, y_i) \in S_R(j,s)} (y_i - c_R)^2$$

for $(\boldsymbol{x}_i, y_i) \in S$ and $S = S_L \cup S_R$

- Inner minimizations (*region response* values) are solved by averaging tree outputs per region:

$$\hat{c}_L = \frac{1}{|S_L(j,s)|} \sum_{(\boldsymbol{x}_i, y_i) \in S_L(j,s)} y_i \quad \text{and} \quad \hat{c}_R = \frac{1}{|S_R(j,s)|} \sum_{(\boldsymbol{x}_i, y_i) \in S_R(j,s)} y_i$$

- Root node: $S = \mathcal{T}^m$

Tree Learning Algorithm

$\text{BUILD-TREE}(S)$

$i = \text{IMPURITY}(S)$ // e.g., the squared loss
$\hat{i}, \hat{j}, \hat{s}, \hat{S}_L, \hat{S}_R = 0, 0, 0, \emptyset, \emptyset$ // current best kept in these
for $j \in \{1, \ldots, p\}$ // iterate over attributes
  for $s \in \text{SPLIT-POINTS}(S, j)$ // iterate over all split points
    $S_L, S_R = \text{SPLIT}(S, j, s)$
    $i_L = \text{IMPURITY}(S_L)$
    $i_R = \text{IMPURITY}(S_R)$
    if $i_L + i_R < \hat{i}$ and $|S_L| > 0$ and $|S_R| > 0$
      $\hat{i}, \hat{j}, \hat{s}, \hat{S}_L, \hat{S}_R = (i_L + i_R), j, s, S_L, S_R$
if $\hat{i} < i$
  $N_L = \text{BUILD-TREE}(\hat{S}_L)$
  $N_R = \text{BUILD-TREE}(\hat{S}_R)$
  return $\text{DECISION-NODE}(\hat{j}, \hat{s}, N_L, N_R)$
else return $\text{LEAF-NODE}(S)$

Bias and Variance of Decision Trees

- Small changes of training data lead to big differences in final trees
- Decision trees grown deep enough have typically:
  - low bias
  - high variance

  $\Rightarrow$ **overfitting**
- Idea: *average multiple models* to reduce variance while (happily) not increasing bias much

Averaging Models

- Define *bagging model* b as an average of K *component models*:

$$b(x) = \frac{1}{K}\sum_{i=1}^{K} h_m^{(i)}(x)$$

trained using a set of i.i.d. datasets of size m: $\mathcal{D}^m = \{\mathcal{T}_1^m, \ldots, \mathcal{T}_K^m\}$ so $h_m^{(1)}(x)$ is trained using $\mathcal{T}_1^m$, $h_m^{(2)}(x)$ using $\mathcal{T}_2^m$, etc.

- Note that $b(x)$ approximates the *averaging model*:

$$g_m(x) = \mathbb{E}_{\mathcal{T}^m}\Big[h_m(x)\Big]$$

- We can define the *averaging model* for $b(x)$ as well:

$$g_m^B(x) = \mathbb{E}_{\mathcal{D}^m}\Big[b(x)\Big]$$

Averaging Models: Bias

- Bias remains unchanged for the *bagging model* compared to any of the *component models*:

$$
\begin{aligned}
\text{bias}(x)^2 &= \Big(g_m^B(x) - h^*(x)\Big)^2 \\
&= \bigg(\mathbb{E}_{\mathcal{D}^m}\Big[b(x)\Big] - h^*(x)\bigg)^2 \\
&= \bigg(\mathbb{E}_{\mathcal{D}^m}\Big[\frac{1}{K}\sum_{i=1}^{K} h_m^{(i)}(x)\Big] - h^*(x)\bigg)^2 \\
&= \bigg(\frac{1}{K}\sum_{i=1}^{K}\mathbb{E}_{\mathcal{T}_i^m}\Big[h_m^{(i)}(x)\Big] - h^*(x)\bigg)^2 \\
&= \bigg(\mathbb{E}_{\mathcal{T}^m}\Big[h_m(x)\Big] - h^*(x)\bigg)^2 = \Big(g_m(x) - h^*(x)\Big)^2
\end{aligned}
$$

Averaging Models: Variance

- For uncorrelated component models $h_m^{(i)}(x)$:

$$\mathrm{Var}_{\mathcal{D}^m}\Big(b(x)\Big) = \mathrm{Var}_{\mathcal{D}^m}\left(\frac{1}{K}\sum_{i=1}^{K} h_m^{(i)}(x)\right)$$

$$= \frac{1}{K^2}\sum_{i=1}^{K}\mathrm{Var}_{\mathcal{T}_i^m}\left(h_m^{(i)}(x)\right) = \frac{1}{K}\mathrm{Var}_{\mathcal{T}^m}\Big(h_m(x)\Big)$$

  which is a great improvement based on the very **strong** assumption

- There is no improvement for maximum correlation, i.e., for all component models equal: $h_m^{(i)}(x) = h_m(x)$ for $i = 1, \ldots, K$, we get:

$$\mathrm{Var}_{\mathcal{D}^m}\Big(b(x)\Big) = \mathrm{Var}_{\mathcal{D}^m}\left(\frac{1}{K}\sum_{i=1}^{K} h_m^{(i)}(x)\right) = \mathrm{Var}_{\mathcal{T}^m}\Big(h_m(x)\Big)$$

$\Rightarrow$ we need to train **uncorrelated** (diverse) component models while **keeping their bias reasonably low**

Bootstrapping

- In practice we have only a single training dataset $\mathcal{T}^m$
- Bootstrapping is a method producing datasets $\mathcal{T}_i^m$ for $i = 1, \ldots K$ by sampling $\mathcal{T}^m$ uniformly with *replacement*
- Bootstrap datasets have the same size as the original dataset $|\mathcal{T}_i^m| = |\mathcal{T}^m|$
- $\mathcal{T}_i^m$ is expected to have the fraction $1 - \frac{1}{e} \approx 63.2\%$ of unique samples from $\mathcal{T}^m$, others are duplicates (see seminar)

Bagging

- Bagging = Bootstrap AGGregating [Breiman 1994]:
  1. Use bootstrapping to generate K datasets
  2. Train a model $h_m^{(i)}(x)$ on each dataset $\mathcal{T}_i^m$
  3. Average the models getting the bagging model $b(x)$
- When decision trees are used as the models $\Rightarrow$ **random forests**
- Low bias is achieved by growing the trees to maximal depth
- Trees are decorrelated by:
  - training each tree on a different bootstrap dataset
  - randomization of split attribute selection

Random Forest Algorithm

1. For $i = 1 \ldots K$:
   (a) draw a bootstrap dataset $\mathcal{T}_i^m$ from $\mathcal{T}^m$, $|\mathcal{T}_i^m| = |\mathcal{T}^m| = m$
   (b) grow a tree $h_m^{(i)}$ using $\mathcal{T}_i^m$ by recursively repeating the following, until the minimum node size n_{min} is reached:
      i. select k attributes at random from the p attributes
      ii. pick the best attribute and split-point among the k
      iii. split the node into two daughter nodes
2. Output ensemble of trees $b(x)$ averaging $h_m^{(i)}(x)$ (regression) or selecting a majority vote (classification)

- Node size n_{min} is the number of the training dataset samples associated with the node, limits tree depth

Out-of-Bag (OOB) Error

- Cheap way of generalization error assessment for bagging
- Bagging produces bootstrapped sets $\mathcal{T}_1^m, \mathcal{T}_2^m, \ldots \mathcal{T}_K^m$
- For each $(\boldsymbol{x}_i, y_i) \in \mathcal{T}^m$ select only trees which were not trained on this sample: $H_i = \{h_m^{(j)} \mid (\boldsymbol{x}_i, y_i) \notin \mathcal{T}_j^m\}$
- Average only the OOB trees in H_i when evaluating error for $(\boldsymbol{x}_i, y_i)$
- Replacement for K-fold cross-validation

Feature Importance

- Random forests allow easy evaluation of feature importances
- Mean Decrease Impurity (MDI):
  - set $f_j = 0$ for all attributes $j = 1, \ldots, p$
  - traverse all trees processing all internal nodes
  - for each node having a split attribute j add its *impurity decrease* multiplied by the proportion of the *node size* to f_j
- Mean Decrease Accuracy (MDA), permutaion importance:
  - evaluate the forest using OOB
  - do the same with permuted values of an attribute j
  - watch decrease in accuracy: low decrease means unimportant feature

Random Forest Summary

- Easy to use method: robust w.r.t. parameter settings (K, node size)
- While *statistical consistency* is proven for decision trees (both regression and classification) we have only proofs for simplified versions of random forests [Breiman, 1984]
- Related methods: boosted trees

Boosting

- Sequentially train weak learners/predictors *low variance high bias*
- Subsequent predictors fix the mistakes of the previous ones reducing bias
- Methods discussed here:
  - Forward Stagewise Additive Modeling
  - Gradient Boosting Machine
  - Gradient Boosted Trees
  - AdaBoost

Forward Stagewise Additive Modeling (FSAM)

1. Initialize $f_0(x) = 0$
2. For $k = 1$ to K:
   (a) Find
   $$(\beta_k, \theta_k) = \operatorname*{argmin}_{\beta,\theta} \sum_{i=1}^{m} \ell\Big(y_i, f_{k-1}(x_i) + \beta b(x_i; \theta)\Big)$$
   where $b(x_i; \theta_k)$ is the *basis function* and β_k the corresponding coefficient
   (b) Set $f_k(x) = f_{k-1}(x) + \beta_k b(x; \theta_k)$
3. Return $h_m(x) = f_K(x)$

FSAM and Gradient Descent

- FSAM update looks very similar to the gradient descent one:

$$f_k(x) = f_{k-1}(x) + \beta_k b(x; \theta_k)$$

- Just think of
  - $\beta_k \approx$ step size (learning rate)
  - $b(x_i; \theta_k) \approx$ the negative of gradient

FSAM for Squared Loss

- Once again, consider regression with the squared loss:

$$\ell(y, f(x)) = (y - f(x))^2$$

- For FSAM we get:

$$\begin{aligned} \ell(y_i, f_k(x_i)) &= \ell(y_i, f_{k-1}(x_i) + \beta_k b(x_i; \theta_k)) \\ &= (y_i - f_{k-1}(x_i) - \beta_k b(x_i; \theta_k))^2 \\ &= (r_{ik} - \beta_k b(x_i; \theta_k))^2 \end{aligned}$$

  where $r_{ik} = y_i - f_{k-1}(x_i)$ is the *residual* of the current model for the i-th sample

- The task of FSAM is to fit the model $\beta_k b(x_i; \theta_k)$ to match the residuals
- The method is sometimes called the *least-squares boosting*

Gradient Boosting for Regression

- In case of regression with squared loss we minimize:

$$\mathcal{L} = \sum_{i=1}^{m} \ell(y_i, f(x_i)) = \sum_{i=1}^{m} \frac{1}{2}(y_i - f(x_i))^2,$$

- We can treat $f(x_1), f(x_2), \ldots, f(x_m)$ as parameters and take the derivatives:

$$\frac{\partial \mathcal{L}}{\partial f(x_i)} = \frac{\partial \left(\sum_{j=1}^{m} \ell(y_j, f(x_j)) \right)}{\partial f(x_i)} = \frac{\partial \ell(y_i, f(x_i))}{\partial f(x_i)}$$
$$= f(x_i) - y_i = -r_i$$

- The *least-squares boosting* hence takes steps in the negative gradient direction where $r_i = -\frac{\partial \mathcal{L}}{\partial f(x_i)}$
- This approach can be generalized for any differentiable loss function!

Gradient Boosting Machine

1. Initialize $f_0(x) = 0$ or $f_0(x) = \operatorname{argmin}_\gamma \sum_{i=1}^m \ell(y_i, \gamma)$

2. For $k = 1$ to K:

   (a) Compute:

$$\boldsymbol{g}_k = \left[\frac{\partial \ell(y_i, f_{k-1}(x_i))}{\partial f_{k-1}(x_i)}\right]_{i=1}^m$$

   (b) Fit a regression model $b(\cdot;\theta)$ to $-\boldsymbol{g}_k$ using squared loss:

$$\theta_k = \operatorname*{argmin}_\theta \sum_{i=1}^m \left[(-\boldsymbol{g}_k)_i - b(x_i;\theta)\right]^2$$

   (c) Choose a fixed step size $\beta_k = \beta > 0$ or use line search:

$$\beta_k = \operatorname*{argmin}_{\beta>0} \sum_{i=1}^m \ell\Big(y_i, f_{k-1}(x_i) + \beta b(x_i;\theta_k)\Big)$$

   (d) Set $f_k(x) = f_{k-1}(x) + \beta_k b(x;\theta_k)$

3. Return $h_m(x) = f_K(x)$

Multinominal Classification: Gradient Boosting Machine

- Training examples: $\mathcal{T}^m = \{(x_i, y_i) \in (\mathcal{X} \times \mathcal{Y}) \mid i = 1, \ldots, m\}$, where $\mathcal{Y} = \{1, \ldots, C\}$
- Train one GBM for each of C target classes:

$$\boldsymbol{f}(x_i) \triangleq [f^c(x_i)]_{c=1}^{C}$$

- Use softmax to get the probability estimates: $p_{ic} \triangleq \sigma_c(\boldsymbol{f}(x_i))$
- Use multinominal cross-entropy as the loss:

$$\mathcal{L} = -\sum_{c=1}^{K} y_{ic} \log(p_{ic}),$$

  where $y_{ic} \triangleq [\![y_i = c]\!]$ holds one-hot encoded target classes
- We then have the following residuals:

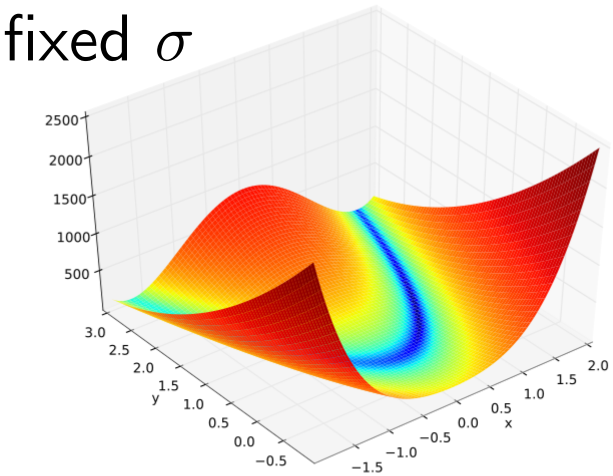
$$\frac{\partial \mathcal{L}}{\partial f^c(x_i)} = p_{ic} - y_{ic}$$

Gradient Boosted Trees

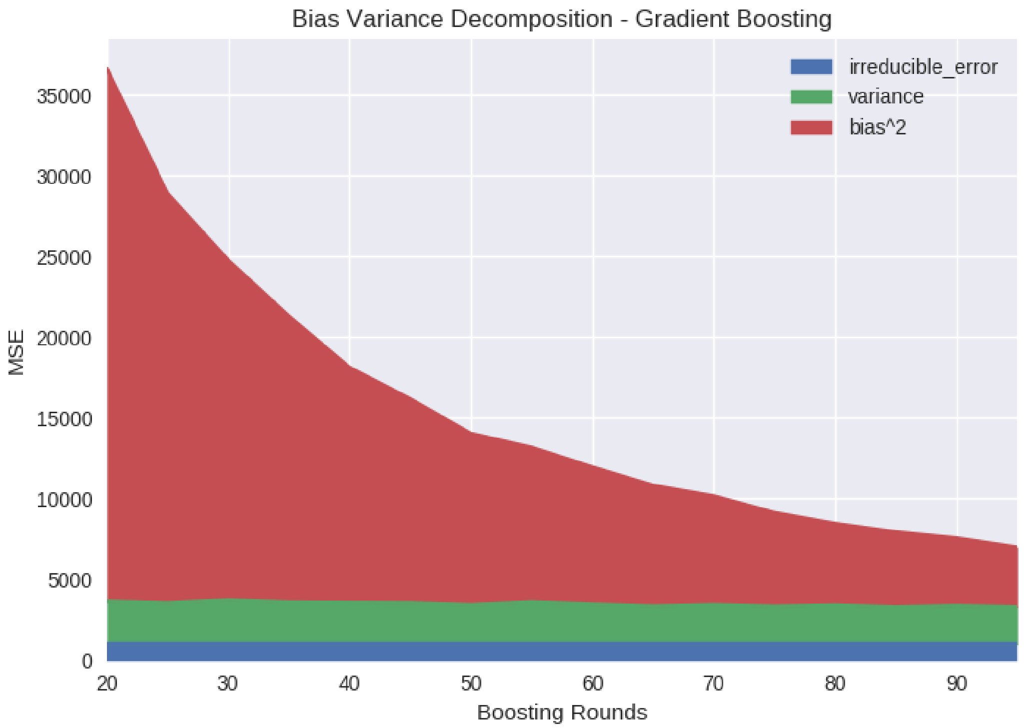
- Gradient Boosting Tree is GBM where all weak learners are decision or regression trees
- Use limit on depth/number of leaves/node size for the weak learners $\Rightarrow$ high bias
- Often single-level tree: *decision stump*
- Meta-parameters such as K (number of trees) and β (learning rate) have to be found using cross validation
- Model is built sequentially (unlike random forests)
- Highly optimized algorithms based on Gradient Boosting Trees:
  - XGBoost, LightGBM
  - parallelization, scalability, regularization

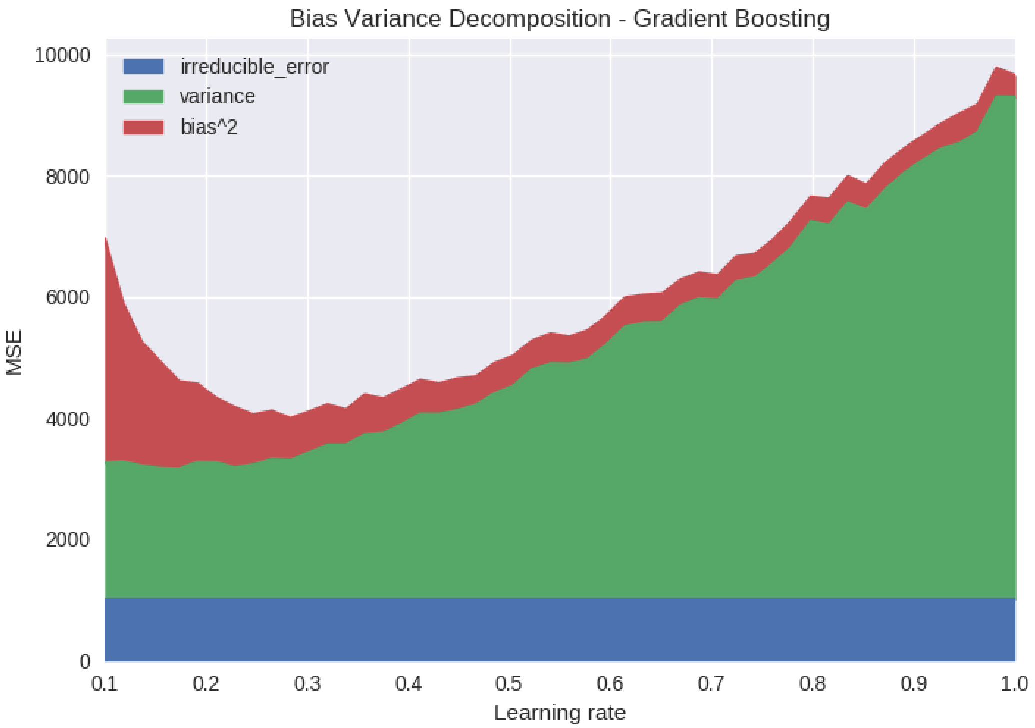
GBM Example (XGBoost)

- Each $\mathcal{T}^m$ is 1000 samples of *Rosenbrock function*, fixed σ
- 100 models for each setting
- Learning rate experiment: $K = 100$



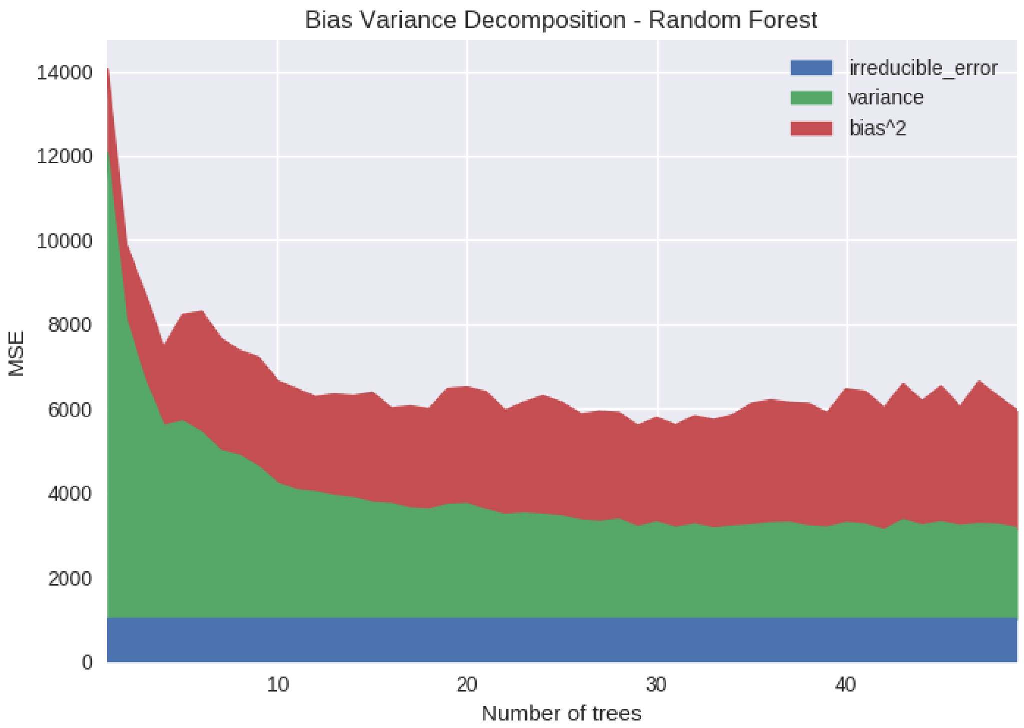
Wikimedia Commons

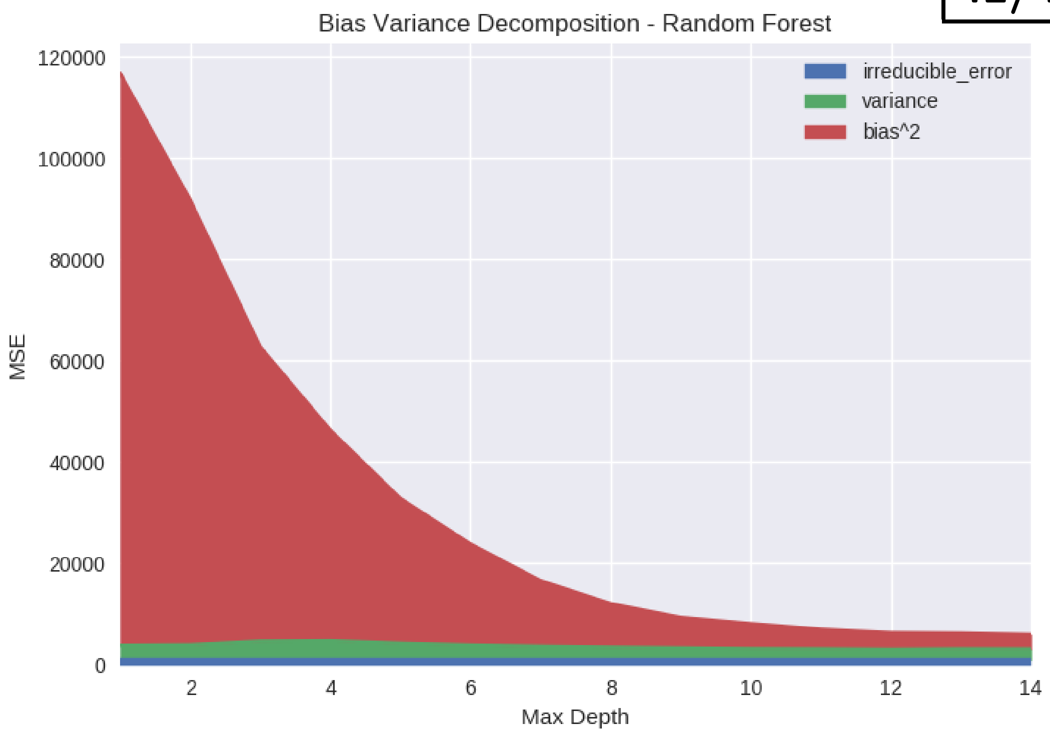


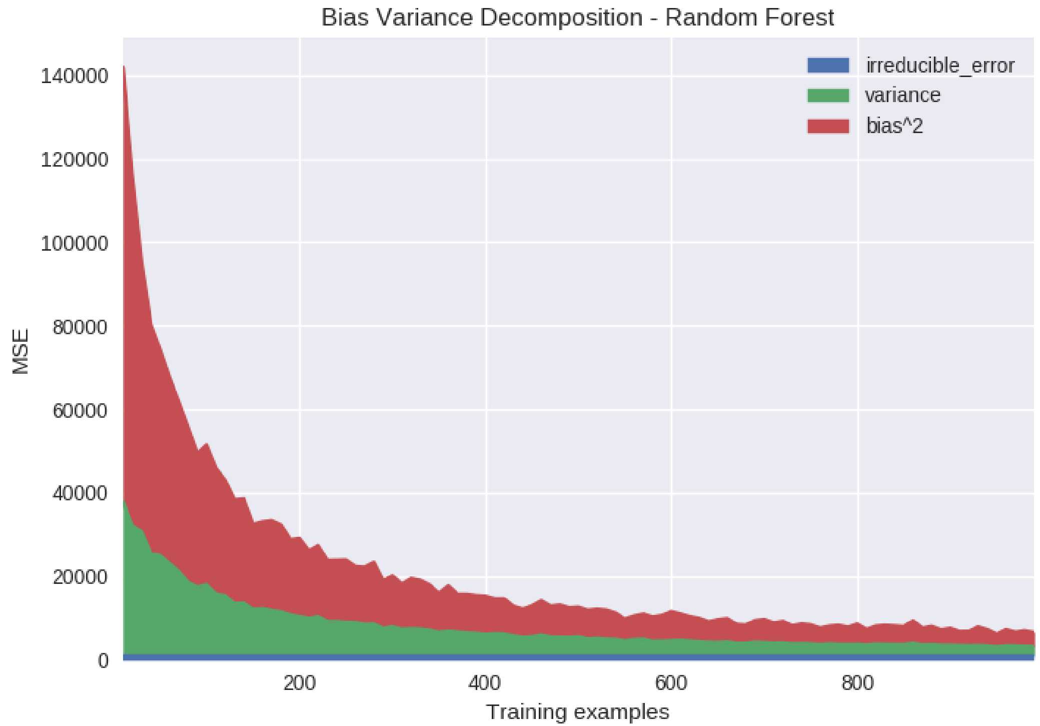


https://developer.nvidia.com/blog/bias-variance-decompositions-using-xgboost/

Comparison to Random Forest







AdaBoost M1

Binary classifier: $\mathcal{Y} = \{-1, 1\}$

1. Initialize the weights $w_i = 1/m$ for $i = 1, 2, \dots m$
2. For $k = 1$ to K:
   (a) Fit a classifier $G_k(x; \theta_k) \in \{-1, 1\}$ to the training data using loss weighted by w_i:

$$\theta_k = \operatorname*{argmin}_{\theta} \sum_{i=1}^{m} w_i [\![y_i \neq G_k(x_i; \theta)]\!]$$

   (b) Compute the weighted error rate

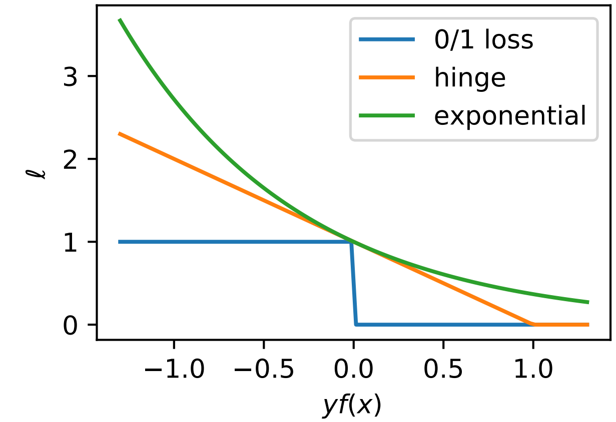
$$\epsilon_k = \frac{\sum_{i=1}^{m} w_i [\![y_i \neq G_k(x_i; \theta_k)]\!]}{\sum_{i=1}^{m} w_i}$$

   (c) Compute the scaling coefficient $\alpha_k = \log((1 - \epsilon_k)/\epsilon_k)$
   (d) Set $w_i \leftarrow w_i \cdot \exp(\alpha_k \cdot [\![y_i \neq G_k(x_i; \theta_k)]\!])$ for $i = 1, 2, \dots m$
3. Return $h_m(x) = \operatorname{sign}\left[\sum_{k=1}^{K} \alpha_k G_k(x; \theta_k)\right]$

AdaBoost is FSAM: the Loss

- Claim: AdaBoost is FSAM using the exponential loss

$$\ell(y, f(x)) = \exp(-yf(x))$$



- For individual classifiers $G_k(x_i; \theta)$ as basis functions, we get:

$$(\beta_k, \theta_k) = \operatorname*{argmin}_{\beta,\theta} \sum_{i=1}^{m} \ell\Big(y_i, f_{k-1}(x_i) + \beta G(x_i;\theta)\Big)$$

$$= \operatorname*{argmin}_{\beta,\theta} \sum_{i=1}^{m} \exp\left(-y_i\Big(f_{k-1}(x_i) + \beta G(x_i;\theta)\Big)\right)$$

$$= \operatorname*{argmin}_{\beta,\theta} \sum_{i=1}^{m} w_i^{(k)} \exp\left(-y_i \beta G(x_i;\theta)\right),$$

where $w_i^{(k)} \triangleq \exp(-y_i f_{k-1}(x_i))$ does not depend neither on β nor on θ

AdaBoost is FSAM II: Fitting the Classifier

- We can rearrange further:

$$
\begin{aligned}
(\beta_k, \theta_k) &= \operatorname*{argmin}_{\beta,\theta} \sum_{i=1}^{m} w_i^{(k)} \exp\Big(- y_i \beta G(x_i;\theta) \Big) \\
&= \operatorname*{argmin}_{\beta,\theta} \left[e^{-\beta} \sum_{y_i = G(x_i;\theta)} w_i^{(k)} + e^{\beta} \sum_{y_i \neq G(x_i;\theta)} w_i^{(k)} \right] \\
&= \operatorname*{argmin}_{\beta,\theta} \left[e^{-\beta} \sum_{i=1}^{m} w_i^{(k)} + \underbrace{(e^{\beta} - e^{-\beta})}_{>0 \text{ for } \beta>0} \sum_{i=1}^{m} w_i^{(k)} [\![y_i \neq G(x_i;\theta)]\!] \right]
\end{aligned}
$$

- For any $\beta > 0$ we can minimize θ separately:

$$
\theta_k = \operatorname*{argmin}_{\theta} \sum_{i=1}^{m} w_i^{(k)} [\![y_i \neq G(x_i;\theta)]\!] \quad \text{(same as } \textit{AdaBoost 2(a)}\text{)}
$$

AdaBoost is FSAM III: the Weighted Error ϵ_k and the Scaling Coefficient α_k

- Let's minimize

$$(e^{\beta} - e^{-\beta}) \sum_{i=1}^{m} w_i^{(k)} [\![y_i \neq G(x_i; \theta_k)]\!] + e^{-\beta} \sum_{i=1}^{m} w_i^{(k)}$$

with respect to β

$$(e^{\beta_k} + e^{-\beta_k}) \sum_{i=1}^{m} w_i^{(k)} [\![y_i \neq G(x_i; \theta_k)]\!] - e^{-\beta_k} \sum_{i=1}^{m} w_i^{(k)} = 0$$

$$(e^{\beta_k} + e^{-\beta_k}) \epsilon_k - e^{-\beta_k} = 0$$

where $\epsilon_k = \frac{\sum_{i=1}^{m} w_i [\![y_i \neq G(x_i;\theta_k)]\!]}{\sum_{i=1}^{m} w_i}$ as in *AdaBoost 2(b)*

- Solving for β_k:

$$\beta_k = \frac{1}{2} \log \frac{1 - \epsilon_k}{\epsilon_k}$$

- Define $\alpha_k \triangleq 2\beta_k$ and compare to *AdaBoost 2(c)*

AdaBoost is FSAM IV: the Weight Update

- We have $w_i^{(k)} = e^{-y_i f_{k-1}(x_i)}$ and $f_k(x) = f_{k-1}(x) + \beta_k G(x;\theta_k)$ so:

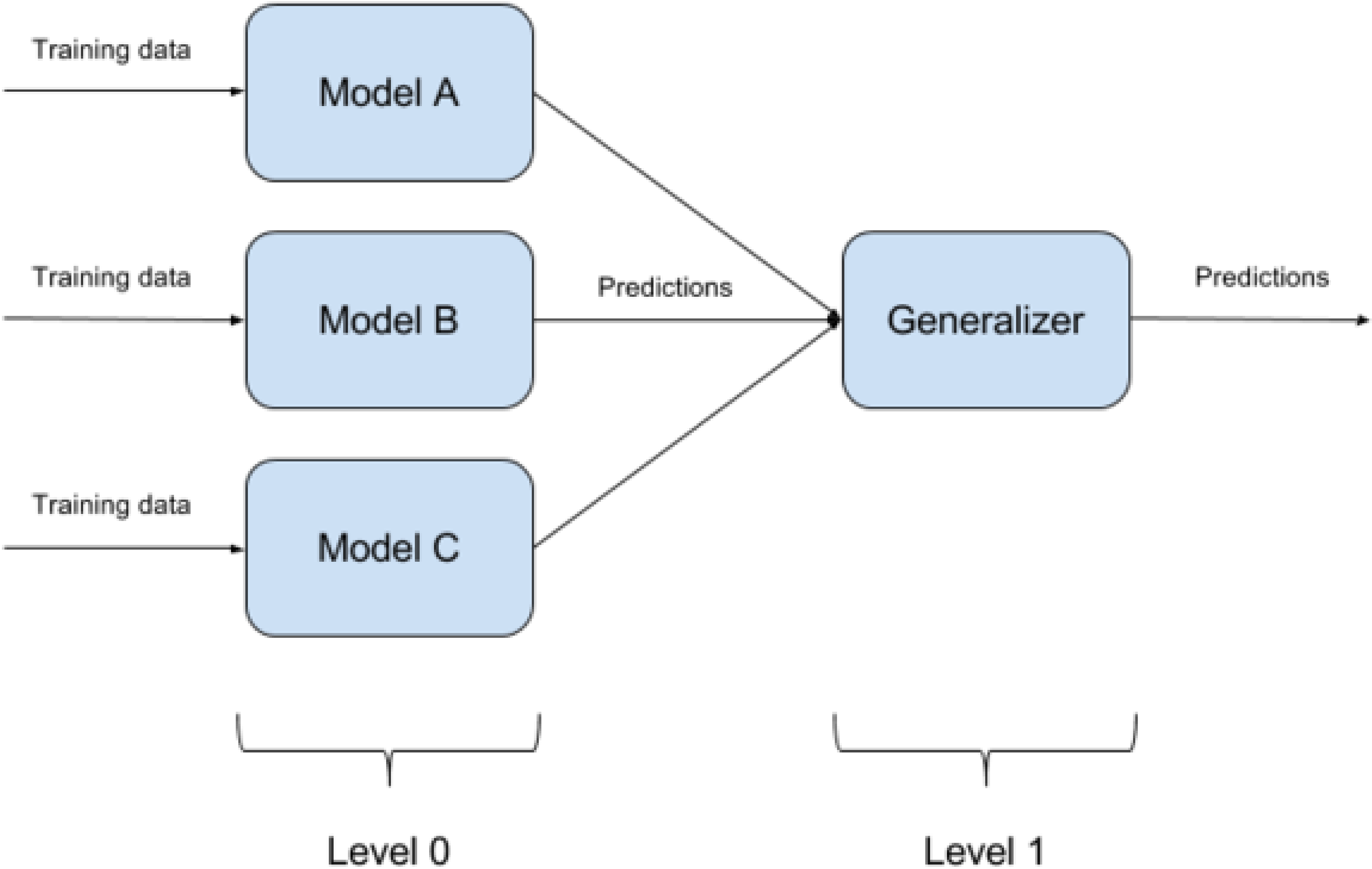
$$w_i^{(k+1)} = e^{-y_i\left(f_{k-1}(x_i)+\beta_k G(x_i;\theta_k)\right)} = w_i^{(k)} \cdot e^{-y_i \beta_k G(x_i;\theta_k)}$$

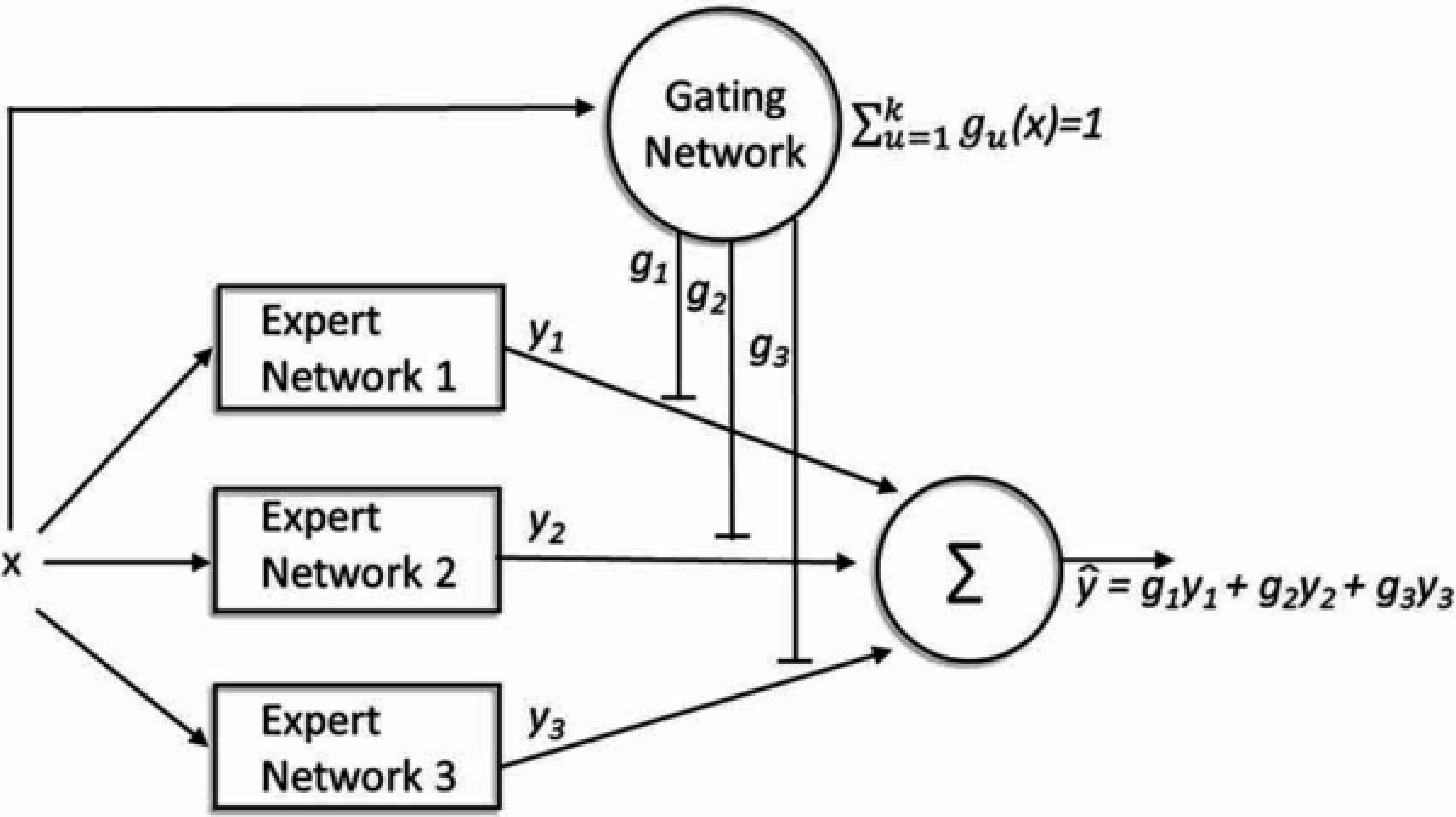
- Finally $-y_i G(x_i;\theta_k) = 2 \cdot [\![y_i \neq G(x_i;\theta_k)]\!] - 1$ gives the weight update:

$$w_i^{(k+1)} = w_i^{(k)} \cdot e^{\alpha_k [\![y_i \neq G(x_i;\theta_k)]\!]} \cdot e^{-\beta_k}$$

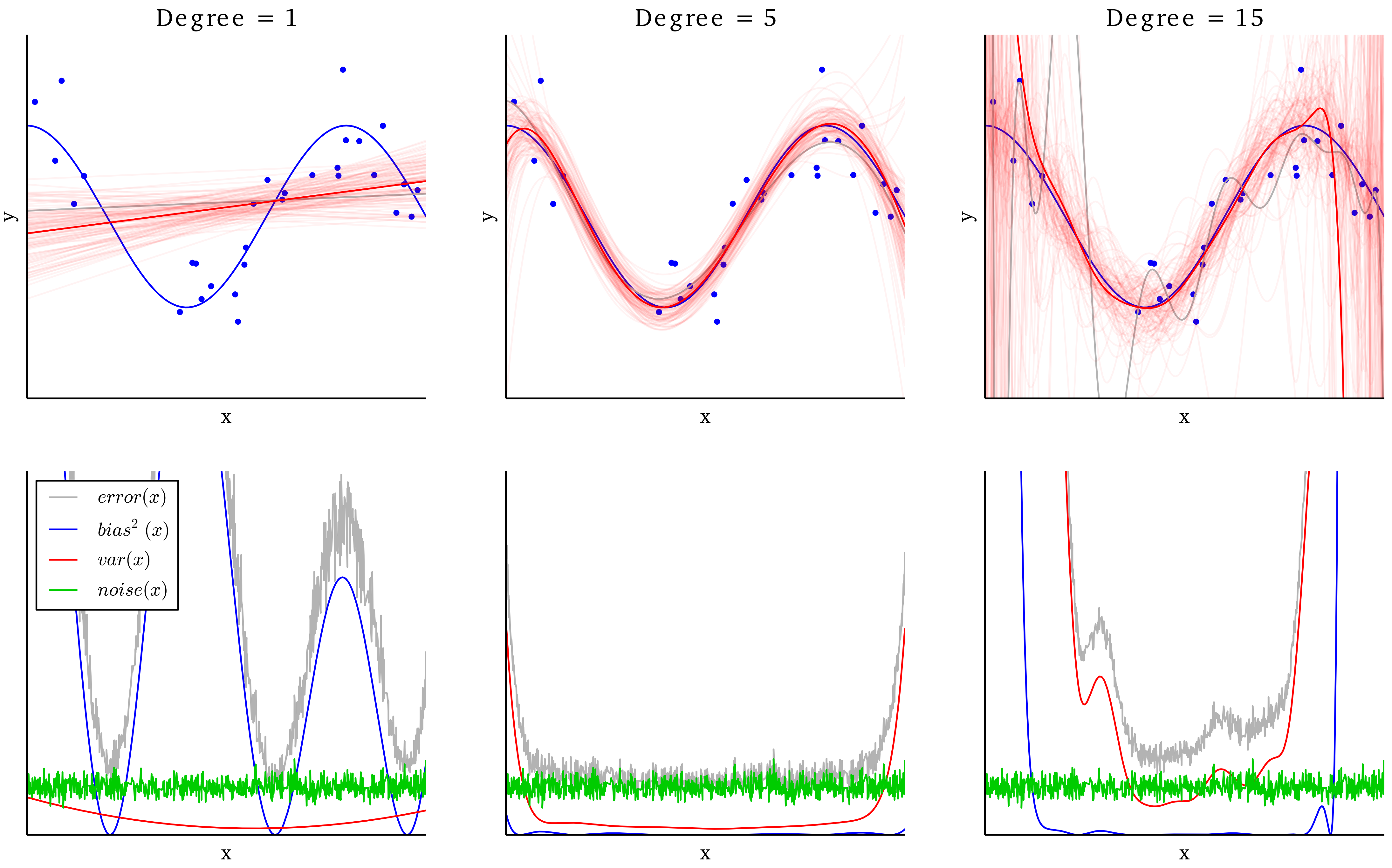
corresponding to *AdaBoost 2(d)* up to the factor $e^{-\beta_k}$ which is same for all weights and hence has no effect


3
3
5
3
5

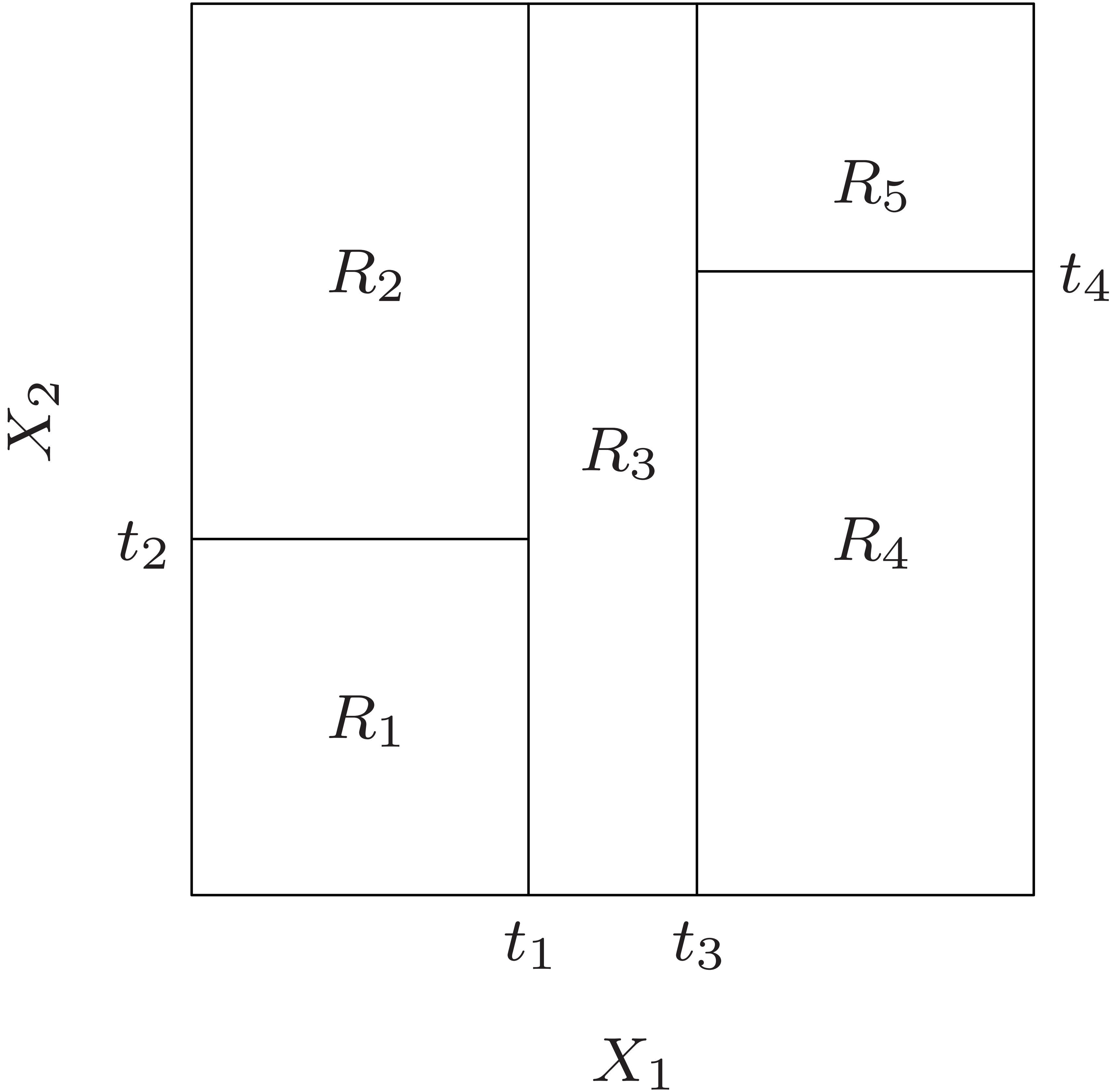

Training data
Model A
Training data
Model B
Predictions
Generalizer
Predictions
Training data
Model C
Level 0
Level 1



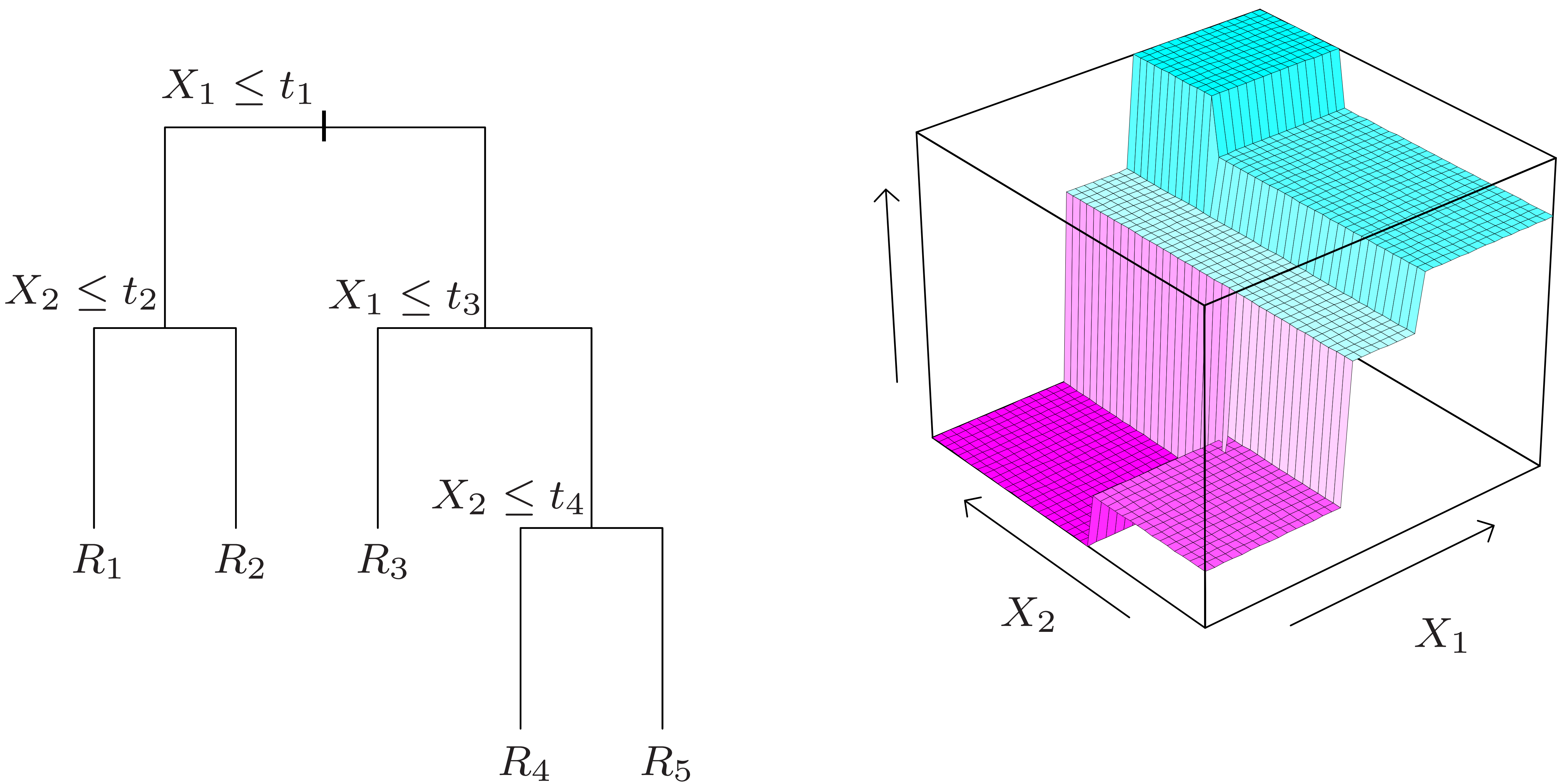
Gating Network
$\sum_{u=1}^{k} g_u(x)=1$
g_1
g_2
g_3
Expert Network 1
Expert Network 2
Expert Network 3
y_1
y_2
y_3
x
Σ
$\hat{y} = g_1y_1 + g_2y_2 + g_3y_3$

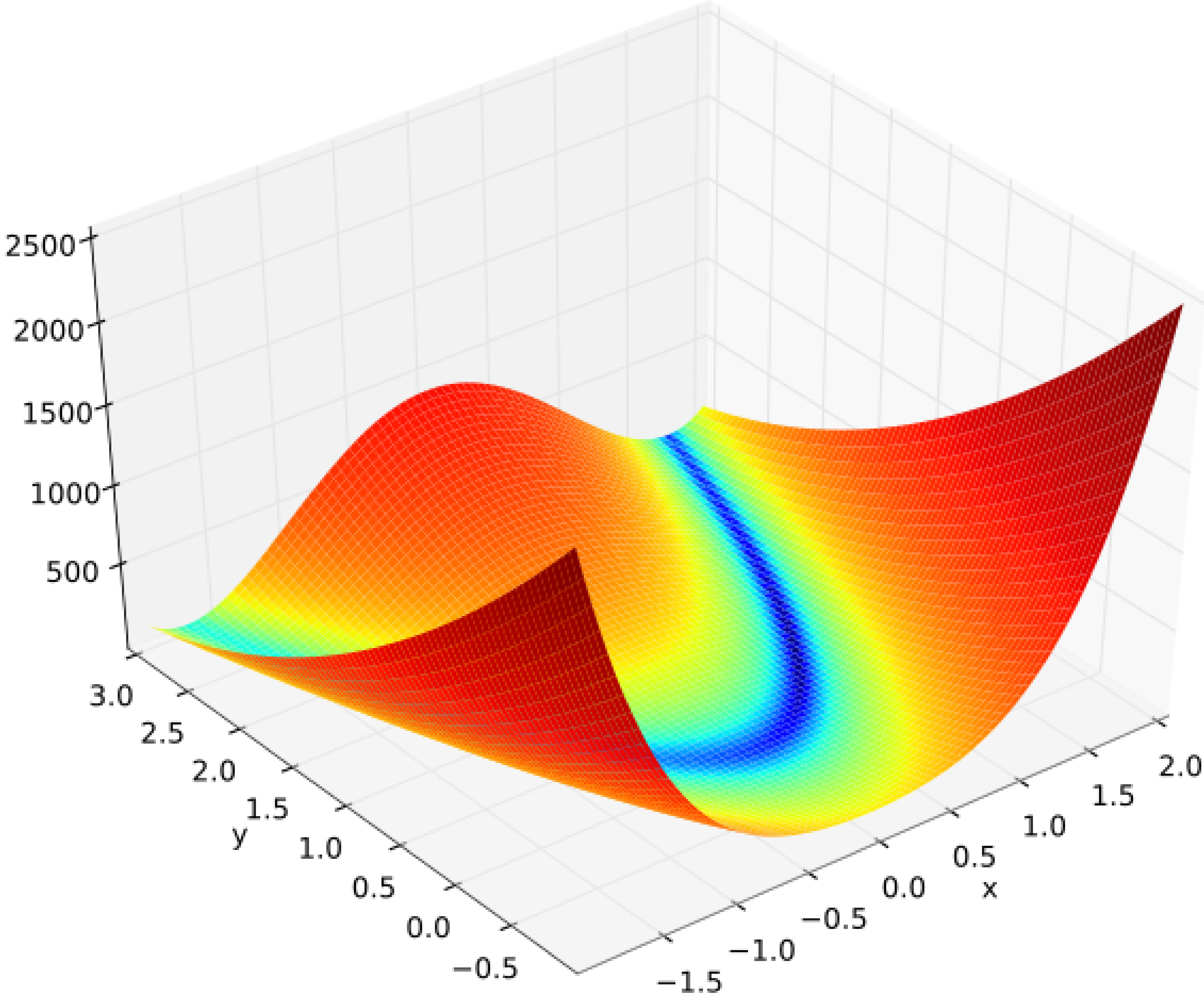


Degree = 1
Degree = 5
Degree = 15
y
x
error(x)
bias² (x)
var(x)
noise(x)

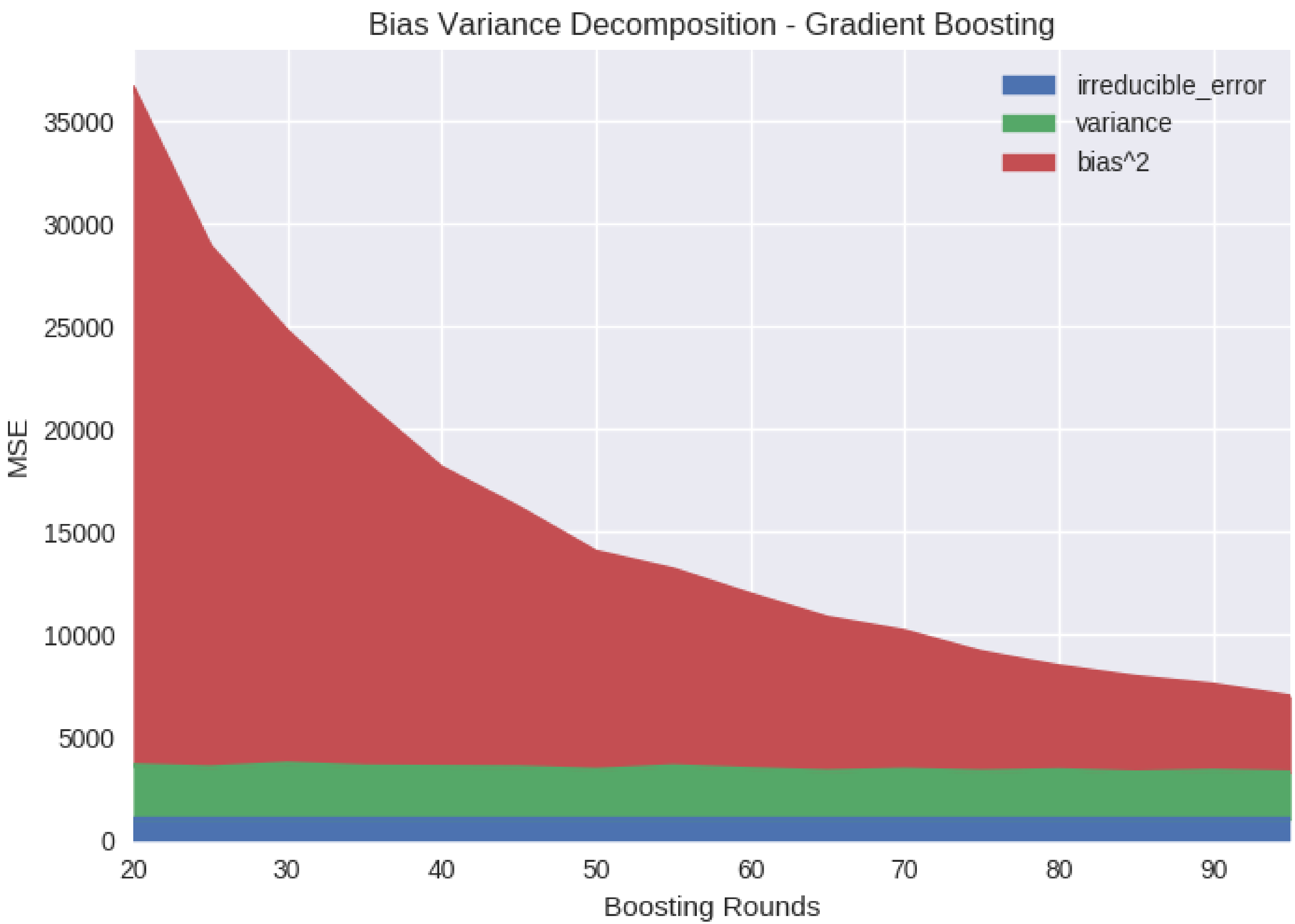


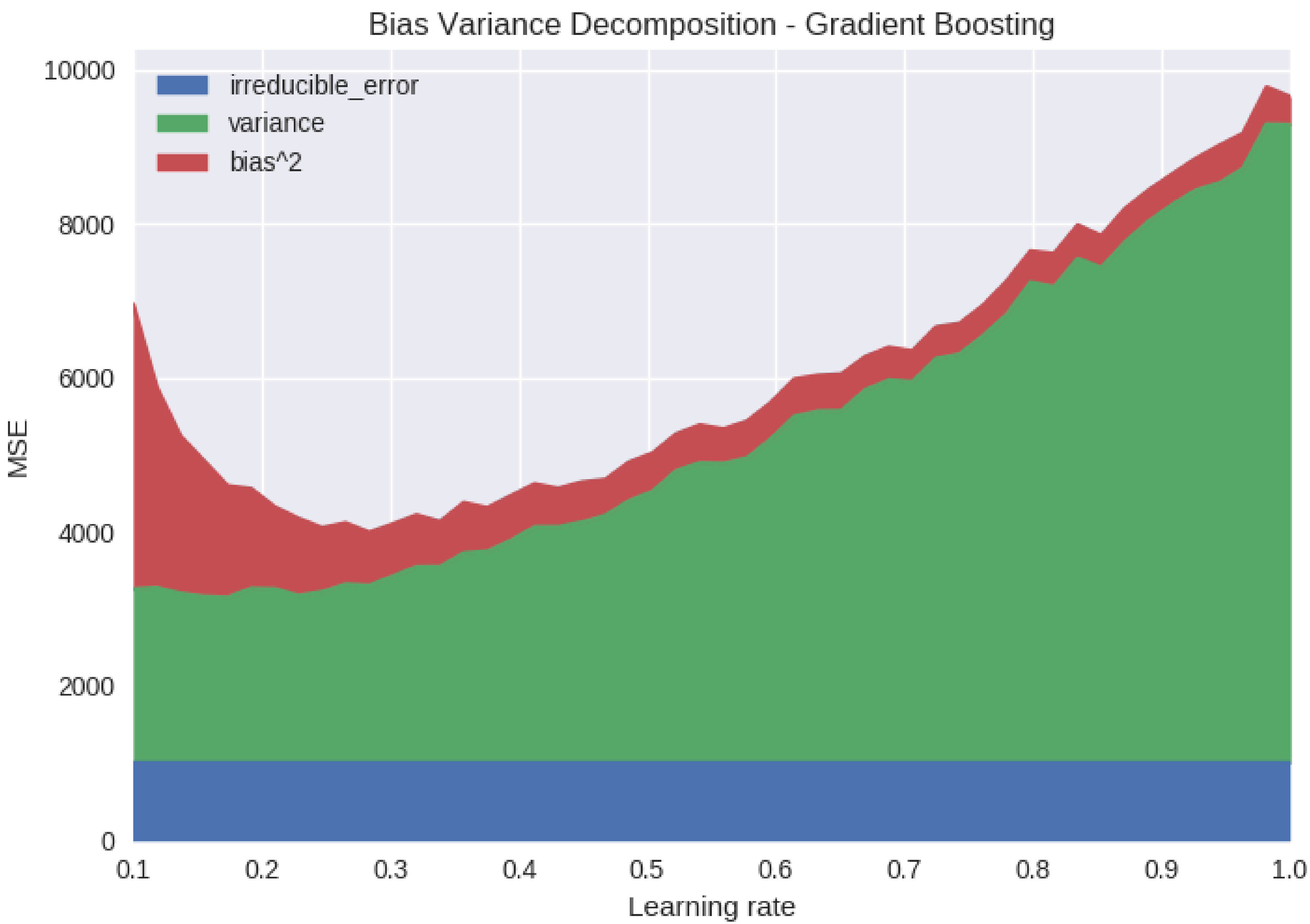
R_1
R_2
R_3
R_4
R_5
t_1
t_2
t_3
t_4
X_1
X_2


$X_1 \le t_1$
$X_2 \le t_2$
$X_1 \le t_3$
$X_2 \le t_4$
R_1
R_2
R_3
R_4
R_5
X_2
X_1

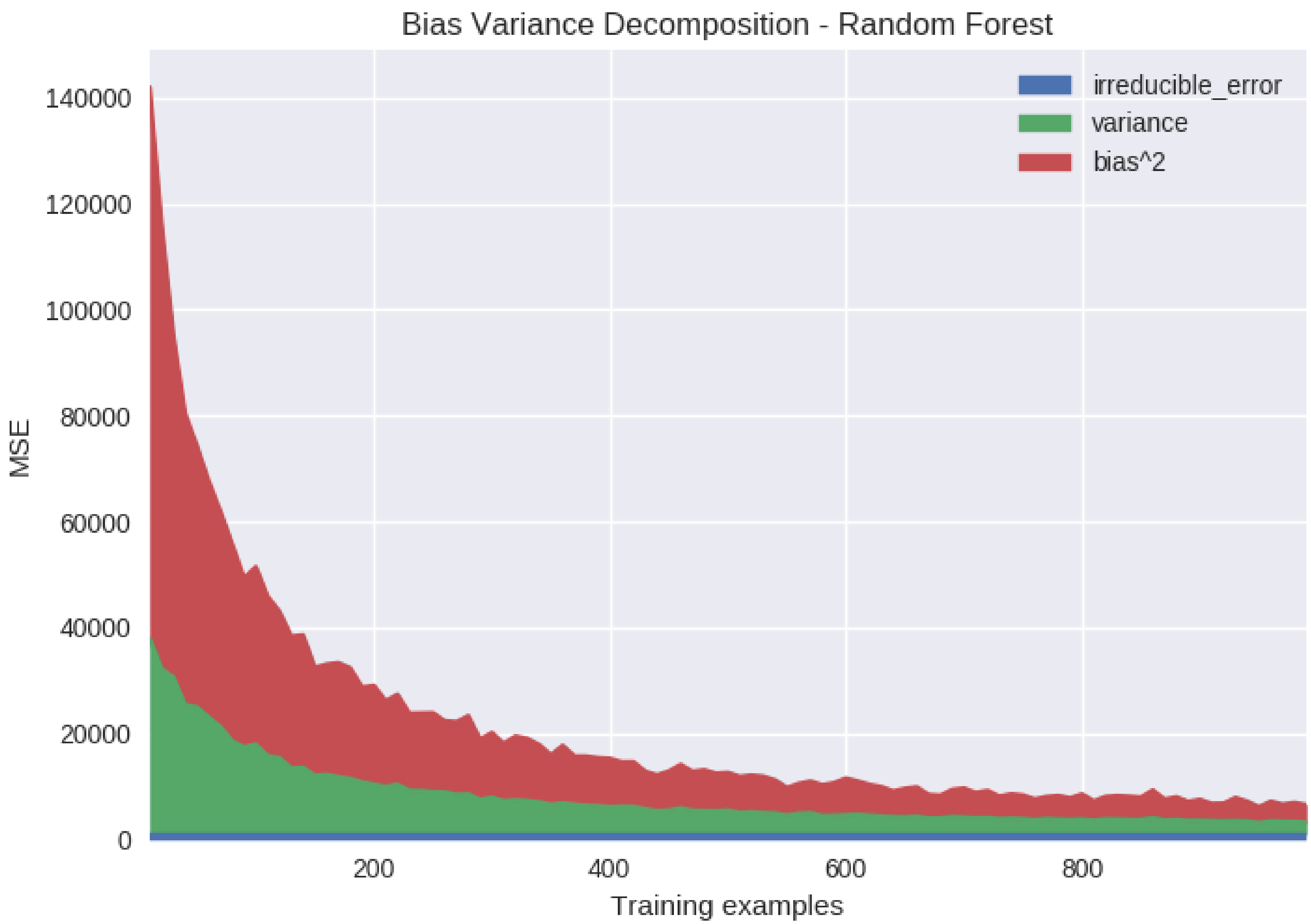


2500
2000
1500
1000
500
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
y
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x

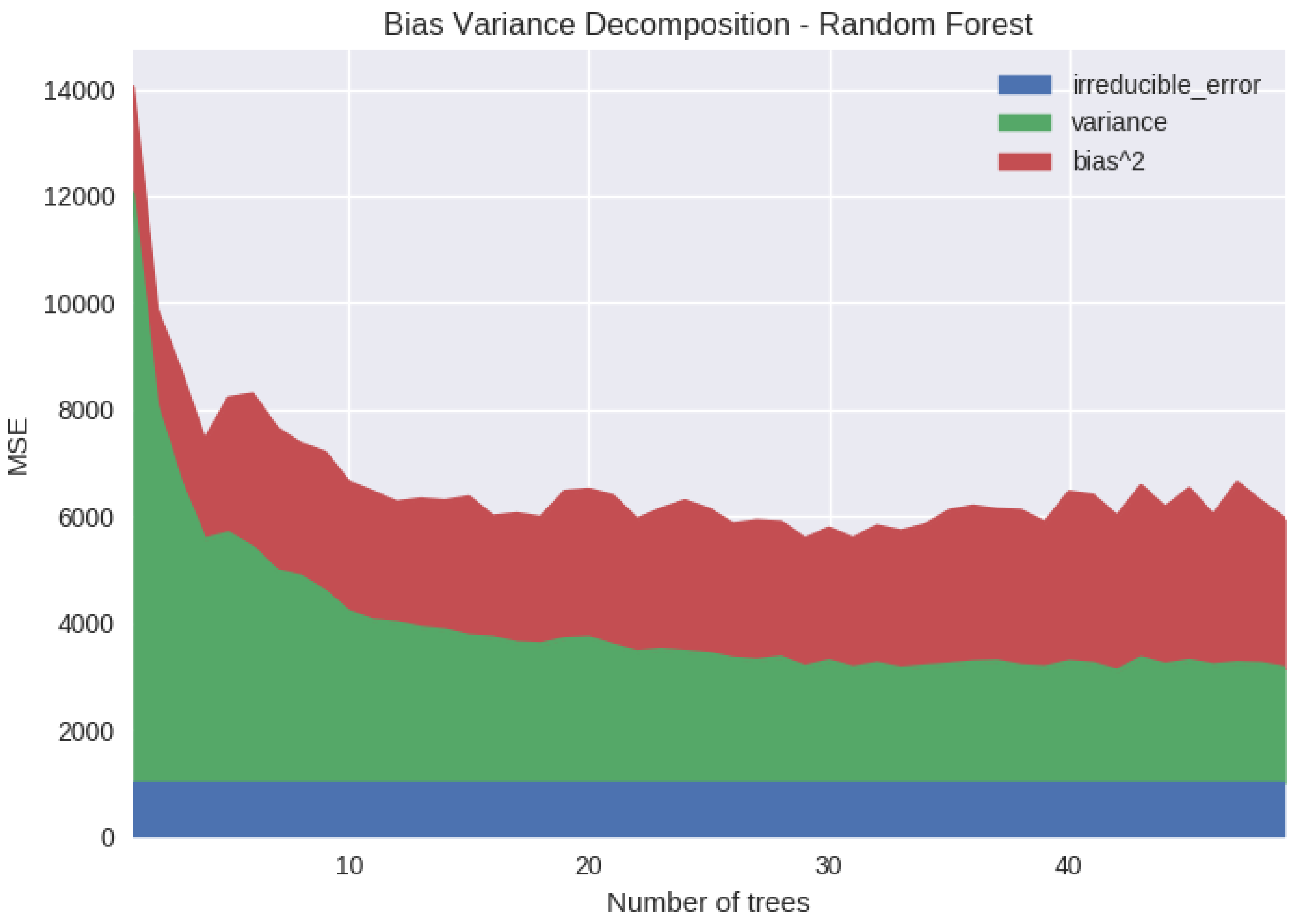

Bias Variance Decomposition - Gradient Boosting
irreducible_error
variance
bias^2
MSE
35000
30000
25000
20000
15000
10000
5000
0
20
30
40
50
60
70
80
90
Boosting Rounds

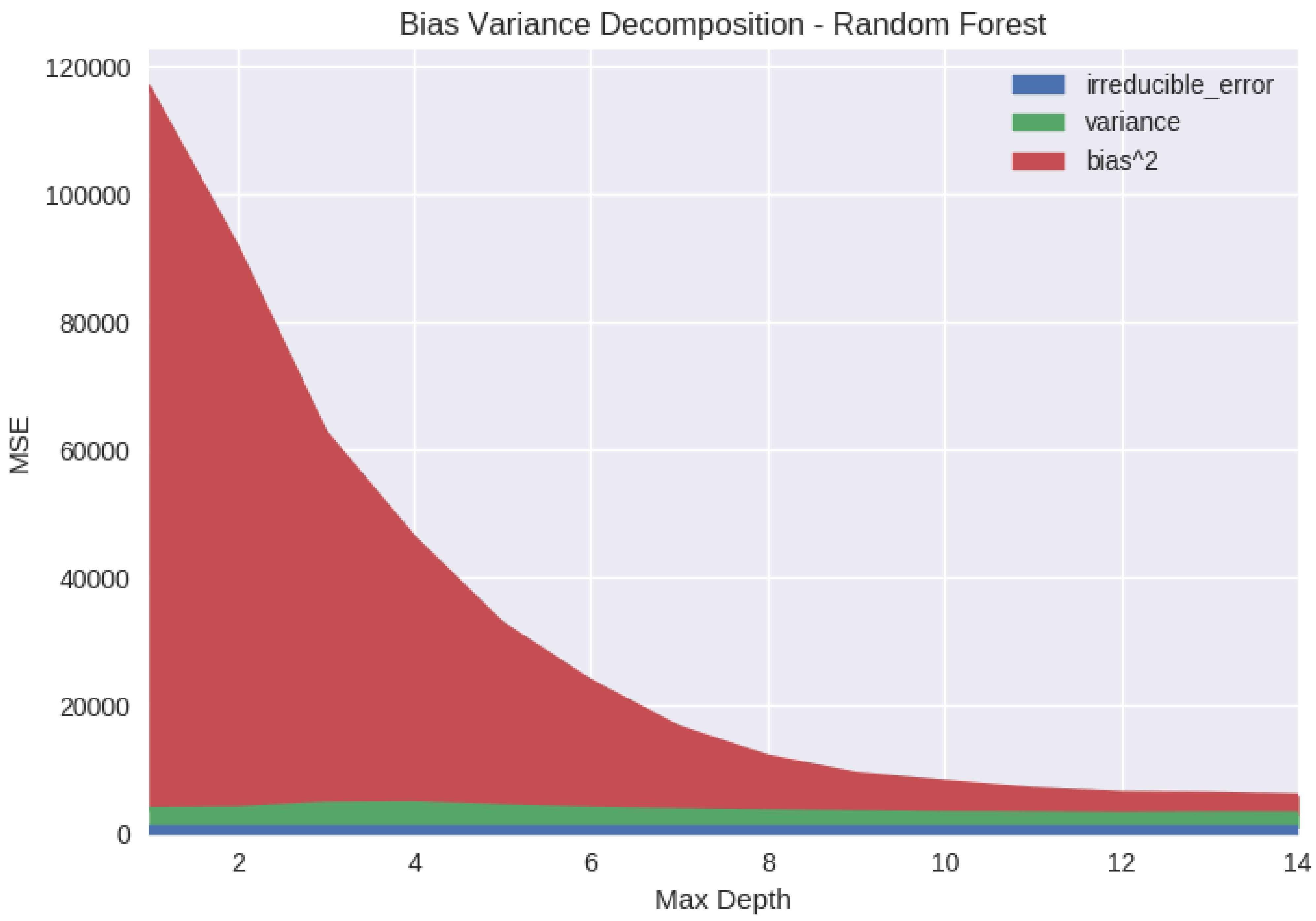


Bias Variance Decomposition - Gradient Boosting
irreducible_error
variance
bias^2
MSE
0
2000
4000
6000
8000
10000
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Learning rate

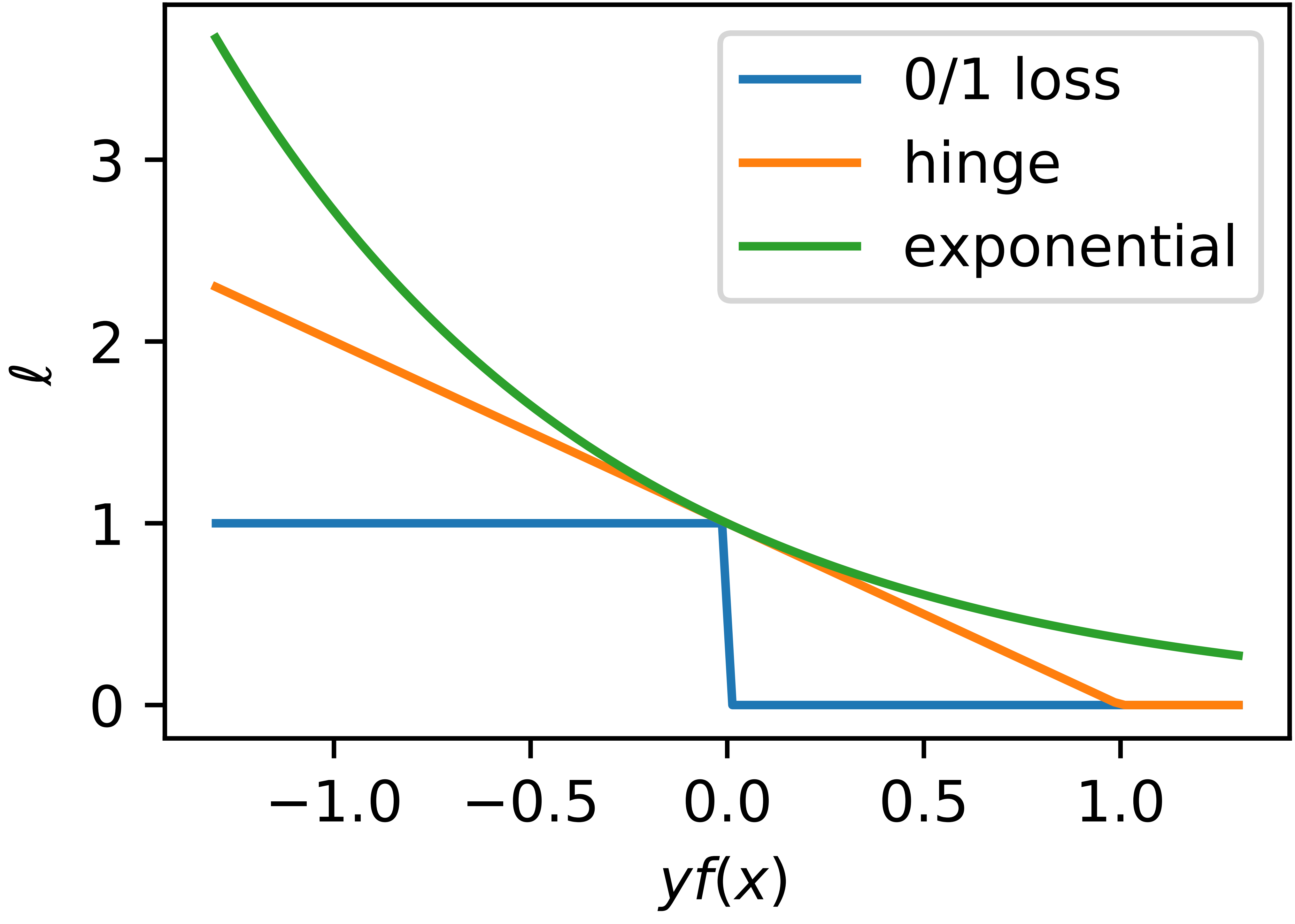


Bias Variance Decomposition - Random Forest
irreducible_error
variance
bias^2
MSE
140000
120000
100000
80000
60000
40000
20000
0
200
400
600
800
Training examples


Bias Variance Decomposition - Random Forest
irreducible_error
variance
bias^2
MSE
14000
12000
10000
8000
6000
4000
2000
0
10
20
30
40
Number of trees



Bias Variance Decomposition - Random Forest
irreducible_error
variance
bias^2
MSE
120000
100000
80000
60000
40000
20000
0
2
4
6
8
10
12
14
Max Depth


0/1 loss
hinge
exponential
3
2
1
0
ℓ
−1.0
−0.5
0.0
0.5
1.0
yf(x)