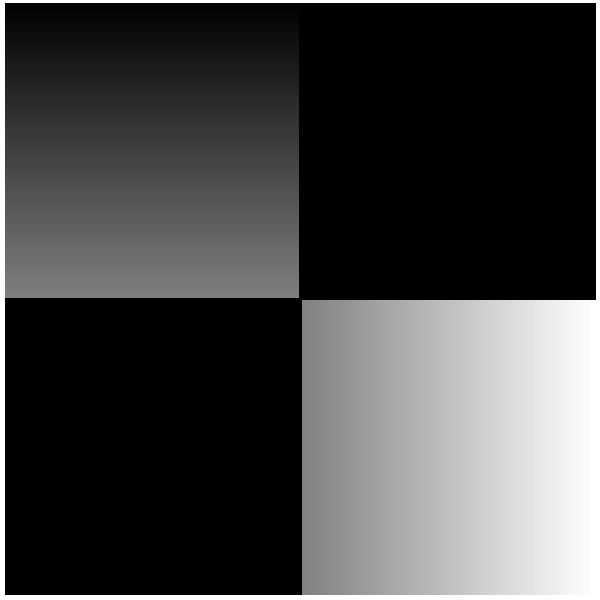
Chapter 15 & 16: Random Forests & Ensemble Learning

DD3364

November 27, 2012

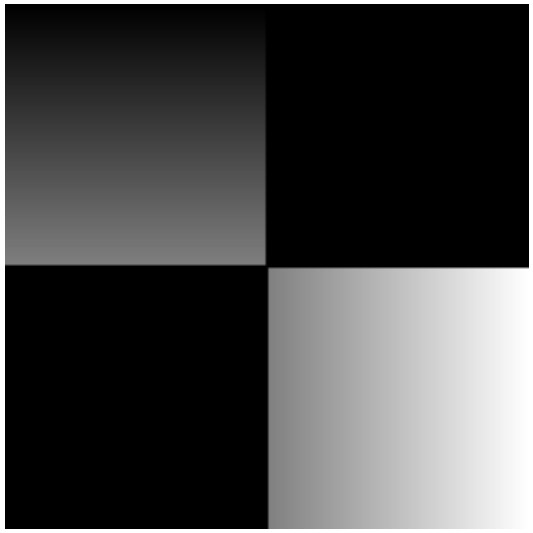
Toy Problem for Boosted Tree

Boosted Tree Example

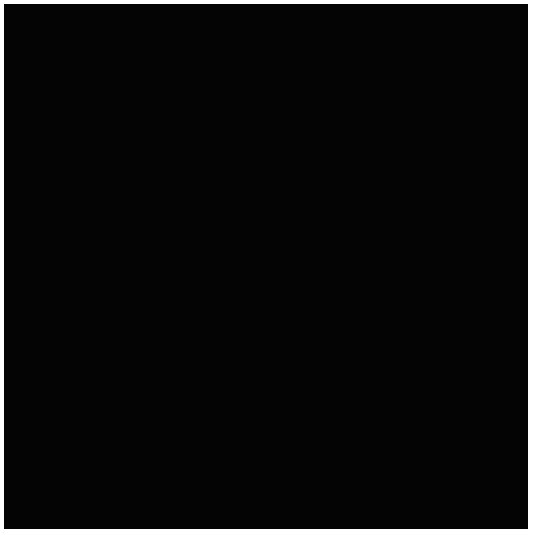


Estimate this function with a sum of trees with 9-terminal nodes by minimizing the sum of the **absolute loss** on $n = 900$ training points.

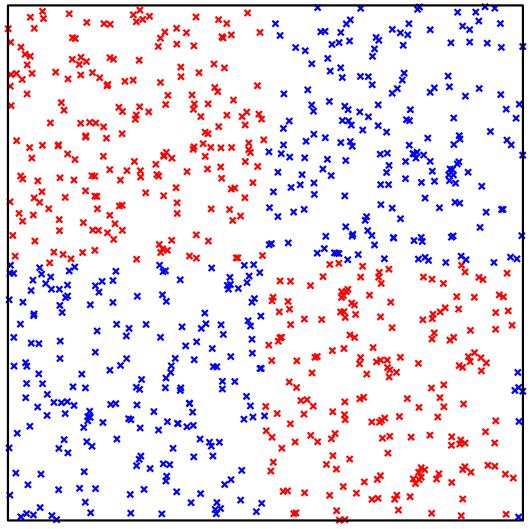
Boosted Tree learning via GBM: $m = 1$



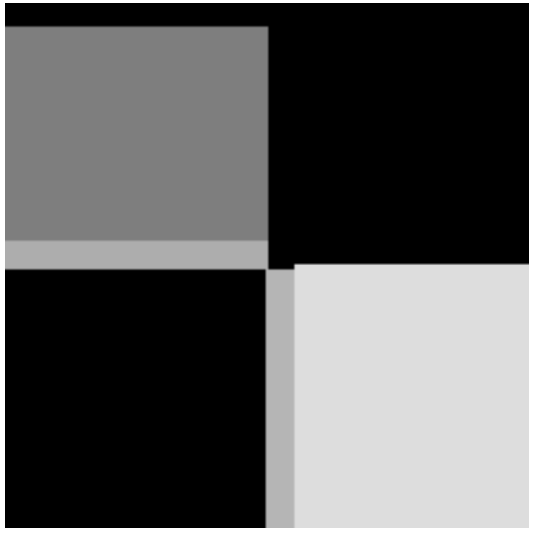
true $f(x)$



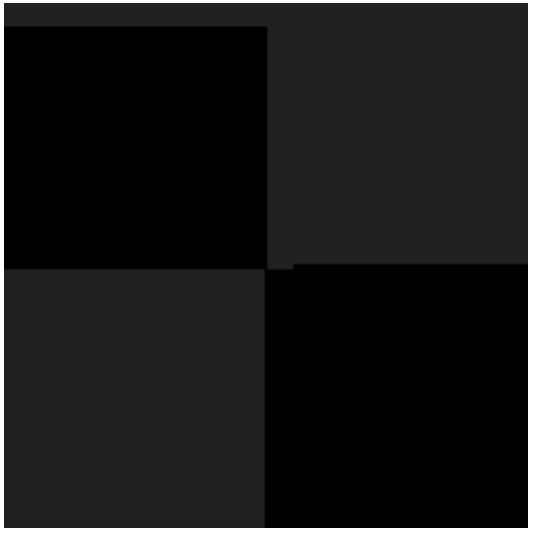
current estimate $f_m(x)$



$r_{im} = \text{sign}(y_i - f_m(x_i))$



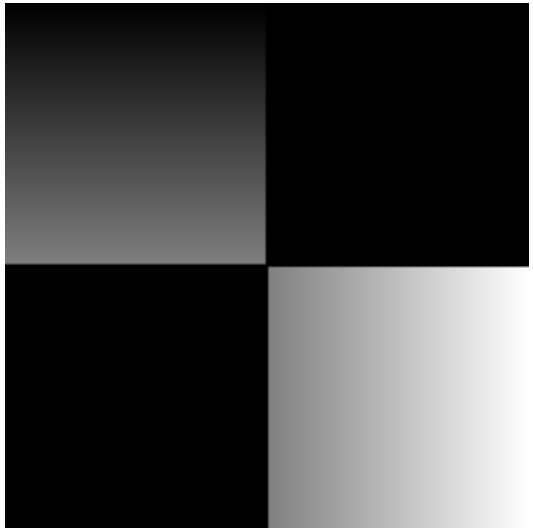
tree added



tree subtracted

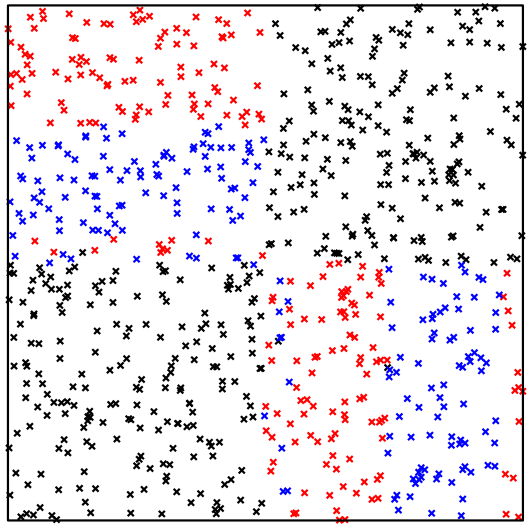
$f_m(x) + T_m(x)$

Boosted Tree learning via GBM: $m = 2$



true $f(x)$

current estimate $f_m(x)$



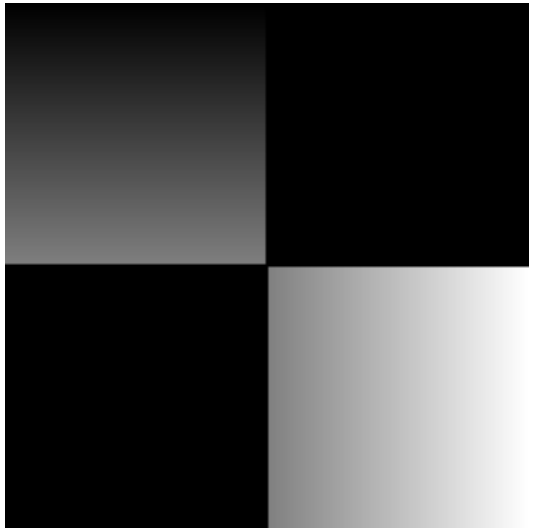
$r_{im} = \text{sign}(y_i - f_m(x_i))$

tree added

tree subtracted

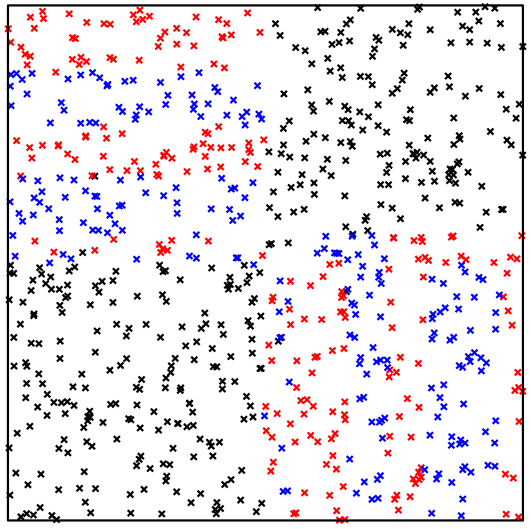
$f_m(x) + T_m(x)$

Boosted Tree learning via GBM: $m = 3$



true $f(x)$

current estimate $f_m(x)$



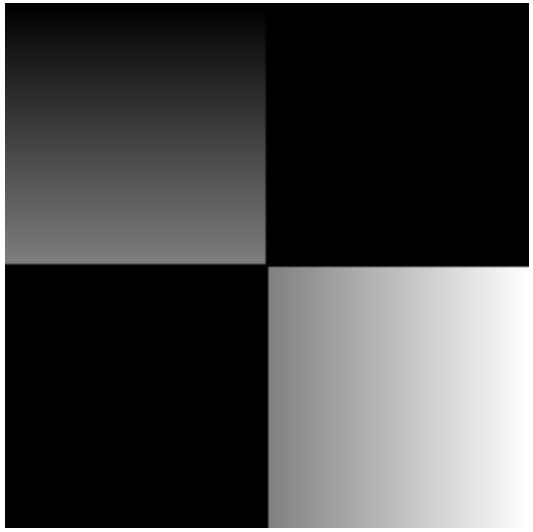
$r_{im} = \text{sign}(y_i - f_m(x_i))$

tree added

tree subtracted

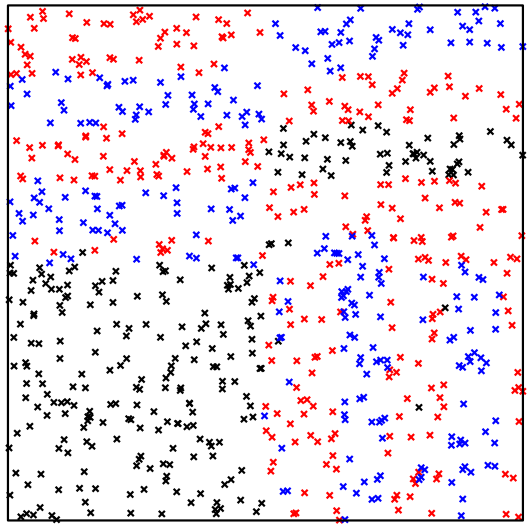
$f_m(x) + T_m(x)$

Boosted Tree learning via GBM: $m = 4$



true $f(x)$

current estimate $f_m(x)$



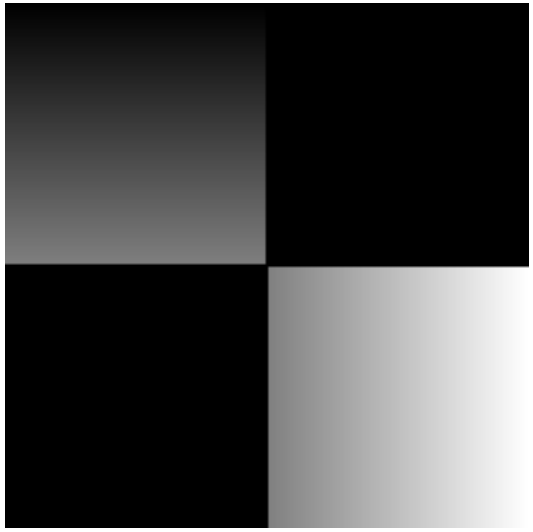
$r_{im} = \text{sign}(y_i - f_m(x_i))$

tree added

tree subtracted

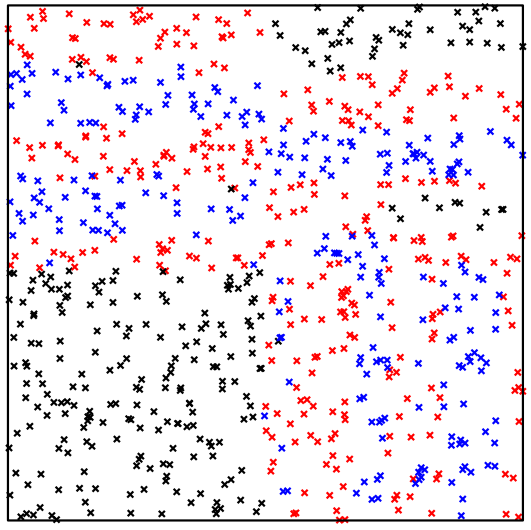
$f_m(x) + T_m(x)$

Boosted Tree learning via GBM: $m = 5$



true $f(x)$

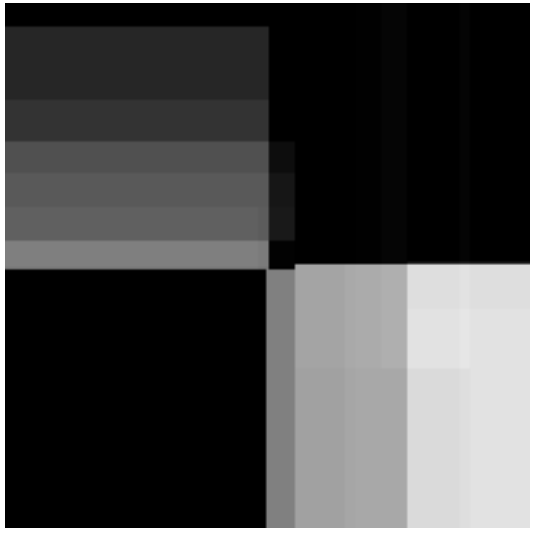
current estimate $f_m(x)$



$r_{im} = \text{sign}(y_i - f_m(x_i))$

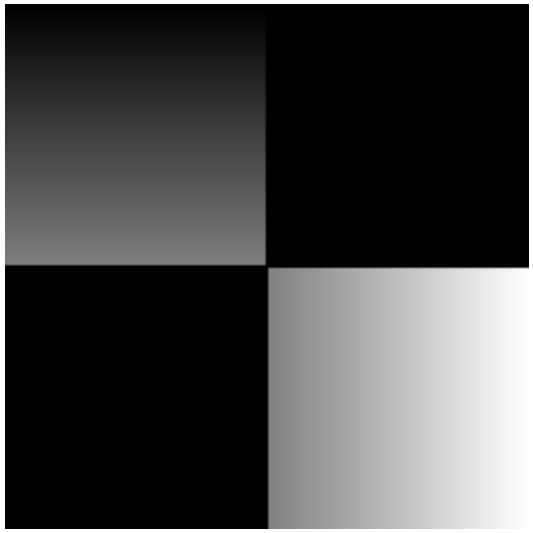
tree added

tree subtracted

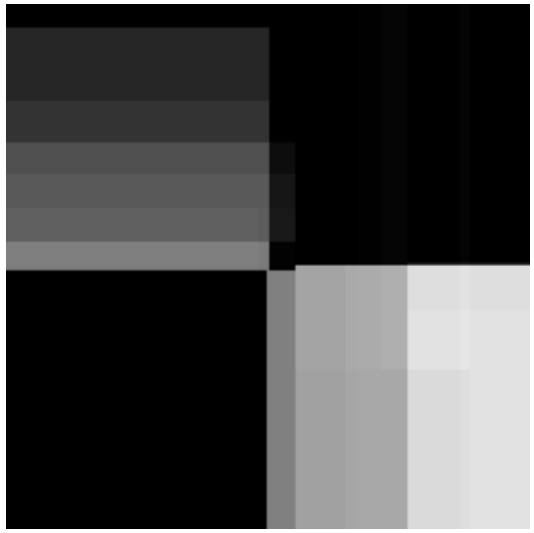


$f_m(x) + T_m(x)$

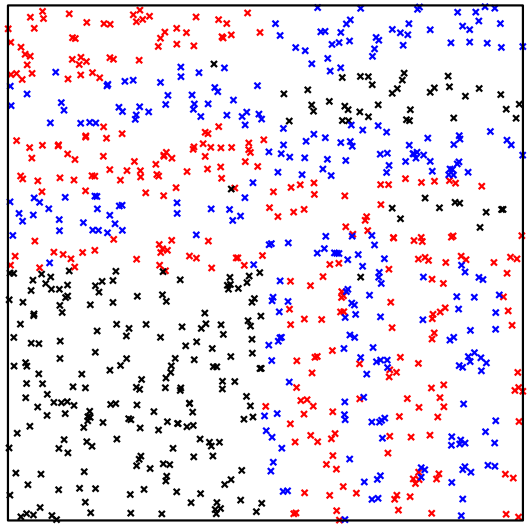
Boosted Tree learning via GBM: $m = 6$



true $f(x)$



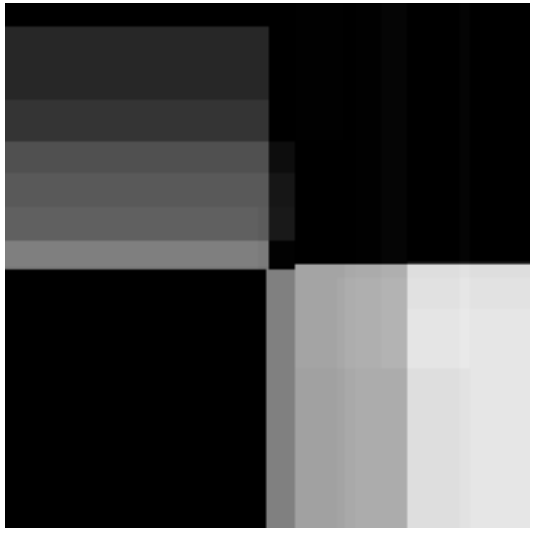
current estimate $f_m(x)$



$r_{im} = \text{sign}(y_i - f_m(x_i))$

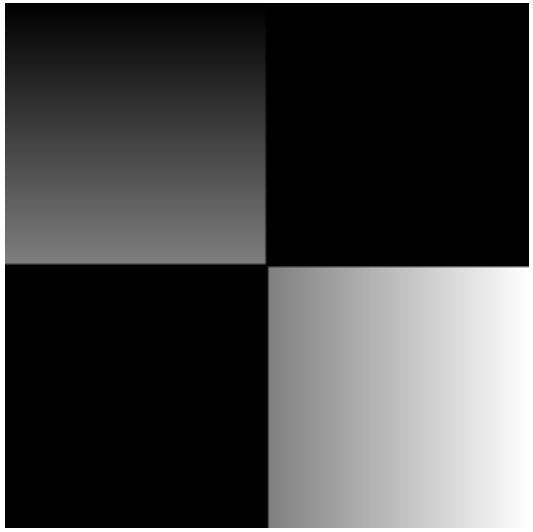
tree added

tree subtracted

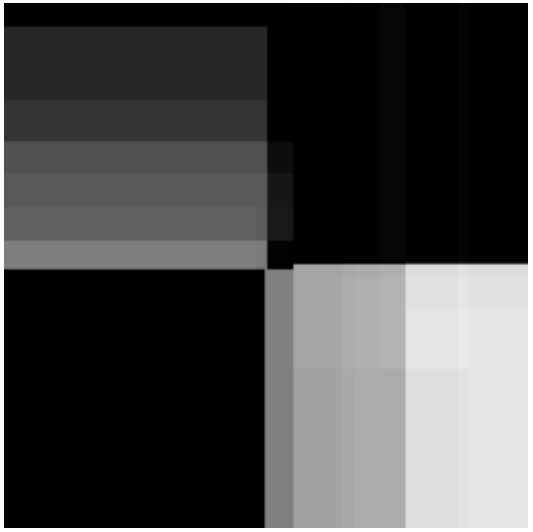


$f_m(x) + T_m(x)$

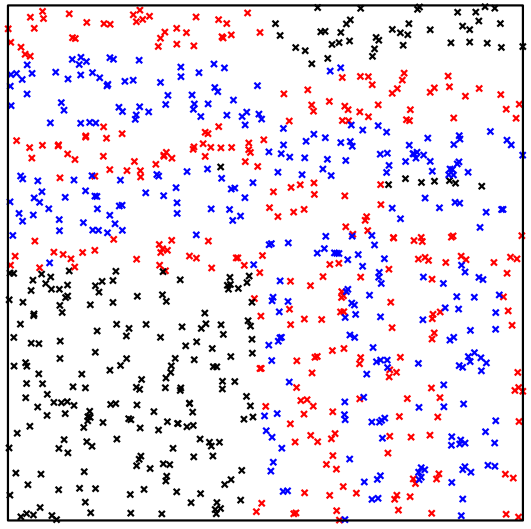
Boosted Tree learning via GBM: $m = 7$



true $f(x)$



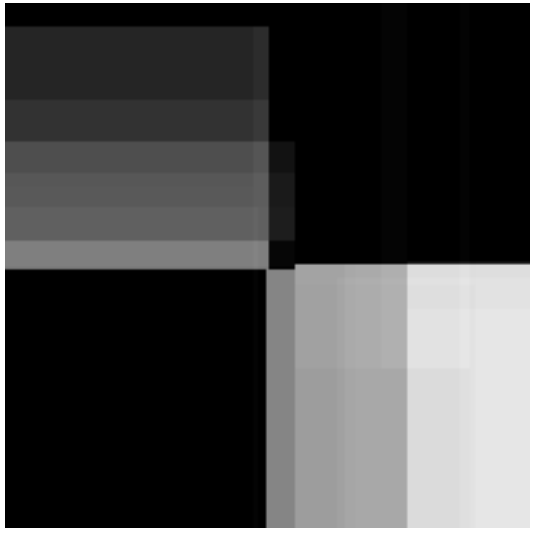
current estimate $f_m(x)$



$r_{im} = \mathsf{sign}(y_i - f_m(x_i))$

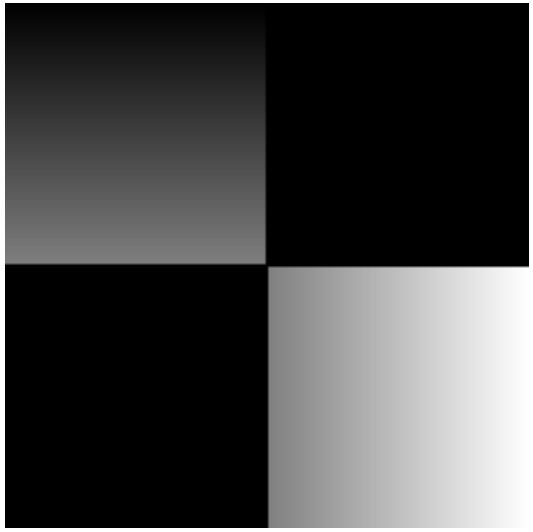
tree added

tree subtracted

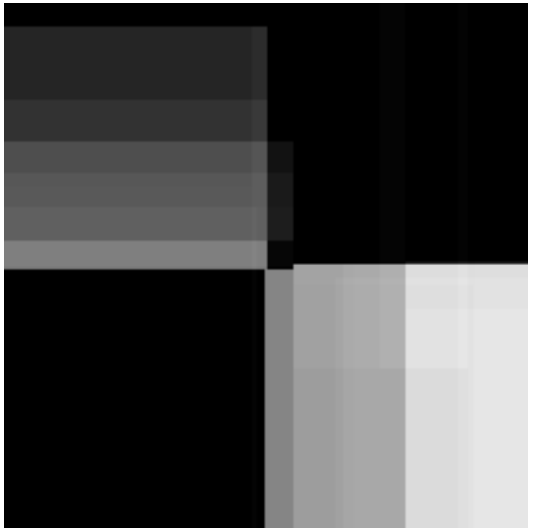


$f_m(x) + T_m(x)$

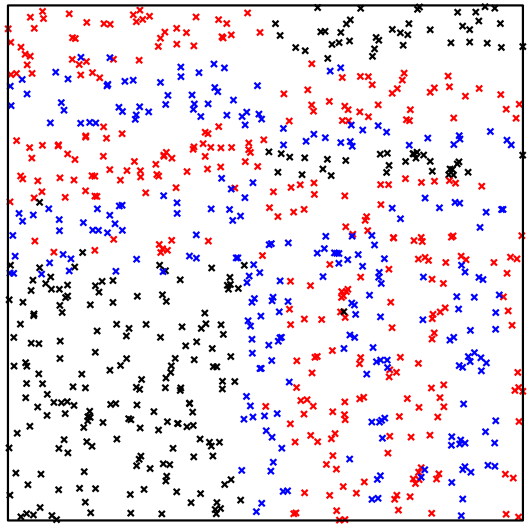
Boosted Tree learning via GBM: $m = 8$



true $f(x)$



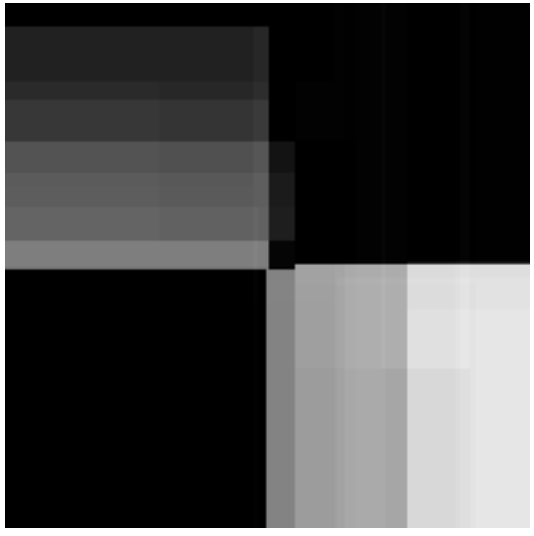
current estimate $f_m(x)$



$r_{im} = \mathsf{sign}(y_i - f_m(x_i))$

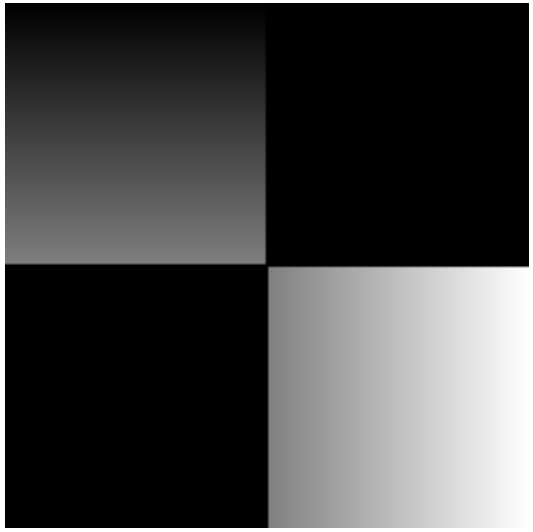
tree added

tree subtracted

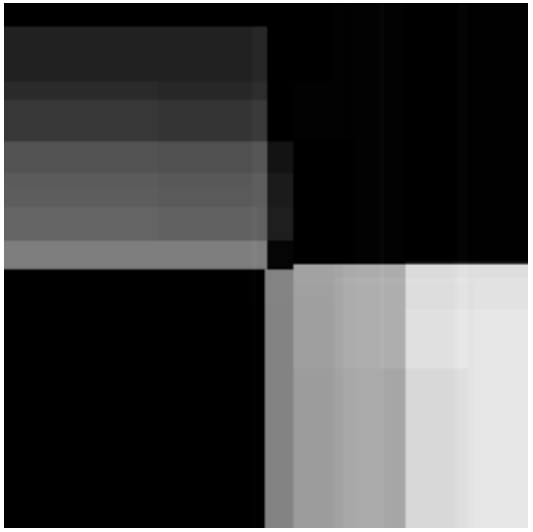


$f_m(x) + T_m(x)$

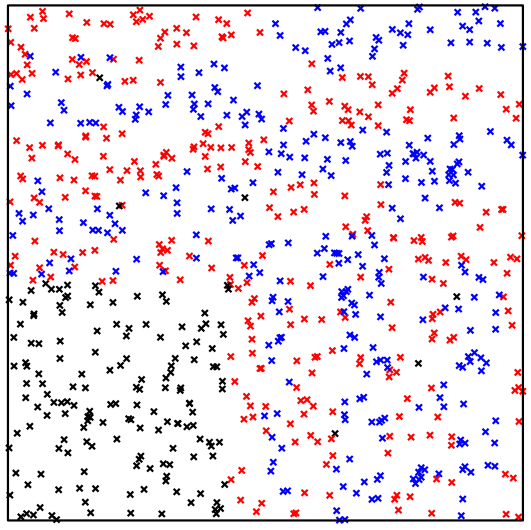
Boosted Tree learning via GBM: $m = 9$



true $f(x)$



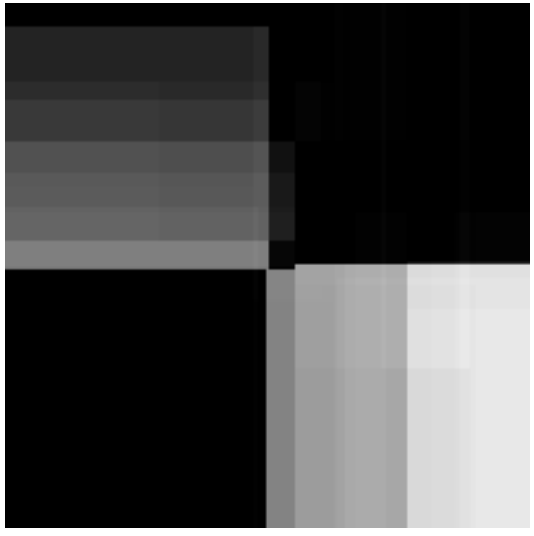
current estimate $f_m(x)$



$r_{im} = \text{sign}(y_i - f_m(x_i))$

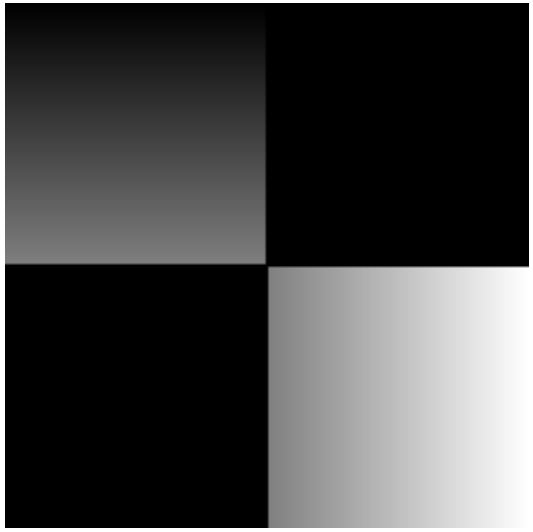
tree added

tree subtracted

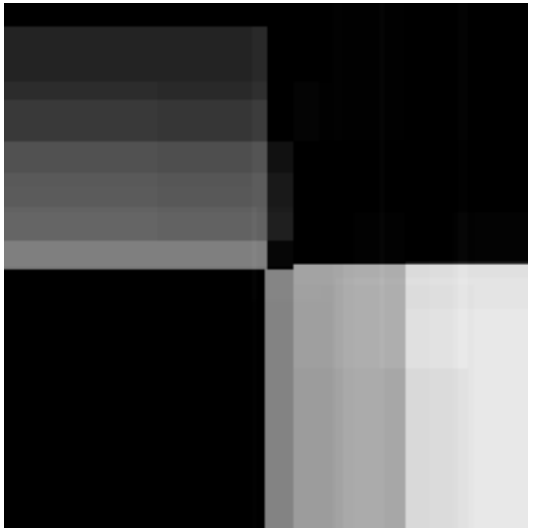


$f_m(x) + T_m(x)$

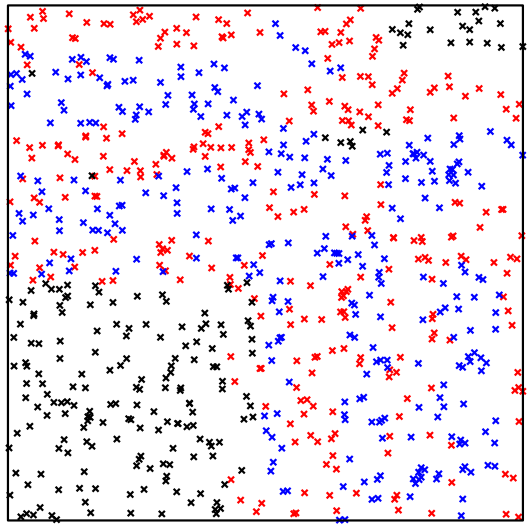
Boosted Tree learning via GBM: $m = 10$



true $f(x)$



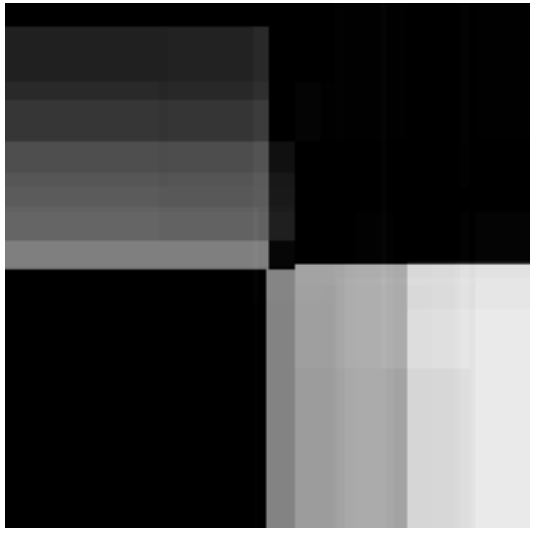
current estimate $f_m(x)$



$r_{im} = \text{sign}(y_i - f_m(x_i))$

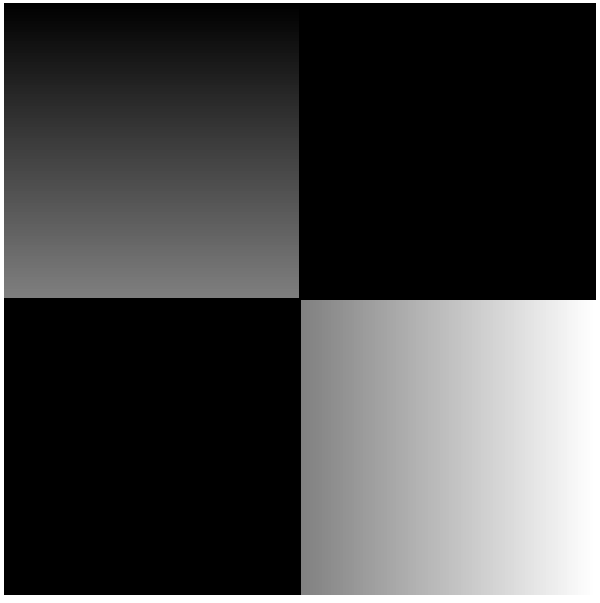
tree added

tree subtracted

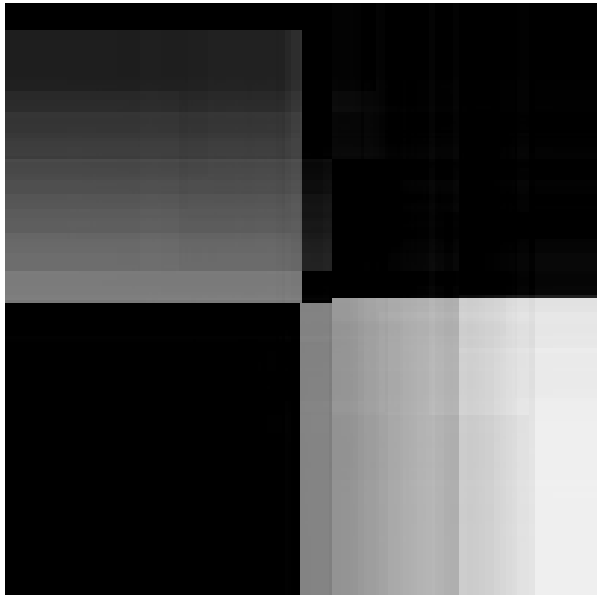


$f_m(x) + T_m(x)$

After 200 iterations

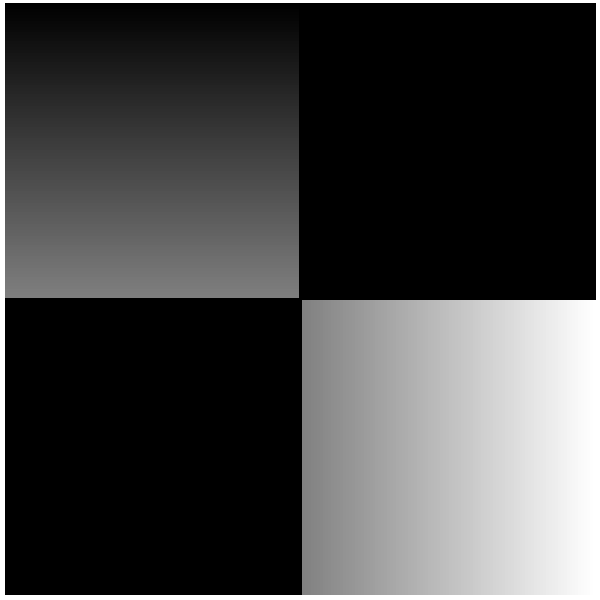


true $f(x)$



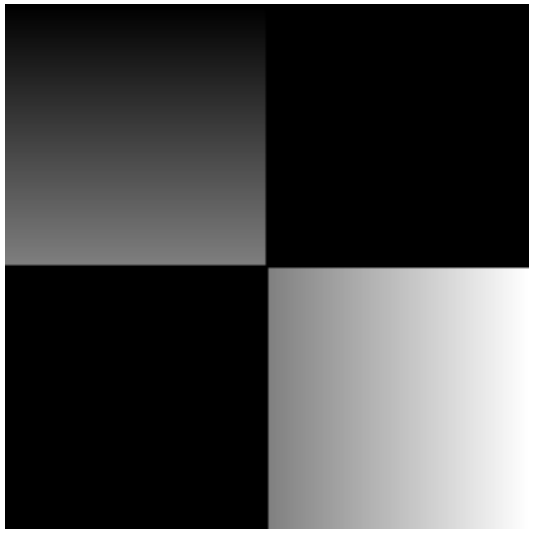
$f_{200}(x)$

Boosted Tree Example



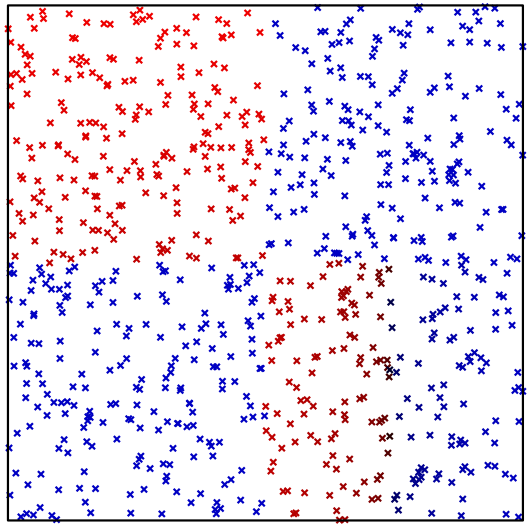
Estimate this function with a sum of trees with 9-terminal nodes by minimizing the sum of the L_2 **loss** on $n = 900$ training points.

Boosted Tree learning via GBM: $m = 1$



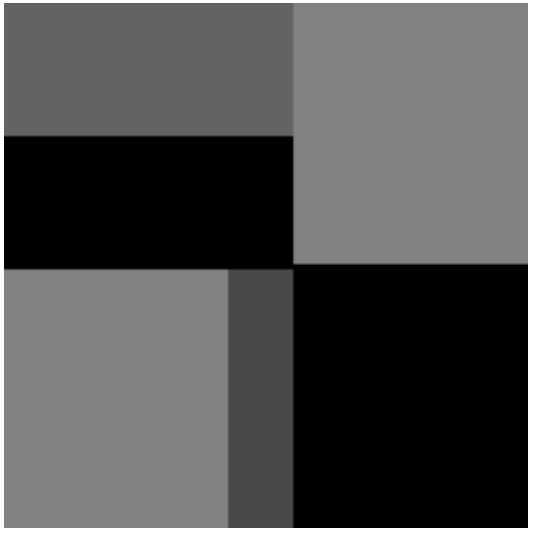
true $f(x)$

current estimate $f_m(x)$



$r_{im} = y_i - f_m(x_i)$

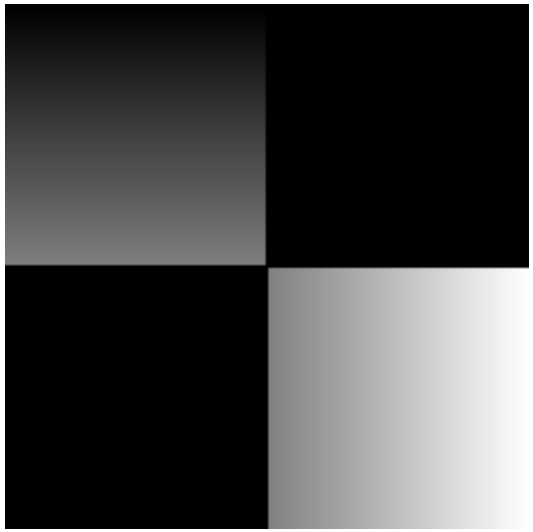
tree added



tree subtracted

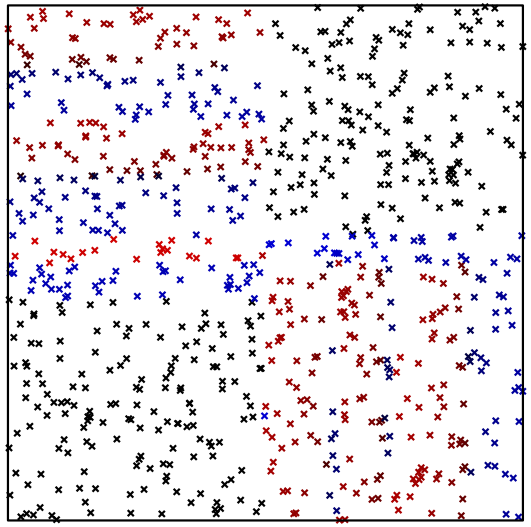
$f_m(x) + T_m(x)$

Boosted Tree learning via GBM: $m = 2$



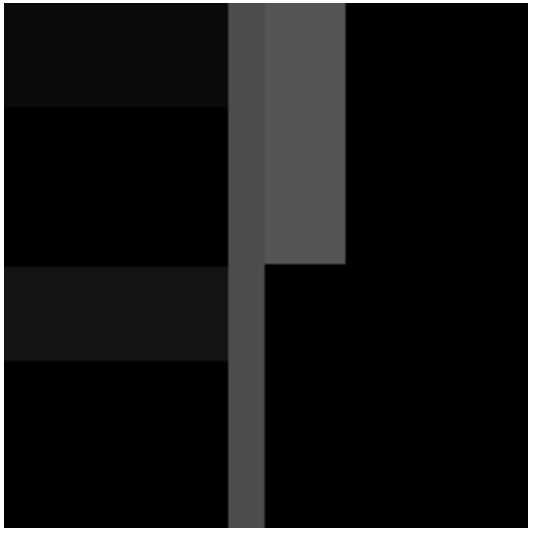
true $f(x)$

current estimate $f_m(x)$



$r_{im} = y_i - f_m(x_i)$

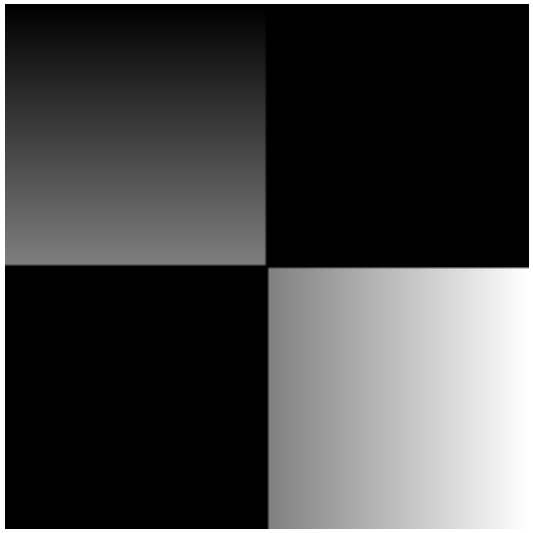
tree added



tree subtracted

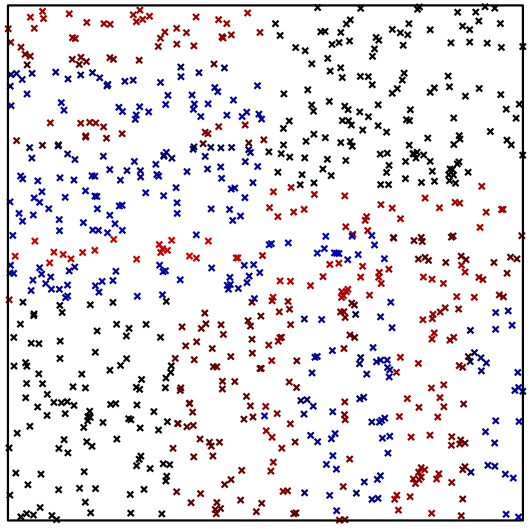
$f_m(x) + T_m(x)$

Boosted Tree learning via GBM: $m = 3$



true $f(x)$

current estimate $f_m(x)$



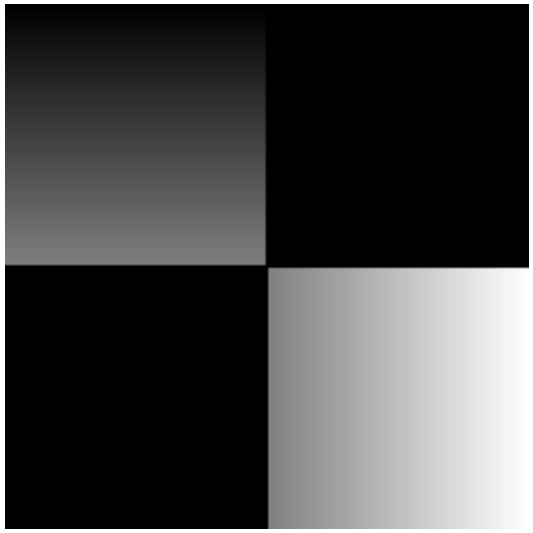
$r_{im} = y_i - f_m(x_i)$

tree added

tree subtracted

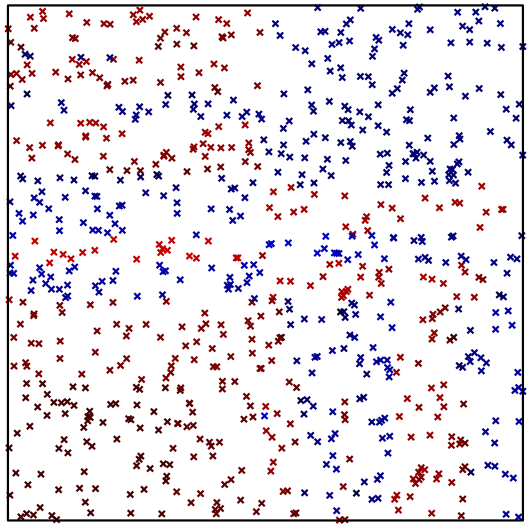
$f_m(x) + T_m(x)$

Boosted Tree learning via GBM: $m = 4$



true $f(x)$

current estimate $f_m(x)$



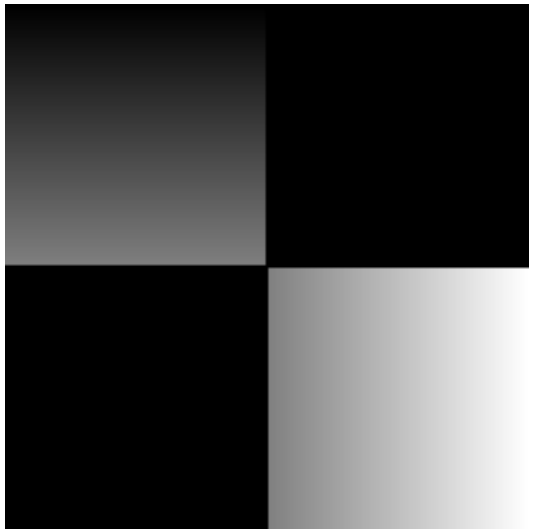
$r_{im} = y_i - f_m(x_i)$

tree added

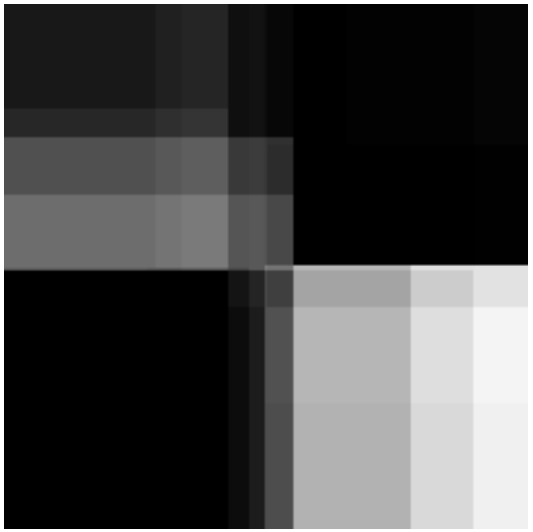
tree subtracted

$f_m(x) + T_m(x)$

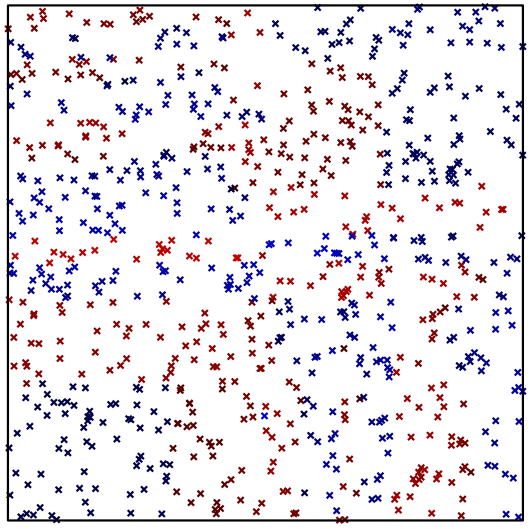
Boosted Tree learning via GBM: $m = 5$



true $f(x)$



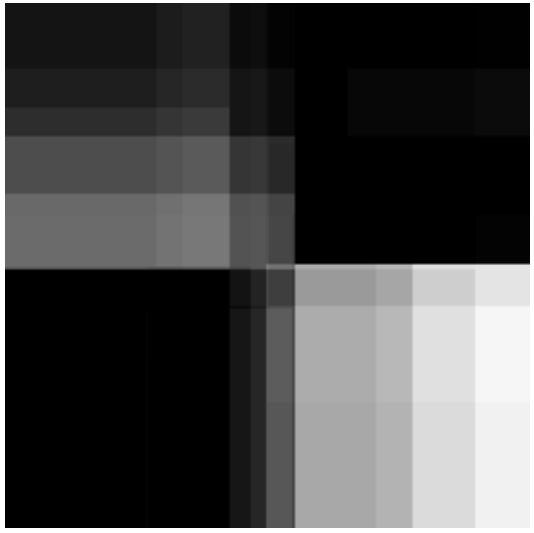
current estimate $f_m(x)$



$r_{im} = y_i - f_m(x_i)$

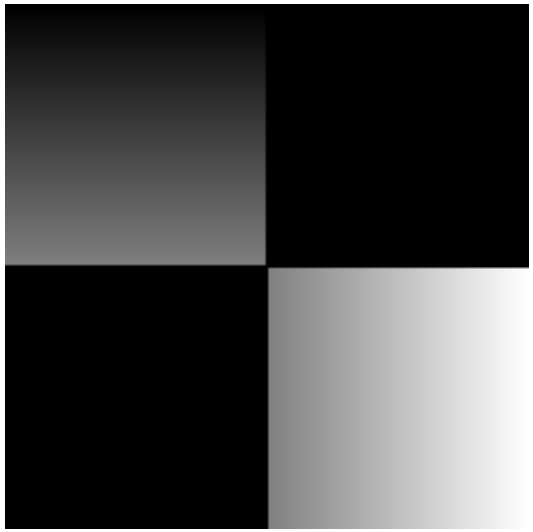
tree added

tree subtracted

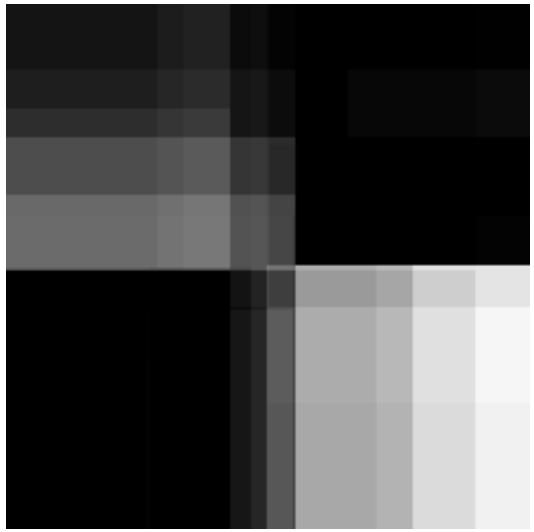


$f_m(x) + T_m(x)$

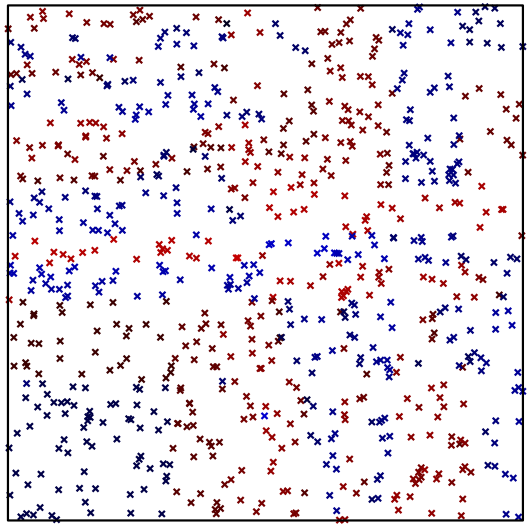
Boosted Tree learning via GBM: $m = 6$



true $f(x)$



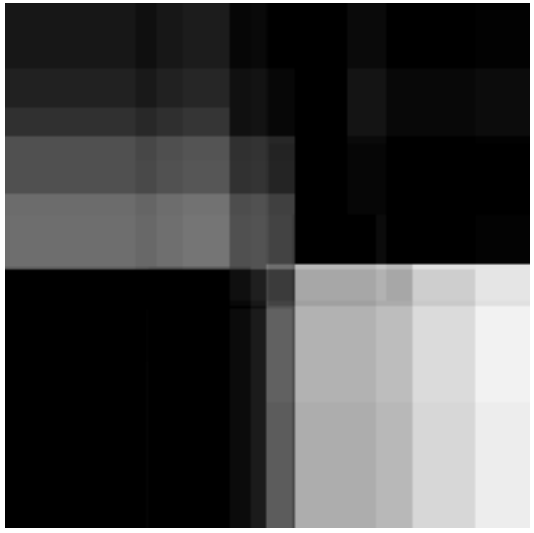
current estimate $f_m(x)$



$r_{im} = y_i - f_m(x_i)$

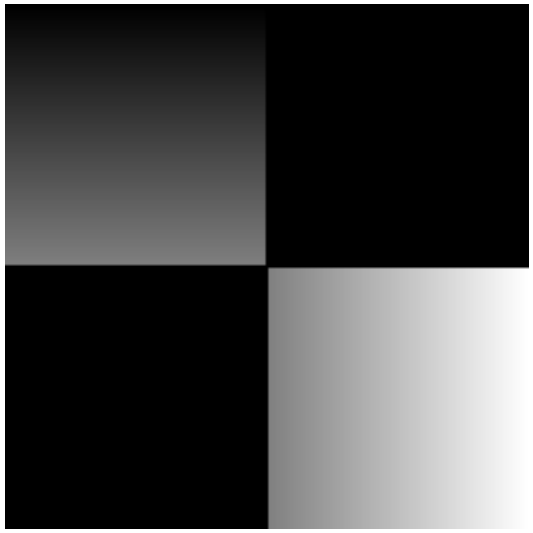
tree added

tree subtracted

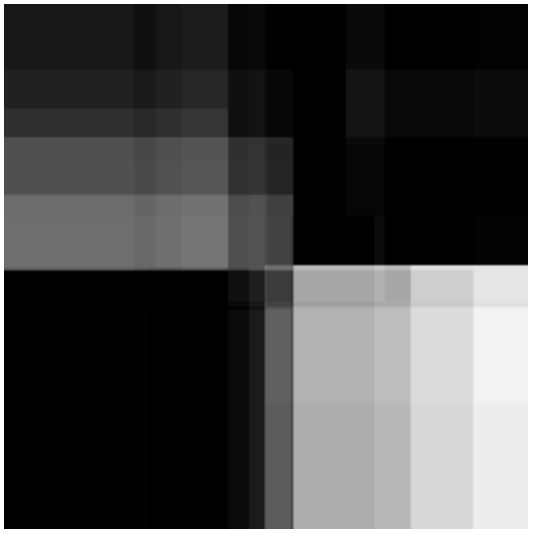


$f_m(x) + T_m(x)$

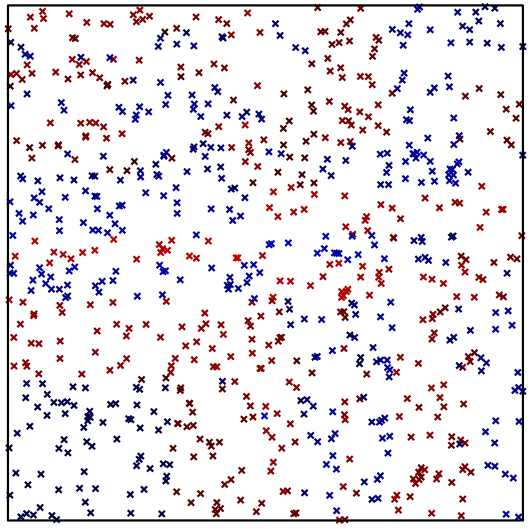
Boosted Tree learning via GBM: $m = 7$



true $f(x)$



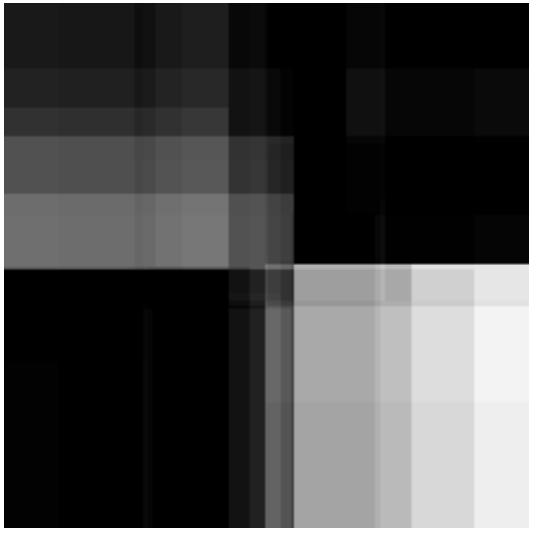
current estimate $f_m(x)$



$r_{im} = y_i - f_m(x_i)$

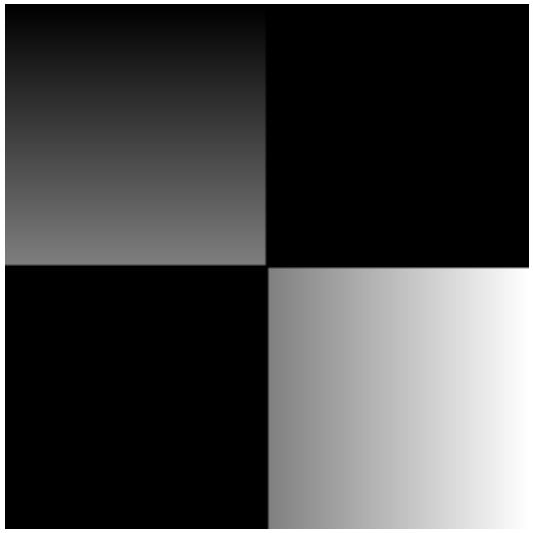
tree added

tree subtracted

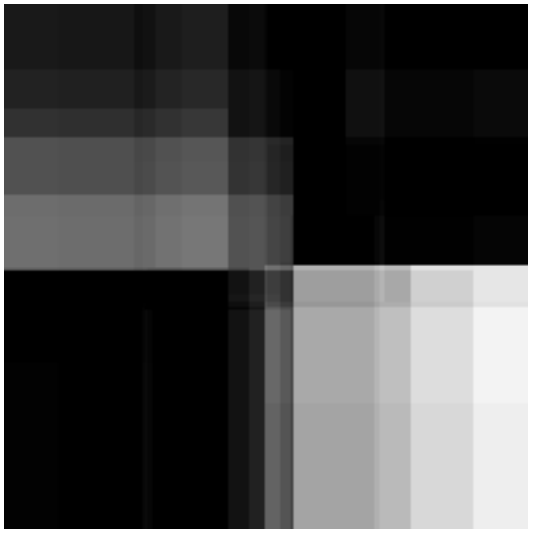


$f_m(x) + T_m(x)$

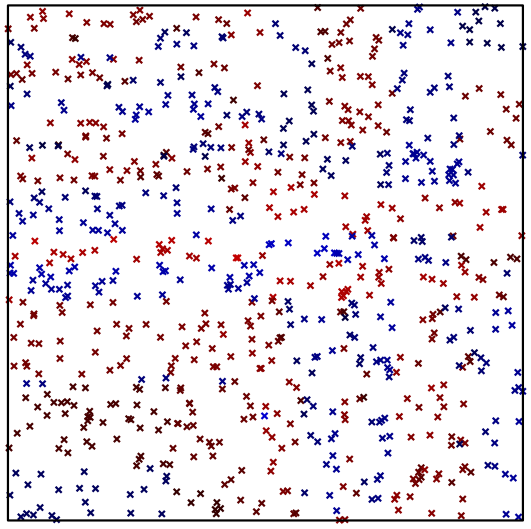
Boosted Tree learning via GBM: $m = 8$



true $f(x)$



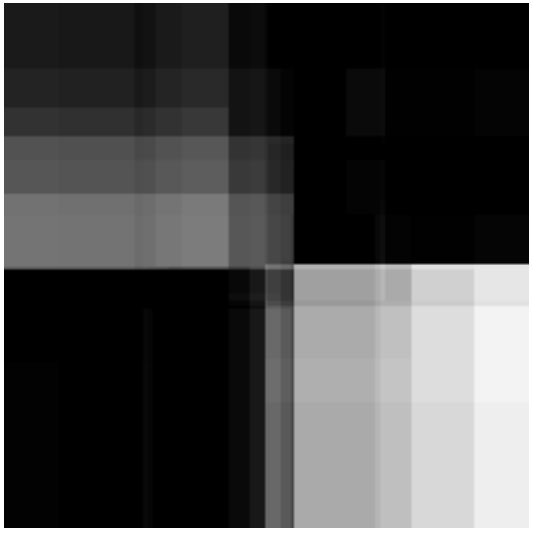
current estimate $f_m(x)$



$r_{im} = y_i - f_m(x_i)$

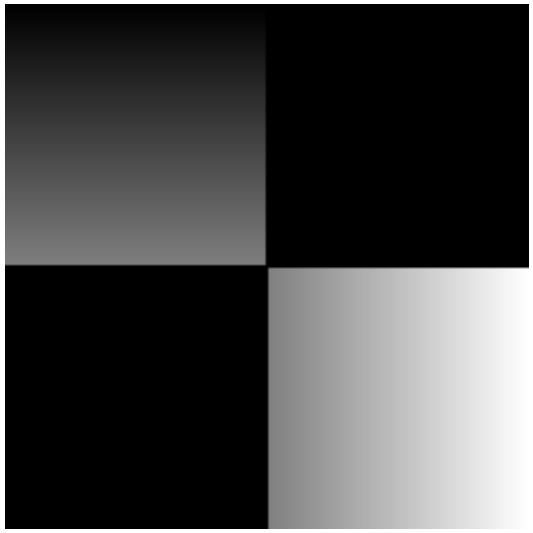
tree added

tree subtracted

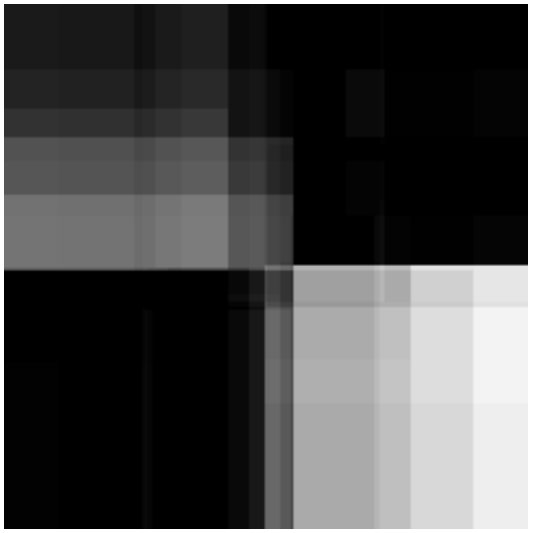


$f_m(x) + T_m(x)$

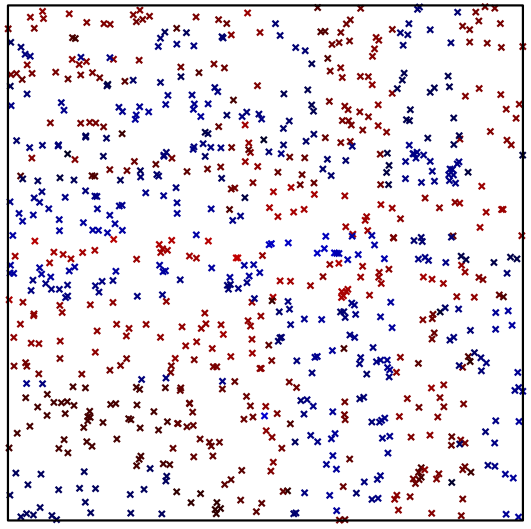
Boosted Tree learning via GBM: $m = 9$



true $f(x)$



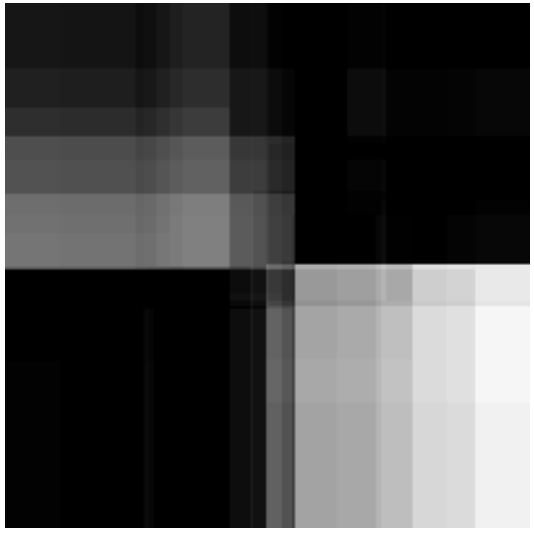
current estimate $f_m(x)$



$r_{im} = y_i - f_m(x_i)$

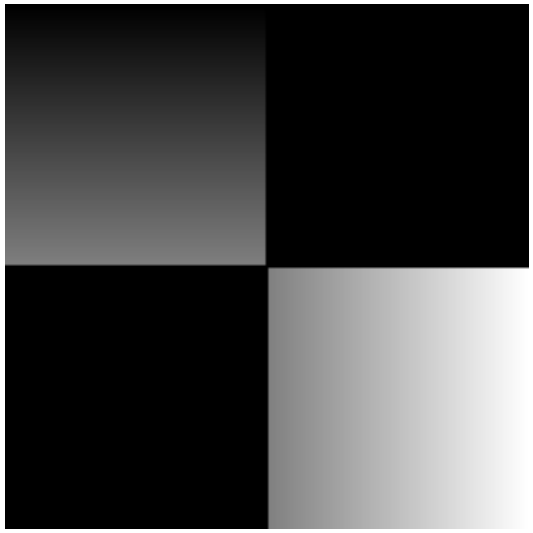
tree added

tree subtracted

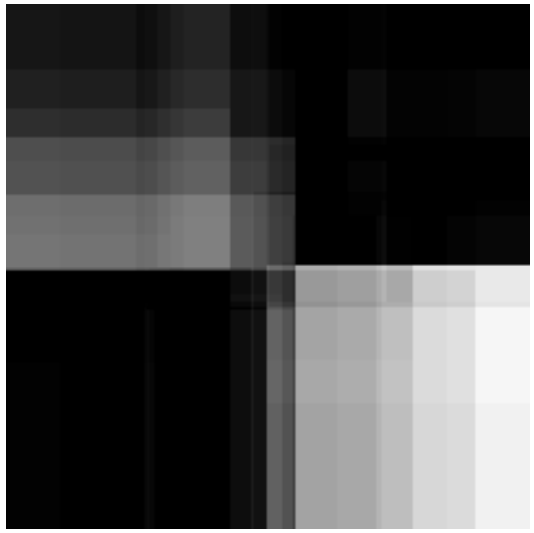


$f_m(x) + T_m(x)$

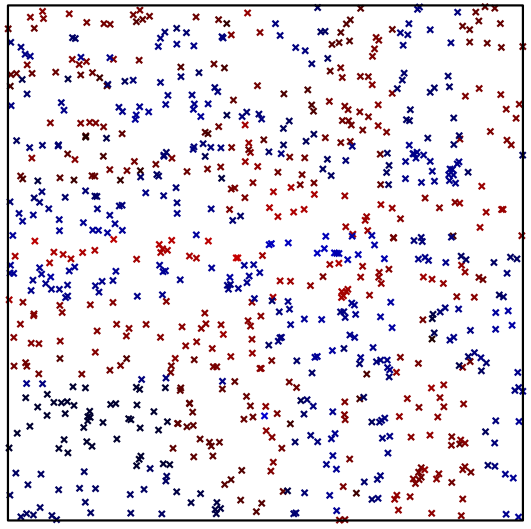
Boosted Tree learning via GBM: $m = 10$



true $f(x)$



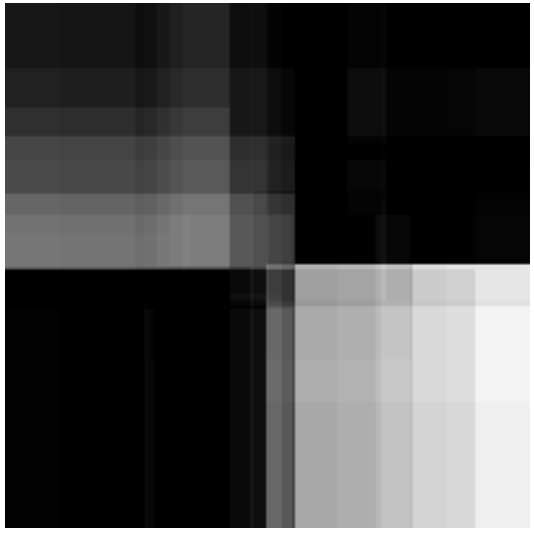
current estimate $f_m(x)$



$r_{im} = y_i - f_m(x_i)$

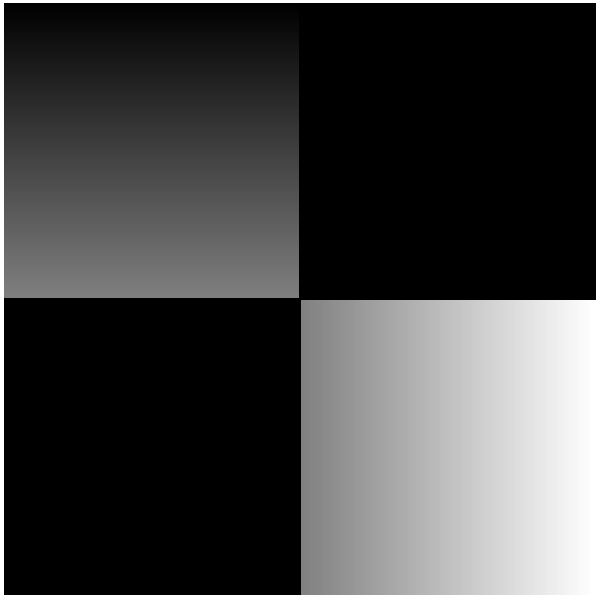
tree added

tree subtracted

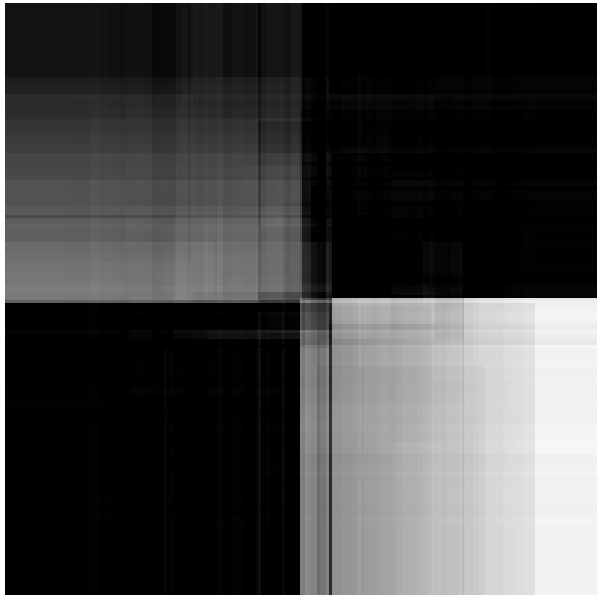


$f_m(x) + T_m(x)$

After 200 iterations



true $f(x)$



$f_{200}(x)$

Introduction

- Random forests (Breiman 2001) build a large collection of de-correlated trees and then averages their predictions.

- On many problems

  performance random forest $\approx$ *performance of boosted tree*

- But random forests are easier to train and tune than boosted trees.

Random Forests

Random forests for regression or classification

- for $b = 1$ to B:
  - Draw bootstrap sample $\mathbf{Z}^*$ of size N from the training data
  - Grow a random-forest tree T_b using $\mathbf{Z}^*$ by recursively
    - ⋆ Select m variables (features) from the p variables (features).
    - ⋆ Pick the best variable/split-point among the m.
    - ⋆ Split the node into two child nodes.
- Output the ensemble of trees $\{T_b\}_1^B$

Make a prediction at a new point x

$$\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x) \quad \leftarrow \textbf{regression}$$

$$\hat{C}_{rf}^B(x) = \text{majority vote } \{\hat{C}_b(x)\}_1^B \quad \leftarrow \textbf{classification}$$

Variance of averages

- Define

$$S_B = X_1 + \cdots + X_B$$

  where each $X_i \sim p(X)$

- If X_i's are independent of each other and $\text{Var}\{X_i\} = \sigma^2$ then

$$\text{Var}\{S_B\} = \frac{1}{B}\sigma^2$$

- If X_i's are not indpt and have pairwise correlation ρ then

$$\text{Var}\{S_B\} = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

- **Note** as $B \to \infty$ then $\text{Var}\{S_B\} \to \rho\sigma^2$

- Therefore higher correlation limits the benefits of averaging.

Variance of averages

- Define

$$S_B = X_1 + \cdots + X_B$$

  where each $X_i \sim p(X)$

- If X_i's are independent of each other and $\text{Var}\{X_i\} = \sigma^2$ then

$$\text{Var}\{S_B\} = \frac{1}{B}\sigma^2$$

- If X_i's are not indpt and have pairwise correlation ρ then

$$\text{Var}\{S_B\} = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

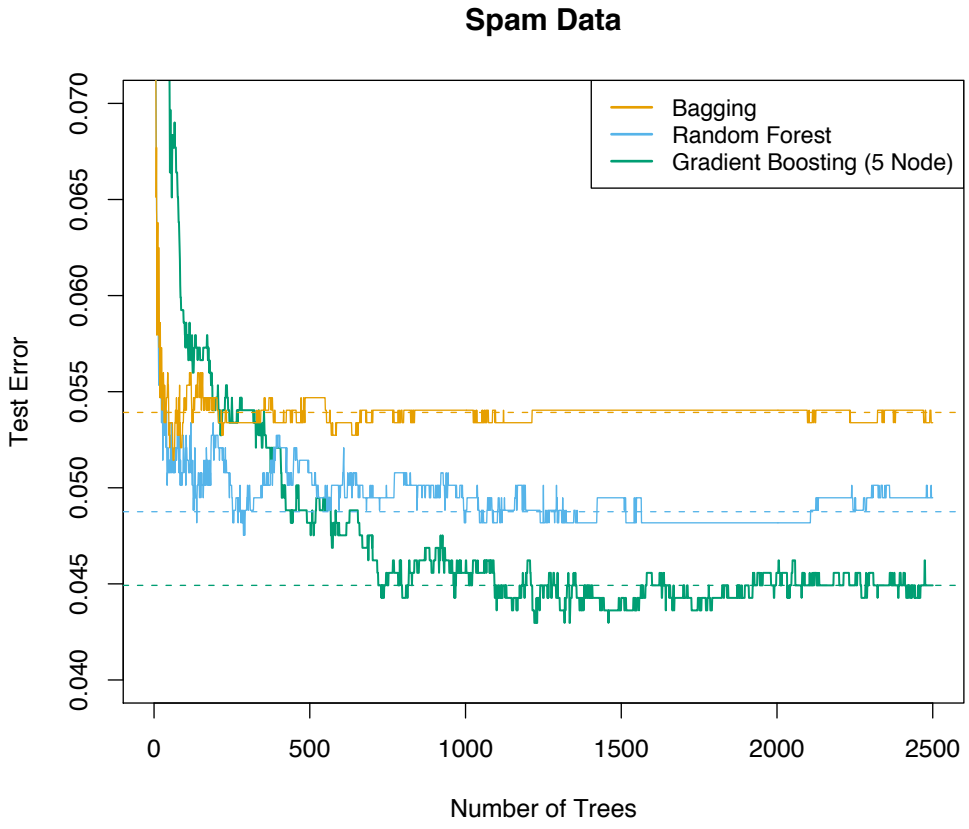
- **Note** as $B \to \infty$ then $\text{Var}\{S_B\} \to \rho\sigma^2$

- Therefore higher correlation limits the benefits of averaging.

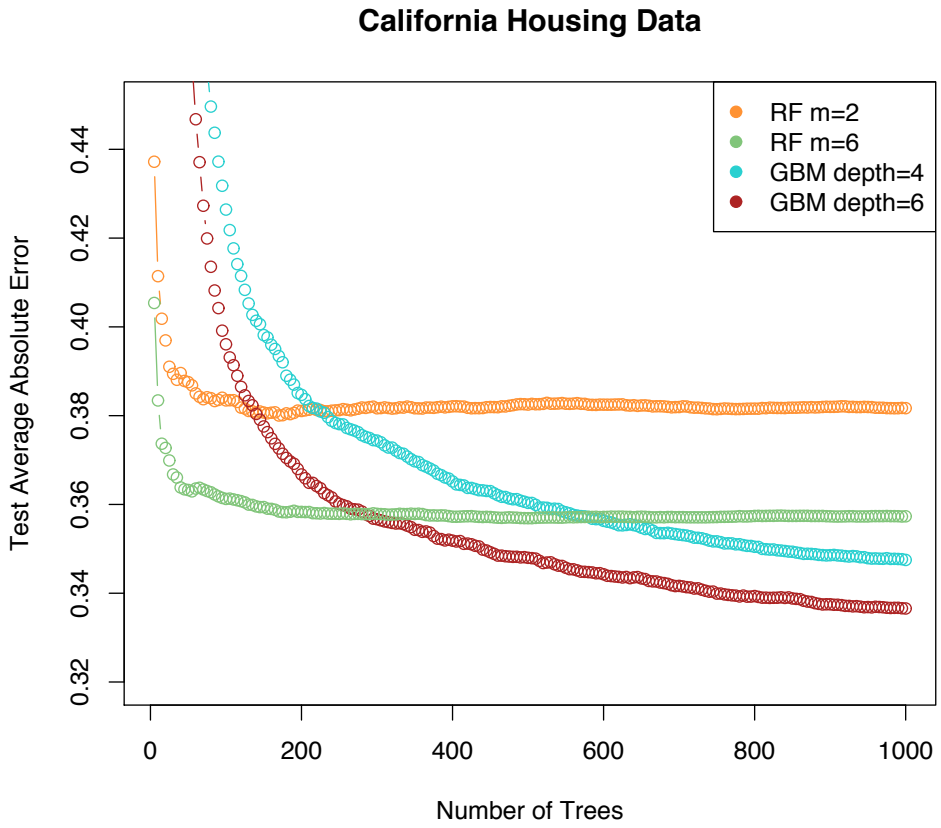
Random forests

- Typically values for m are $\sqrt{p}$ or even as low as 1.
- Reducing m will reduce the correlation between trees.
- Trees benefit alot from the randomization as they have low-bias and high variance.
- Random forests do remarkably well, with very little tuning required.

Random forests - example



Random forests - example



- Random forests stabilize at about 200 trees ($p = 8$).
- At 1000 trees boosting continues to improve.
- Boosting is slowed by shrinkage and smaller depth trees.
- For larger m the random forests performed no better.

Details of Random Forests

Size of m & n_{min}?

The inventors make the following recommendations for the parameters in the random forest

- **Regression**: $m = \lfloor \sqrt{p} \rfloor$ and $n_{\text{min}} = 1$
- **Classification**: $m = \lfloor p/3 \rfloor$ and $n_{\text{min}} = 5$

Out of Bag Samples

- For each observation $z_i = (x_i, y_i)$ its *out-of-bag* estimate is

$$\hat{f}_{\text{oob}}(x_i) = \sum_{b \in \mathcal{B}_i} T_b(x_i)$$

  where $\mathcal{B}_i$ is the index of the bootstrap samples in which z_i did not appear.

- The OOB error estimate $\approx n$-fold cross validation

- Therefore can predict test-error along the way without using cross-validation.

Random Forests and Noisy Variables

- With small m performance will drop as the ratio of relevant variables decrease
- Probability of choosing an irrelevant feature is

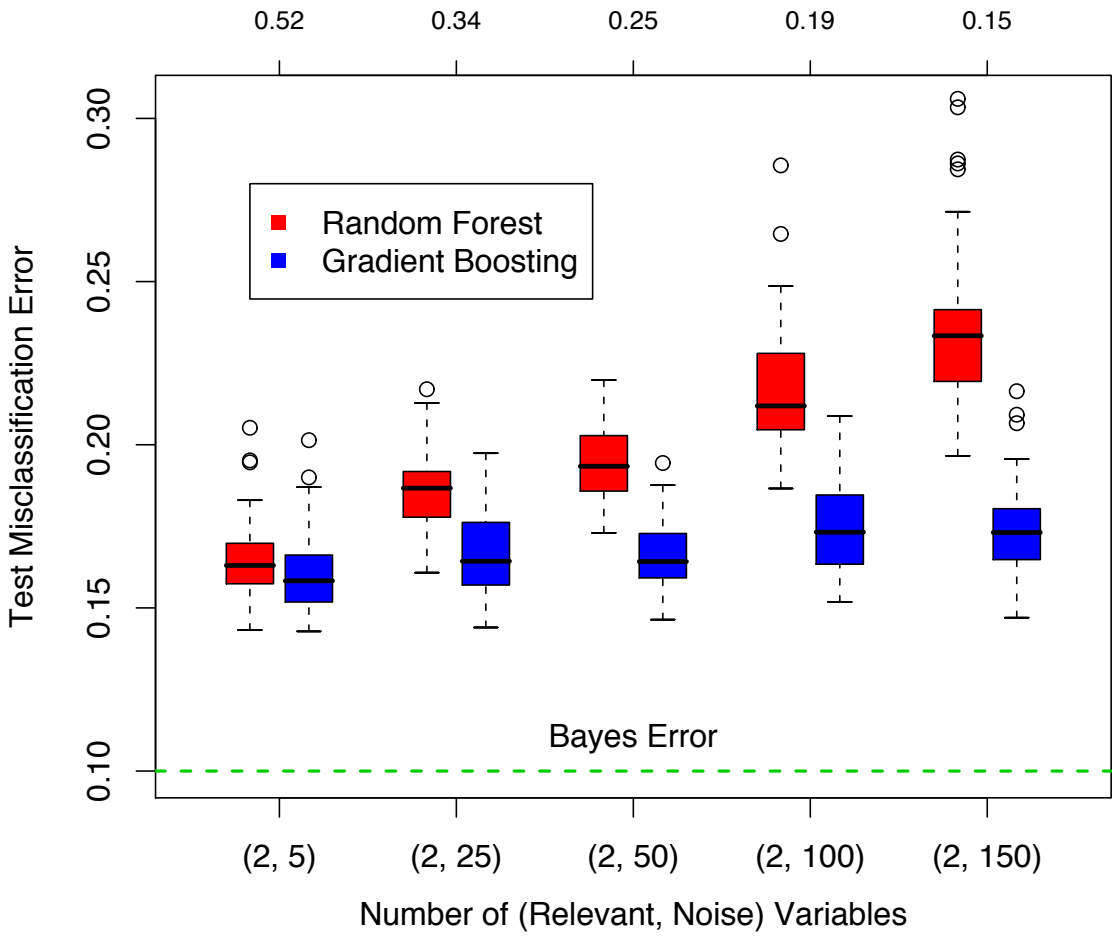
$$p = \frac{n_{\text{irrel}}}{n_{\text{rel}} + n_{\text{irrel}}}$$

- To learn a split node the chance of choosing at least one relevant variable (if n_{irrel} is large) $\approx$

$$1 - p^m$$

- However, random forests seem relatively robust to an increase in the number of noise features....

Random Forests and Noisy Variables - example

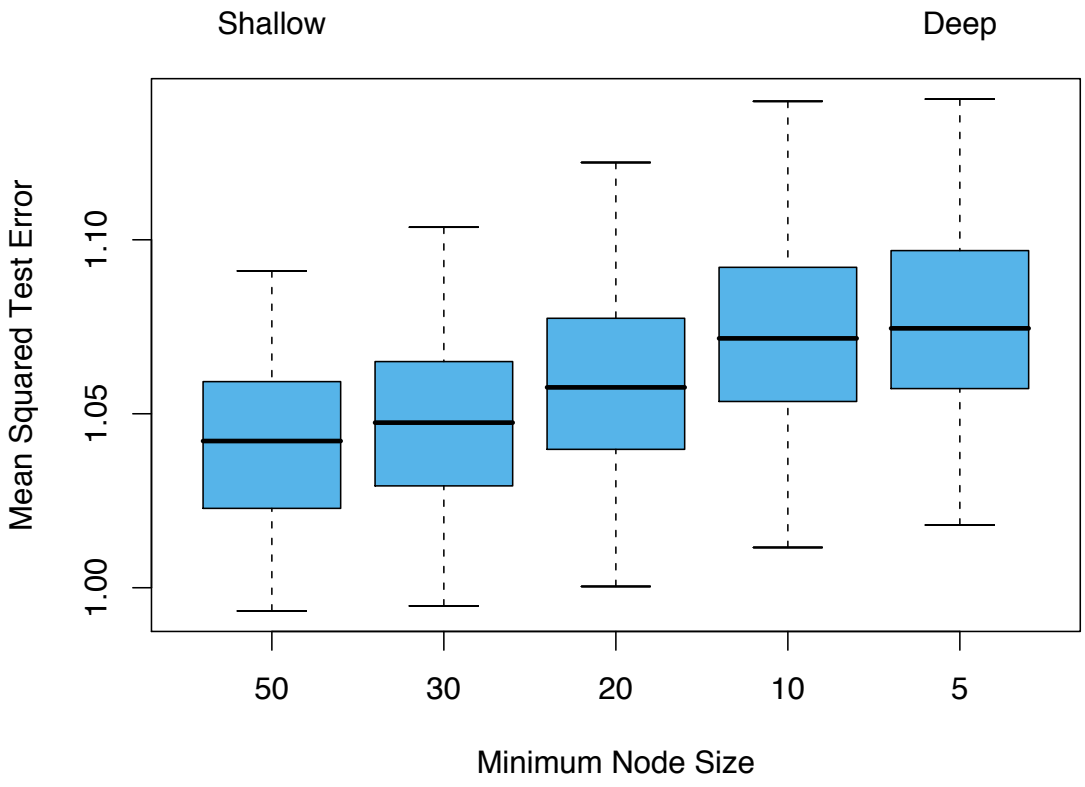


Random Forests and overfitting

$$\hat{f}_{\text{rf}}(x) = \mathsf{E}_{\Theta}\, T(x;\Theta) = \lim_{B\to\infty} \hat{f}_{\text{rf}}^{B}(x)$$

- The distribution of Θ is conditional on the training data.
- May have higher variance if fit a deep tree.
- Authors' experience: using full-grown tree does not incur much cost.
- **Note**: Classifiers are much less sensitive to variance and the effect of over-fitting is seldom seen with random-forest classification.

Random Forests and overfitting



Analysis of Random Forests

Variance and De-Correlation Effect

- The limiting form of the random forest regression estimate is

$$\hat{f}_{\text{rf}}(x) = \mathsf{E}_{\Theta|\mathbf{Z}}\{\, T(x; \Theta(\mathbf{Z}))\,\}$$

- The variance of this estimate at x is

$$\hat{f}_{\text{rf}}(x) = \rho(x)\,\sigma^2(x)$$

where

- $\rho(x)$ is the sampling correlation between any pair of trees

$$\rho(x) = \text{corr}\{\, T(x; \Theta_1(\mathbf{Z})), T(x; \Theta_2(\mathbf{Z}))\,\}$$

where $\Theta_1(\mathbf{Z})$ and $\Theta_2(\mathbf{Z})$ are a randomly drawn pair of random forests grown to the randomly sampled $\mathbf{Z}$.

- $\sigma^2(x) =$ sampling variance of any single randomly drawn tree

$$\sigma^2(x) = \text{Var}\{\, T(x; \Theta(\mathbf{Z}))\,\}$$

Variance and De-Correlation Effect

The variability averaged over these calculations is both:

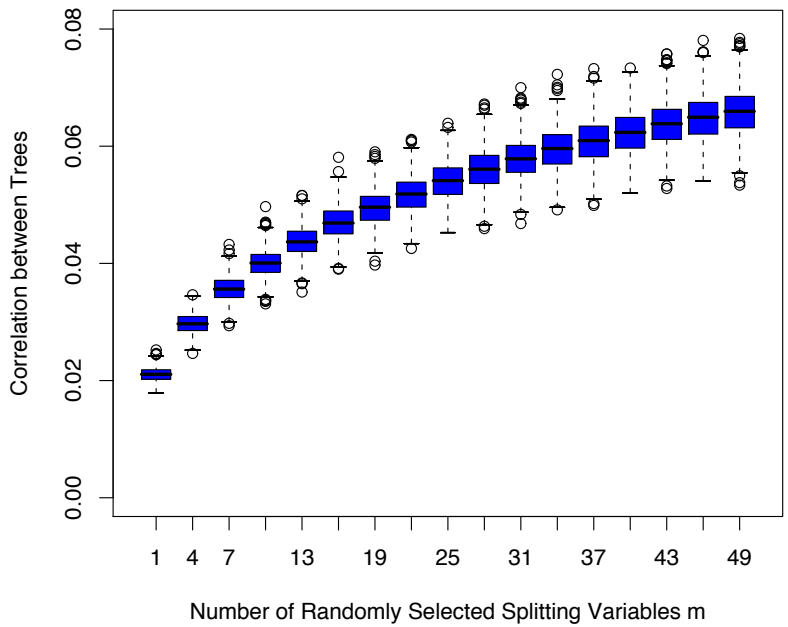
- conditional on $\mathbf{Z}$: due to bootstrap sample and feature sampling at each split **and**
- a result of the sampling variability of $\mathbf{Z}$ itself.

Note: the conditional covariance of a pair of tree fits at x is zero, because bootstrap and feature sampling is i.i.d.

Simple Example: Correlation between trees

$$Y = \frac{1}{\sqrt{50}} \sum_{j=1}^{50} X_j + \epsilon$$

with all the X_j and ϵ iid Gaussian.



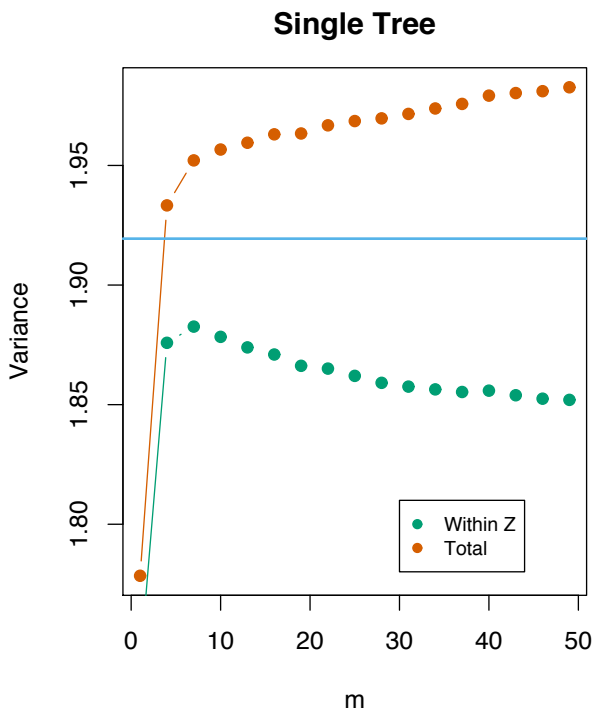
- Use 500 training sets of size 100
- Single test set of size 600

Variance of single tree predictors

The total variance can be decomposed into two parts

$$\mathrm{Var}_{\Theta,\mathbf{Z}}\{T(x;\Theta(\mathbf{Z}))\} = \mathrm{Var}_{\mathbf{Z}}\{\mathrm{E}_{\Theta|\mathbf{Z}}\{T(x;\Theta(\mathbf{Z}))\}\} + \mathrm{E}_{\mathbf{Z}}\{\mathrm{Var}_{\Theta|\mathbf{Z}}\{T(x;\Theta(\mathbf{Z}))\}\}$$

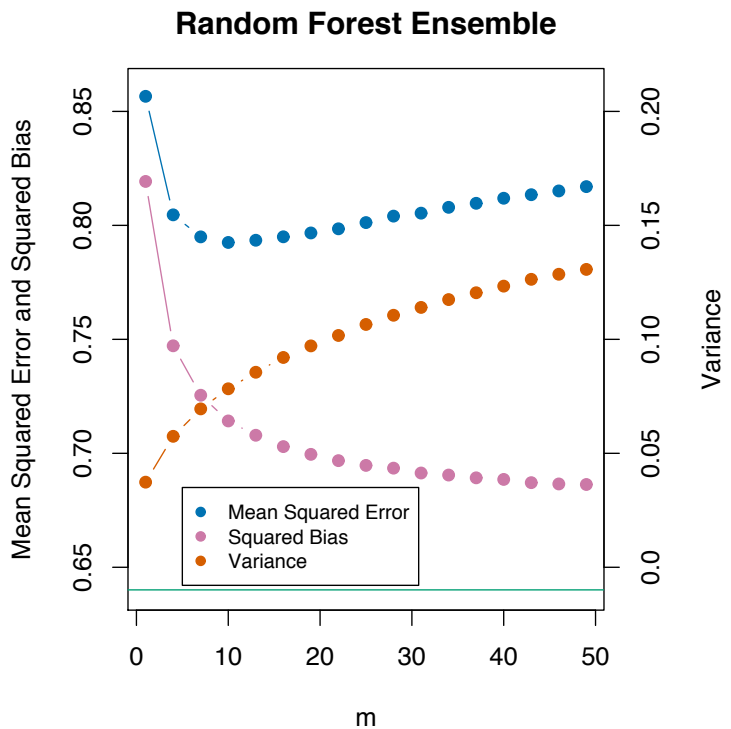
$$\text{Total Variance} = \mathrm{Var}\{\hat{f}_{\mathrm{rf}}(x)\} + \text{within-}\mathbf{Z}\text{ Variance}$$



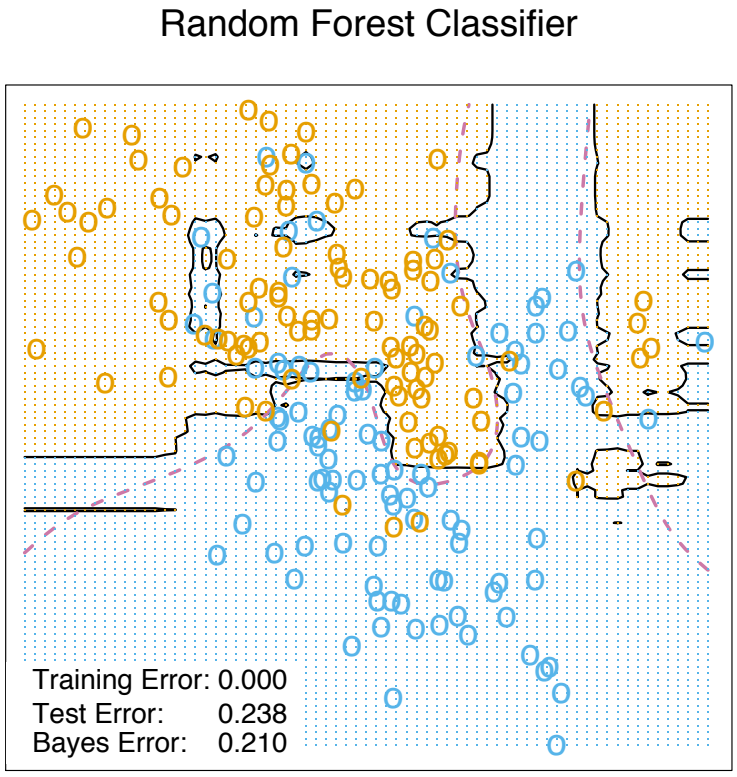
(numbers estimated by averaging over 600 randomly chosen x)

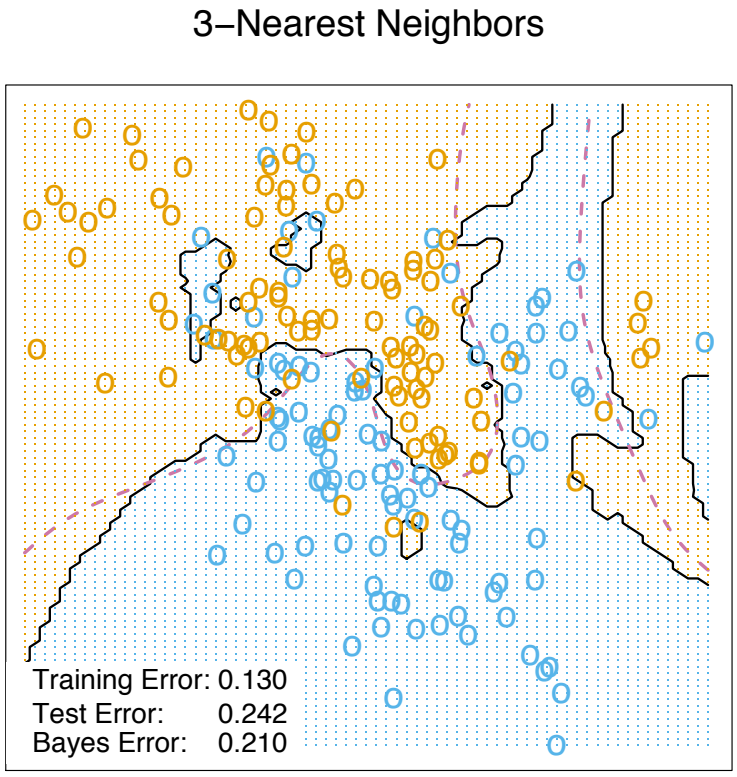
Bias

- Bias of a rf is the same as the bias of any of the individual sampled trees $T(x; \Theta(\mathbf{Z}))$
- The improvements made by random forests are solely a result of variance reduction.
- General trend as m decreases, the bias increases.



Random Forests and k-nearest neighbour have similarities





Ensemble Learning

- **Ensemble learning**

  Build a prediction model by combining the strengths of a collection of simpler base models.

- Examples of ensemble methods
  - Bagging
  - Boosting
  - Stacking
  - Dictionary methods....

- Ensemble consists of two tasks:
  - Build a population of base learners from training data
  - Combine base learners to form a composite predictor

- Focus on these issues in this chapter.

- **Ensemble learning**

  Build a prediction model by combining the strengths of a collection of simpler base models.

- Examples of ensemble methods
  - Bagging
  - Boosting
  - Stacking
  - Dictionary methods....

- Ensemble consists of two tasks:
  - Build a population of base learners from training data
  - Combine base learners to form a composite predictor

- Focus on these issues in this chapter.

Boosting and Regularization Paths

- Consider the dictionary of all J-terminal node regression trees $\mathcal{T} = \{T_k\}$ that could be realized by the training data.

- The linear model is

$$f(x) = \sum_{i=1}^{|\mathcal{T}|} \alpha_k \, T_k(x)$$

- Estimation of α's from training data requires regularization

$$\min_{\alpha} \left\{ \sum_{i=1}^{n} \left(y_i - \sum_{i=1}^{|\mathcal{T}|} \alpha_k \, T_k(x) \right)^2 + \lambda \, J(\alpha) \right\}$$

Ridge regression: $J(\alpha) = \sum_{k=1}^{|\mathcal{T}|} |\alpha_k|^2$

Lasso: $J(\alpha) = \sum_{k=1}^{|\mathcal{T}|} |\alpha_k|$

- Solution to the **lasso** solution with moderate to large λ gives a sparse α.
- If $|\mathcal{T}|$ is very large then solving the optimization with the lasso penalty is not possible.
- A feasible forward stagewise strategy exists that closely approximates the effect of lasso

Forward Stagewise Linear Regression

- Initialize $\tilde{\alpha}_k, k = 1, \ldots, K$. Set $\epsilon > 0$ small and M large.

- for $m = 1$ to M:

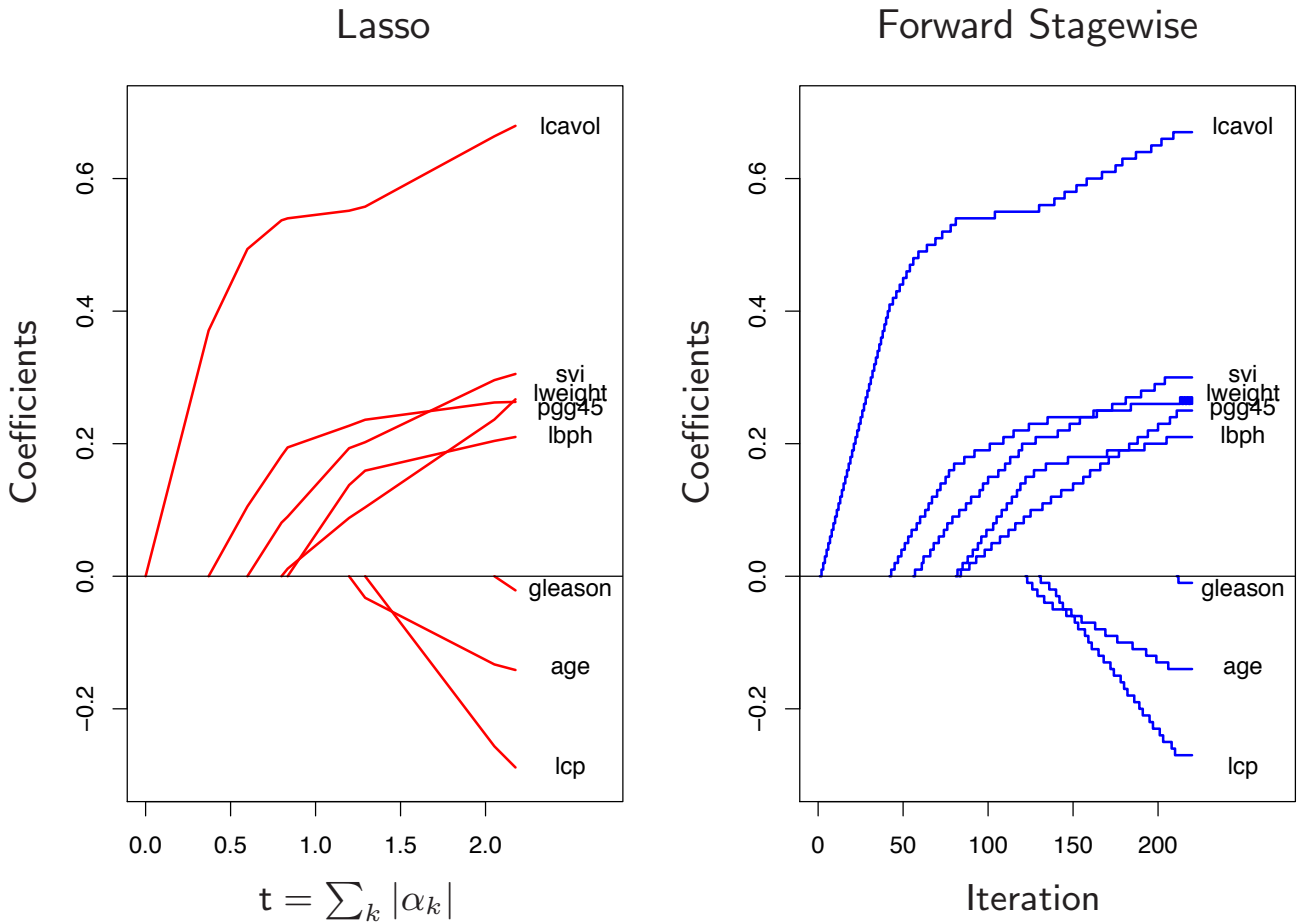
  ⋆ $(\beta^*, k^*) = \arg\min_{\beta,k} \sum_{i=1}^{n} \left(y_i - \sum_{l=1}^{|\mathcal{T}|} \tilde{\alpha}_l T_l(x_i) - \beta\, T_k(x_i) \right)^2$

  ⋆ $\tilde{\alpha}_{k^*} \rightarrow \tilde{\alpha}_{k^*} + \epsilon\, \mathsf{sign}(\beta^*)$

- Output:

$$f_M(x) = \sum_{k=1}^{|\mathcal{T}|} \tilde{\alpha}_k T_k(x)$$

Similarity between Lasso & Forward Stagewise Paths



- 7 dimensional input vectors, $M = 220, \epsilon = .01$
- $\mathcal{T} = \{X_1, X_2, \ldots, X_7\}$

The "Bet on Sparsity" Principle

- Minimizing a loss function with a L_1 penalty is slow and involves searching through the "*model space*".
- The L_2 penalty is computationally much easier.
- However, L_1 penalty is better suited to sparse situations.
- Consider this example:
  - 10,000 data points
  - Model is a linear combination of a million trees
  - If the coefficients for these trees arise from a Gaussian distribution $\Longrightarrow$ best predictor is ridge regression $\Longrightarrow$ L_2 penalty.
  - But, if there are only a small number coefficients that are nonzero, the L_1 penalty, will work better.

  In the dense scenario, L_2 best but will fail as too little data to estimate 1 million coefficients.

  In the sparse setting, L_1 penalty can do well but L_2 penalty will fail.

The "Bet on Sparsity" Principle

- Minimizing a loss function with a L_1 penalty is slow and involves searching through the "*model space*".
- The L_2 penalty is computationally much easier.
- However, L_1 penalty is better suited to sparse situations.
- Consider this example:
  - 10,000 data points
  - Model is a linear combination of a million trees
  - If the coefficients for these trees arise from a Gaussian distribution $\Longrightarrow$ best predictor is ridge regression $\Longrightarrow$ L_2 penalty.
  - But, if there are only a small number coefficients that are nonzero, the L_1 penalty, will work better.

  In the dense scenario, L_2 best but will fail as too little data to estimate 1 million coefficients.

  In the sparse setting, L_1 penalty can do well but L_2 penalty will fail.

The "Bet on Sparsity" Principle

Take home message: For high-dimensional problems

Use a procedure that does well in *sparse problems*, since no procedure does well in dense problems.

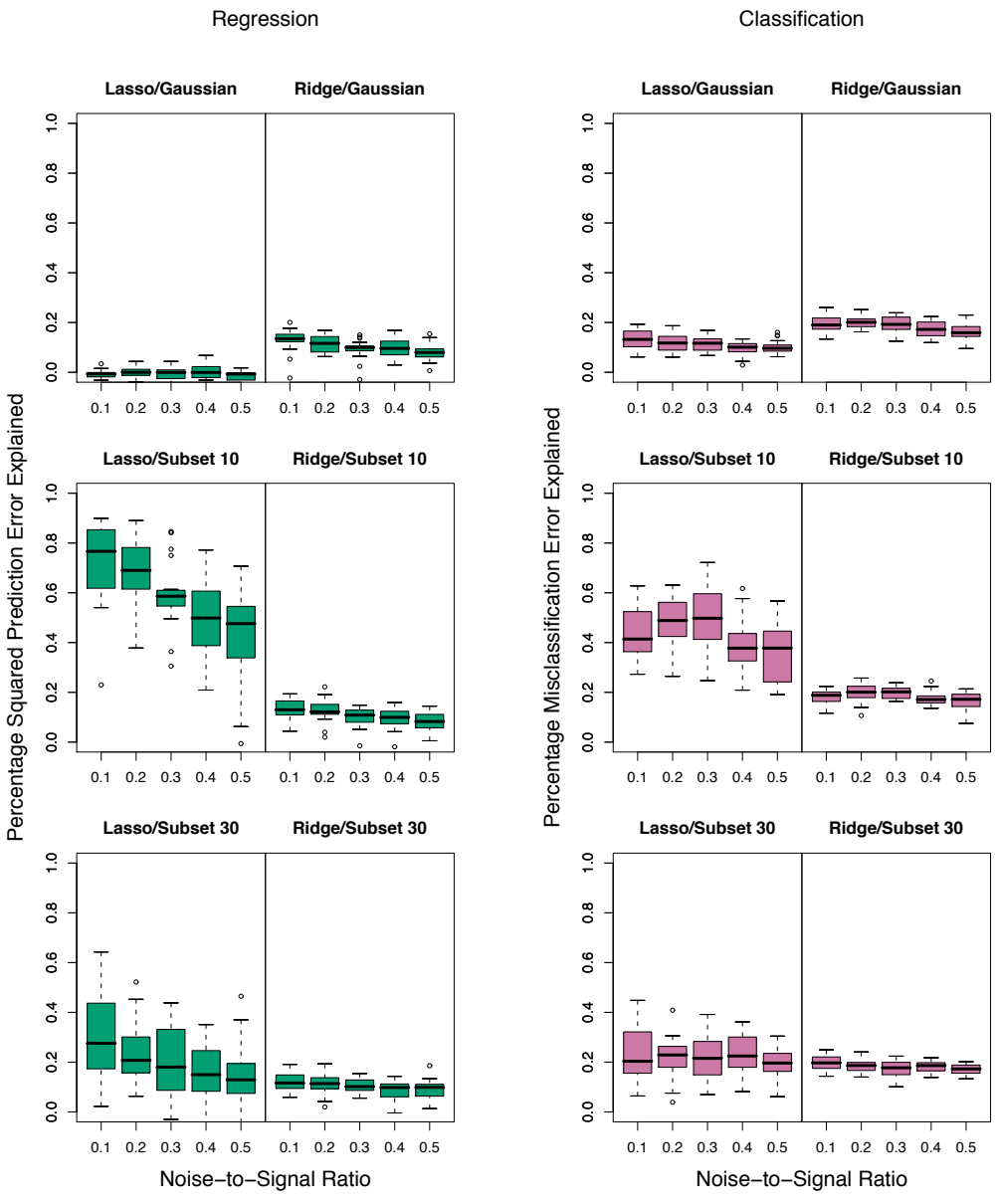
Comment need some qualification:

- Sparseness/denseness depends on target function and dictionary $\mathcal{T}$
- Notion of *sparse* Vs *dense* is relative to size of the training data set and/or the noise-to-signal ratio.

  More training data $\Longrightarrow$ can estimate coeffs with smaller standard errors

  Small NSR $\Longrightarrow$ can identify more non-zero coeffs with a given sample size than with high NSR
- Increase size of the dictionary $\Longrightarrow$ probable sparser representation, but $\Longrightarrow$ harder search problem $\Longrightarrow$ higher variance.

Lasso penalty Vs Ridge penalty



Regression problem

$$Y = X^t\beta + \epsilon$$

with

- $\epsilon \sim \mathcal{N}(0, \sigma^2)$ and
- $X \in \mathbb{R}^{300}$
- **Top row**: $\beta_j \neq 0,\ 1 \leq j \leq 300$
- **Mid row**: 10 non-zero β_j
- **Last row**: 30 non-zero β_j

L_1 estimation is superior in sparse settings.

Learning Ensembles

- How should one learn functions of the form

$$f(x) = \alpha_0 + \sum_{T_k \in \mathcal{T}} \alpha_k T_k(x)$$

  where $\mathcal{T}$ is a dictionary of basis functions - typically trees ?

- Suggested approach
  - Construct a finite dictionary $\mathcal{T}_L = \{T_1(x), \ldots, T_M(x)\}$ from the training data.
  - Build a family of functions $f_\lambda(x)$ by fitting a lasso path

$$\alpha(\lambda) = \arg\min_{\alpha} \sum_{i=1}^{n} L(y_i, \alpha_0 + \textstyle\sum_{m=1}^{M} \alpha_m T_m(x)) + \lambda \displaystyle\sum_{m=1}^{M} |\alpha_m|$$

- Can view this as a way to post-processing a boosted trees or random forest

- How should one learn functions of the form

$$f(x) = \alpha_0 + \sum_{T_k \in \mathcal{T}} \alpha_k T_k(x)$$

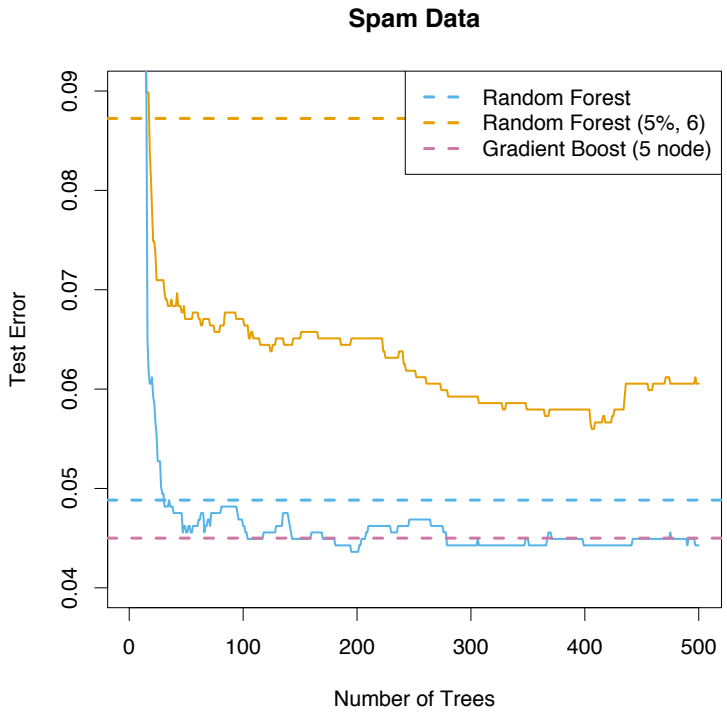
  where $\mathcal{T}$ is a dictionary of basis functions - typically trees ?

- Suggested approach
  - Construct a finite dictionary $\mathcal{T}_L = \{T_1(x), \ldots, T_M(x)\}$ from the training data.
  - Build a family of functions $f_\lambda(x)$ by fitting a lasso path

$$\alpha(\lambda) = \arg\min_{\alpha} \sum_{i=1}^{n} L(y_i, \alpha_0 + \textstyle\sum_{m=1}^{M} \alpha_m T_m(x)) + \lambda \displaystyle\sum_{m=1}^{M} |\alpha_m|$$

- Can view this as a way to post-processing a boosted trees or random forest

Example



- Classification problem, 57 dimensional feature vector.
- Solid curves are the post-processed functions.
- Dashed line - test error of random forest, using 1000 tree grown to maximum depth ($m = 7$).
- Dashed line - test error of random forest where 5% of data used to grow each shallow tree in the forest.

Learning a Good Ensemble

- Not all ensembles $\mathcal{T}_L$ will perform well with post-processing.
- For the ensemble of basis functions $\mathcal{T}_L$ want
  - a collection that offers **good coverage** in the places needed
  - and are **sufficiently different** from each offer to allow the post-processing to be effective.
- Freidman and Popescu suggested an ensemble-generation algorithm....

Importance Sampled Learning Ensemble Generation

- $f_0(x) = \arg\min_c \sum_{i=1}^{n} L(y_i, c)$
- For $m = 1$ to M do
  - $\gamma_m = \arg\min_\gamma \sum_{i \in S_m(\eta)} L\left(y_i, f_{m-1}(x_i) + b(x_i; \gamma)\right)$
  - $f_m(x) = f_{m-1}(x) + \nu\, b(x; \gamma)$
- $\mathcal{T}_{\text{ISLE}} = \{b(x; \gamma_1), b(x; \gamma_2), \ldots, b(x; \gamma_M)\}$

where

- $\nu \in [0, 1]$ introduces **memory** into the randomization process,
- $S_m(\eta)$ refers to a subsample $\eta \cdot n$, $\eta \in [0, 1]$, of the training observations.
- Suggested values of eta are$\eta \leq .5$ and for large n pick $\eta \approx 1/\sqrt{n}$.

A number of familiar randomization schemes are special cases of this algorithm:

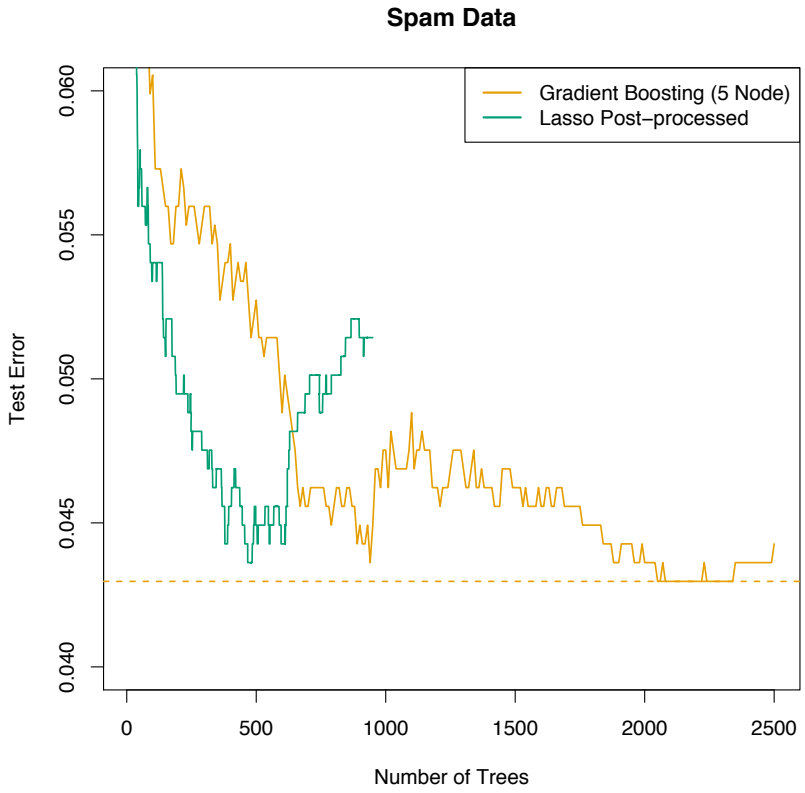
- **Bagging**:
  Has $\eta = 1$, samples with replacement and $\nu = 1$

- **Random forest**:
  Sampling is similar, with more randomness introduced by the selection of the splitting variable.

- **Gradient boosting**:
  With shrinkage uses $\eta = 1$, but does not produce sufficient width σ.

- **Stochastic gradient boosting**:
  Follows the recipe exactly.

Example



ISLE, $\eta = .5, \nu = 0.05$, used to generate an ensemble of trees with 5 terminal nodes

Example

- Consider this function of $X \sim U[0,1]^{100}$

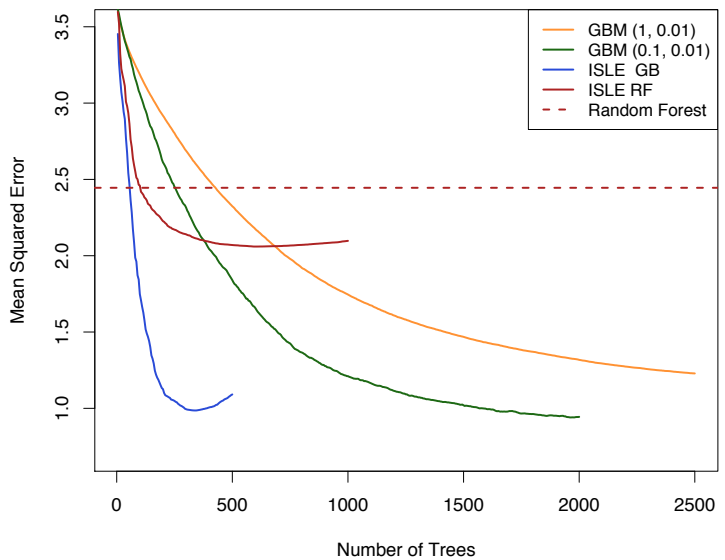
$$f(X) = 10 \cdot \prod_{j=1}^{5} \exp\{-2X_j^2\} + \sum_{j=6}^{36} X_j$$

- The response variable, with $\sigma = 1.3$, is

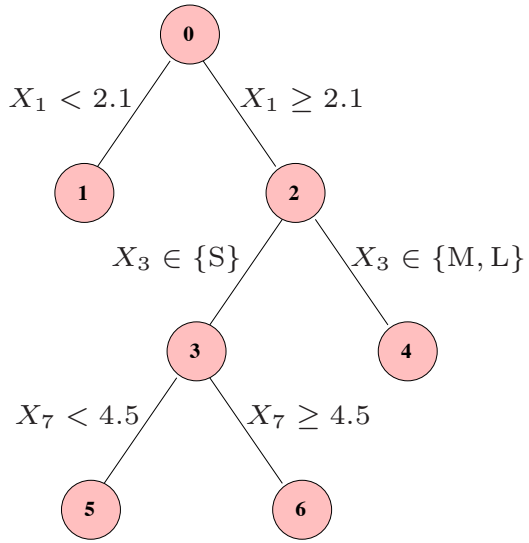
$$Y = f(X) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

- Estimate $f(X)$ from a training set of size $n = 1000$.

- Results: $n_{\text{test}} = 600$ and averaged over 20 different training sets.



A typical tree in an ensemble from which rules can be derived



Derived rules:

$$
\begin{aligned}
R_1(X) &= I(X_1 < 2.1) \\
R_2(X) &= I(X_1 \geq 2.1) \\
R_3(X) &= I(X_1 \geq 2.1) \cdot I(X_3 \in \{S\}) \\
R_4(X) &= I(X_1 \geq 2.1) \cdot I(X_3 \in \{M, L\}) \\
R_5(X) &= I(X_1 \geq 2.1) \cdot I(X_3 \in \{S\}) \cdot I(X_7 < 4.5) \\
R_6(X) &= I(X_1 \geq 2.1) \cdot I(X_3 \in \{S\}) \cdot I(X_7 \geq 4.5)
\end{aligned}
$$

- This rule set is an over-complete basis for the tree.

$$
\begin{aligned}
R_1(X) &= I(X_1 < 2.1) \\
R_2(X) &= I(X_1 \geq 2.1) \\
R_3(X) &= I(X_1 \geq 2.1) \cdot I(X_3 \in \{S\}) \\
R_4(X) &= I(X_1 \geq 2.1) \cdot I(X_3 \in \{M, L\}) \\
R_5(X) &= I(X_1 \geq 2.1) \cdot I(X_3 \in \{S\}) \cdot I(X_7 < 4.5) \\
R_6(X) &= I(X_1 \geq 2.1) \cdot I(X_3 \in \{S\}) \cdot I(X_7 \geq 4.5)
\end{aligned}
$$

- For each tree $T_m \in \mathcal{T}$ construct its ensemble of rules $\mathcal{T}_{\text{RULE}}^m$ and set

$$\mathcal{T}_{\text{RULE}} = \bigcup_{m=1}^{M} \mathcal{T}_{\text{RULE}}^m$$

- This ensemble then treated like any other and post-processed.
- Via the lasso, that is, find α to minimize

$$\arg\min_{\alpha} \left\{ \sum_{i=1}^{n} L(y_i, \textstyle\sum_{k=1}^{K} \alpha_k \, R_k(x_i)) + \lambda \textstyle\sum_{k=1}^{K} |\alpha_k| \right\}$$

or some other regularized procedure.

Rule Ensembles: Advantages?

- Space of possible models enlarged $\implies$ potential greater capacity of final f.
- Rules are easier to interpret than trees.
- Can augment $\mathcal{T}_{\text{RULE}}$ with each variable X_j to allow ensemble to also model linear functions.

Example

- Consider this function of $X \sim U[0,1]^{100}$

$$f(X) = 10 \cdot \prod_{j=1}^{5} \exp\{-2X_j^2\} + \sum_{j=6}^{36} X_j$$

- The response variable, with $\sigma = 1.3$, is

$$Y = f(X) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

- Estimate $f(X)$ from a training set of size $n = 1000$.

- Results: $n_{\text{test}} = 600$ and averaged over 20 different training sets.