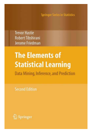
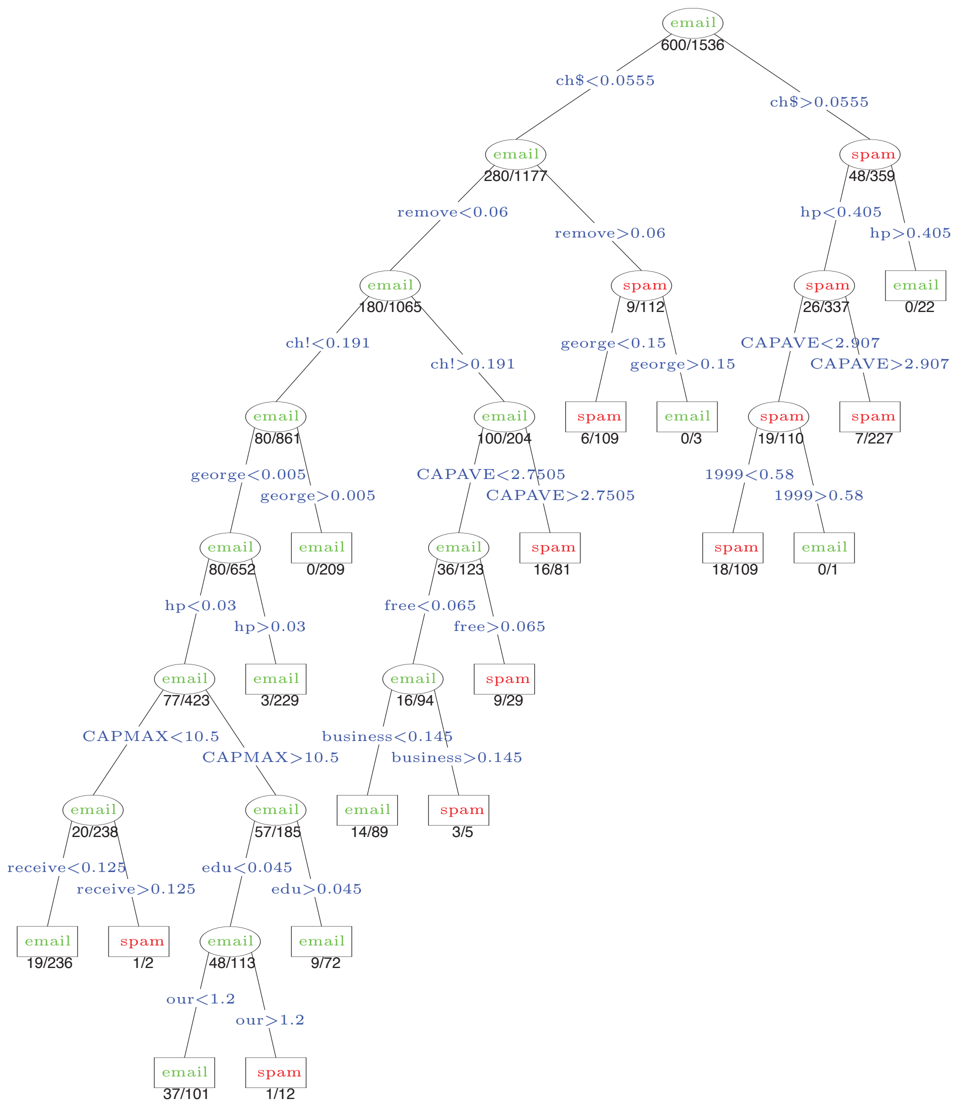
Ensemble Learners

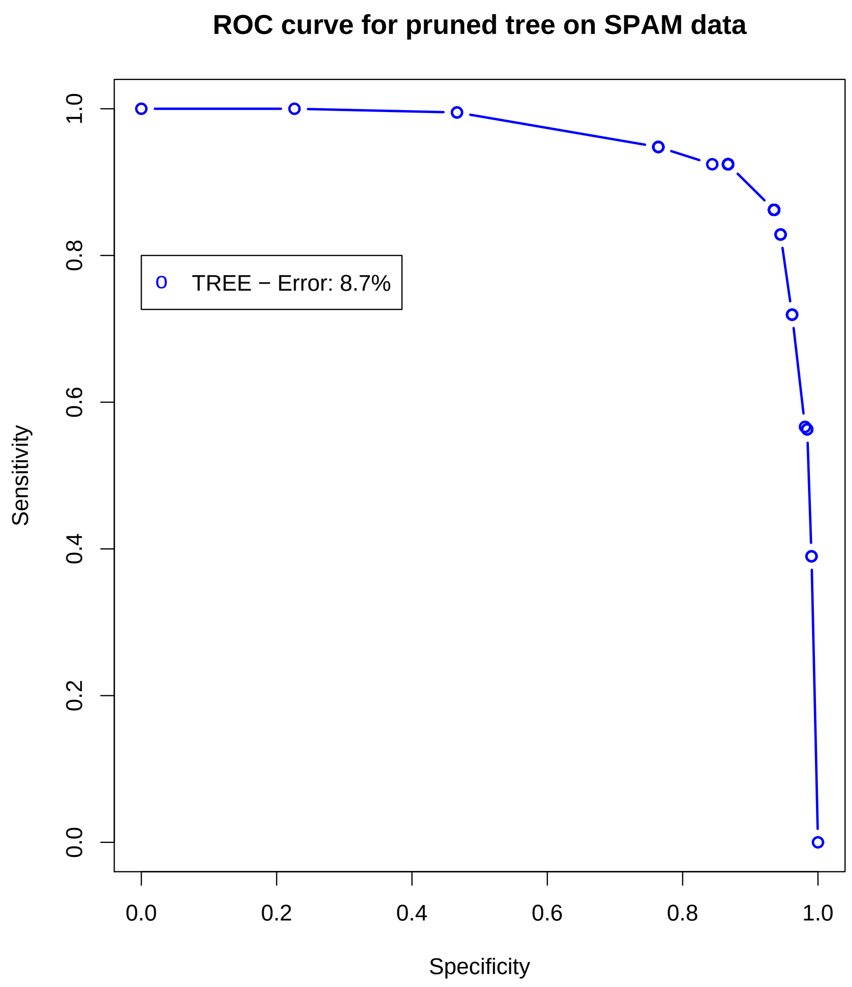
- Classification Trees (Done)
- Bagging: Averaging Trees
- Random Forests: Cleverer Averaging of Trees
- Boosting: Cleverest Averaging of Trees
- Details in  chaps 10, 15 and 16

Methods for dramatically improving the performance of weak learners such as Trees. Classification trees are adaptive and robust, but do not make good prediction models. The techniques discussed here enhance their performance considerably.

Properties of Trees

- [✔]Can handle huge datasets
- [✔]Can handle *mixed* predictors—quantitative and qualitative
- [✔]Easily ignore redundant variables
- [✔]Handle missing data elegantly through surrogate splits
- [✔]Small trees are easy to interpret
- [✘]Large trees are hard to interpret
- [✘]Prediction performance is often poor


email
600/1536
ch$<0.0555
ch$>0.0555
email
280/1177
spam
48/359
remove<0.06
remove>0.06
hp<0.405
hp>0.405
email
180/1065
spam
9/112
spam
26/337
email
0/22
ch!<0.191
ch!>0.191
george<0.15
george>0.15
CAPAVE<2.907
CAPAVE>2.907
email
80/861
email
100/204
spam
6/109
email
0/3
spam
19/110
spam
7/227
george<0.005
george>0.005
CAPAVE<2.7505
CAPAVE>2.7505
1999<0.58
1999>0.58
email
80/652
email
0/209
email
36/123
spam
16/81
spam
18/109
email
0/1
hp<0.03
hp>0.03
free<0.065
free>0.065
email
77/423
email
3/229
email
16/94
spam
9/29
CAPMAX<10.5
CAPMAX>10.5
business<0.145
business>0.145
email
20/238
email
57/185
email
14/89
spam
3/5
receive<0.125
receive>0.125
edu<0.045
edu>0.045
email
19/236
spam
1/2
email
48/113
email
9/72
our<1.2
our>1.2
email
37/101
spam
1/12



SPAM Data

Overall error rate on test data: 8.7%.

ROC curve obtained by varying the *threshold* c_0 of the classifier:

$C(X) = +1$ if $\widehat{\text{Pr}}(+1|X) > c_0$.

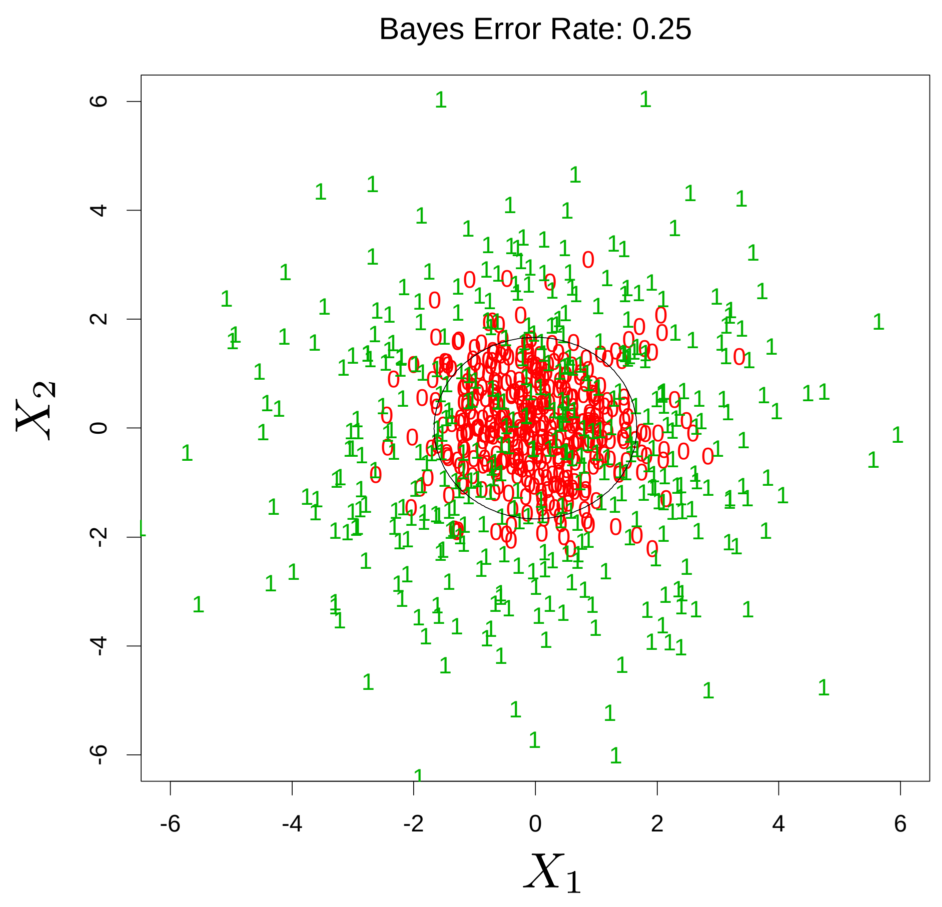
Sensitivity: proportion of true spam identified

Specificity: proportion of true email identified.

We may want specificity to be high, and suffer some spam:

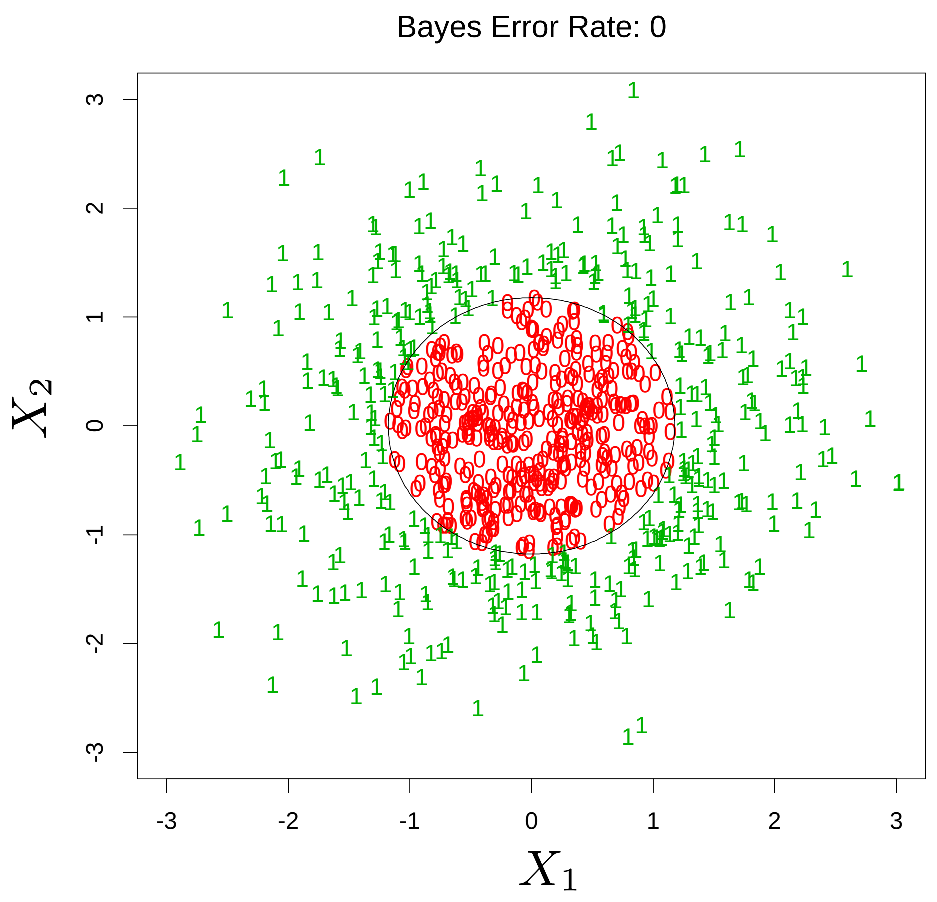
$$\text{Specificity} : 95\% \implies \text{Sensitivity} : 79\%$$

Toy Classification Problem



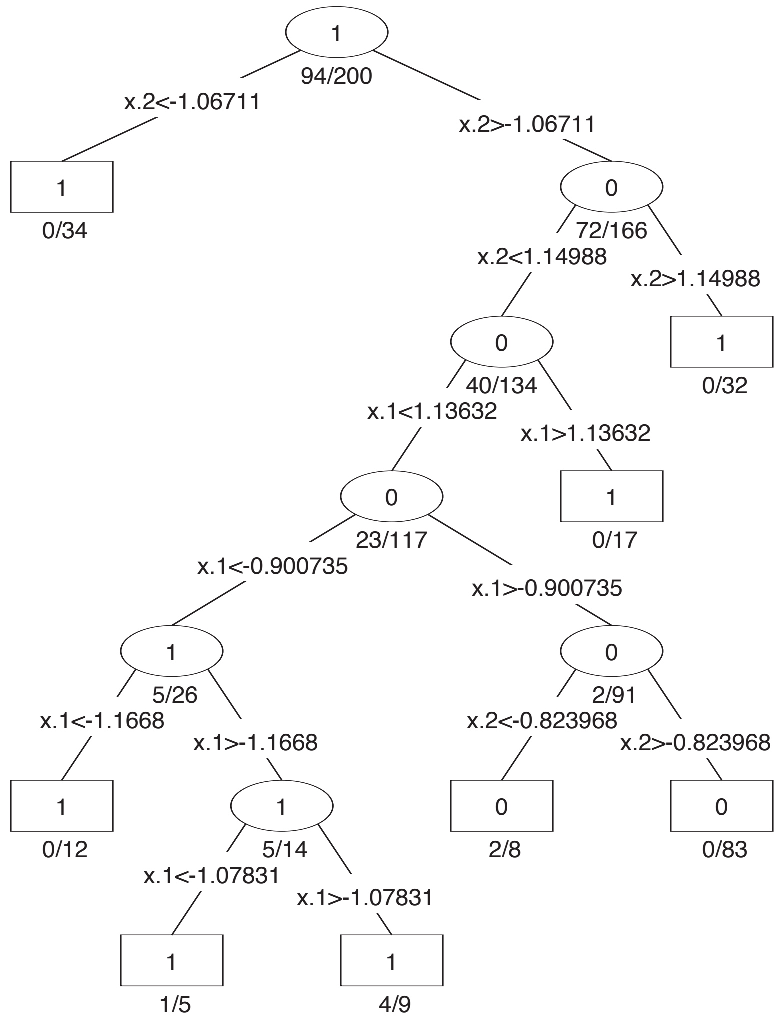
- Data X and Y, with Y taking values $+1$ or -1.
- Here $X =\in \mathbb{R}^2$
- The black boundary is the *Bayes Decision Boundary* - the best one can do. Here 25%
- Goal: given N training pairs (X_i, Y_i) produce a *classifier* $\hat{C}(X) \in \{-1, 1\}$
- Also estimate the *probability* of the class labels $\Pr(Y = +1|X)$.

Toy Example — No Noise

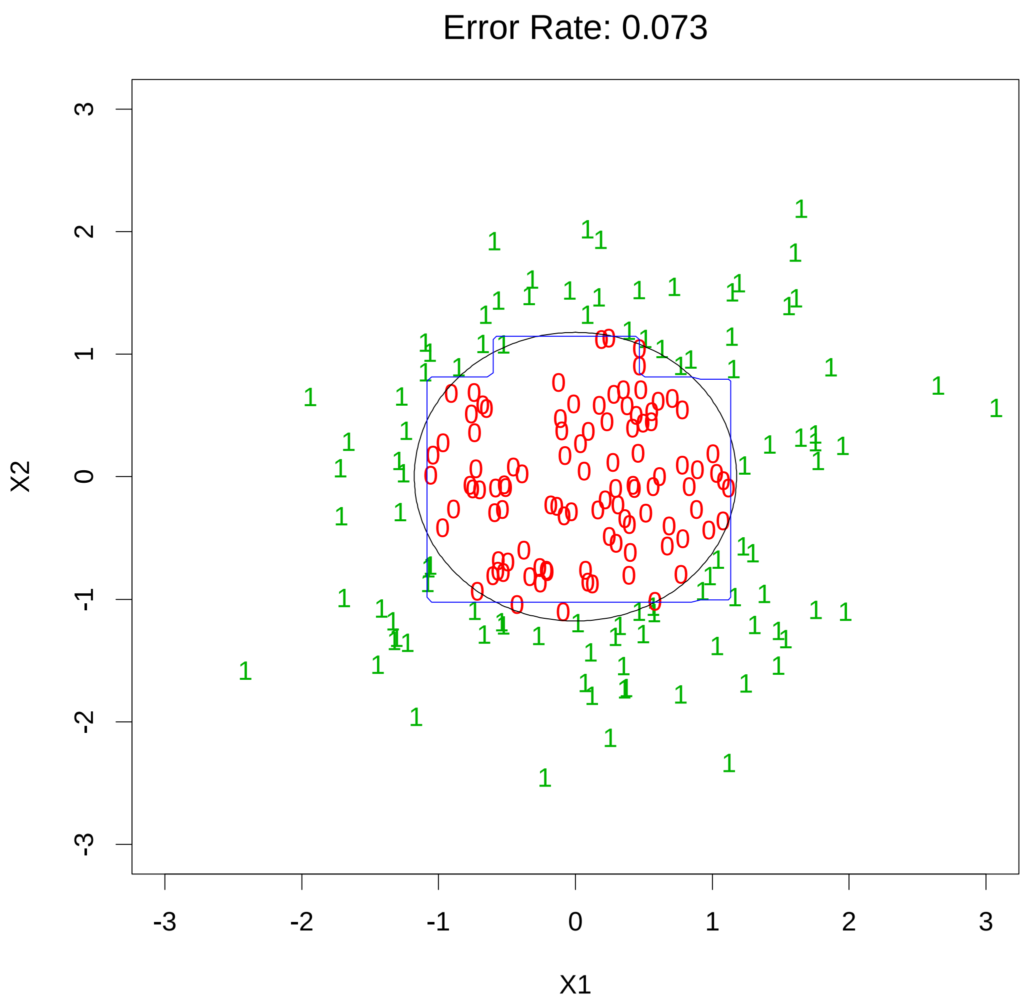


- Deterministic problem; noise comes from sampling distribution of X.
- Use a training sample of size 200.
- Here *Bayes Error* is 0%.

Classification Tree



Decision Boundary: Tree



- The eight-node tree is *boxy*, and has a test-error rate 0f 7.3%
- When the *nested spheres* are in 10-dimensions, a classification tree produces a rather noisy and inaccurate rule $\hat{C}(X)$, with error rates around 30%.

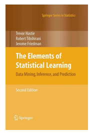
Model Averaging

Classification trees can be simple, but often produce noisy (bushy) or weak (stunted) classifiers.

- *Bagging (Breiman, 1996):* Fit many large trees to bootstrap-resampled versions of the training data, and classify by majority vote.
- *Boosting (Freund & Shapire, 1996):* Fit many large or small trees to *reweighted* versions of the training data. Classify by weighted majority vote.
- *Random Forests (Breiman 1999):* Fancier version of bagging.

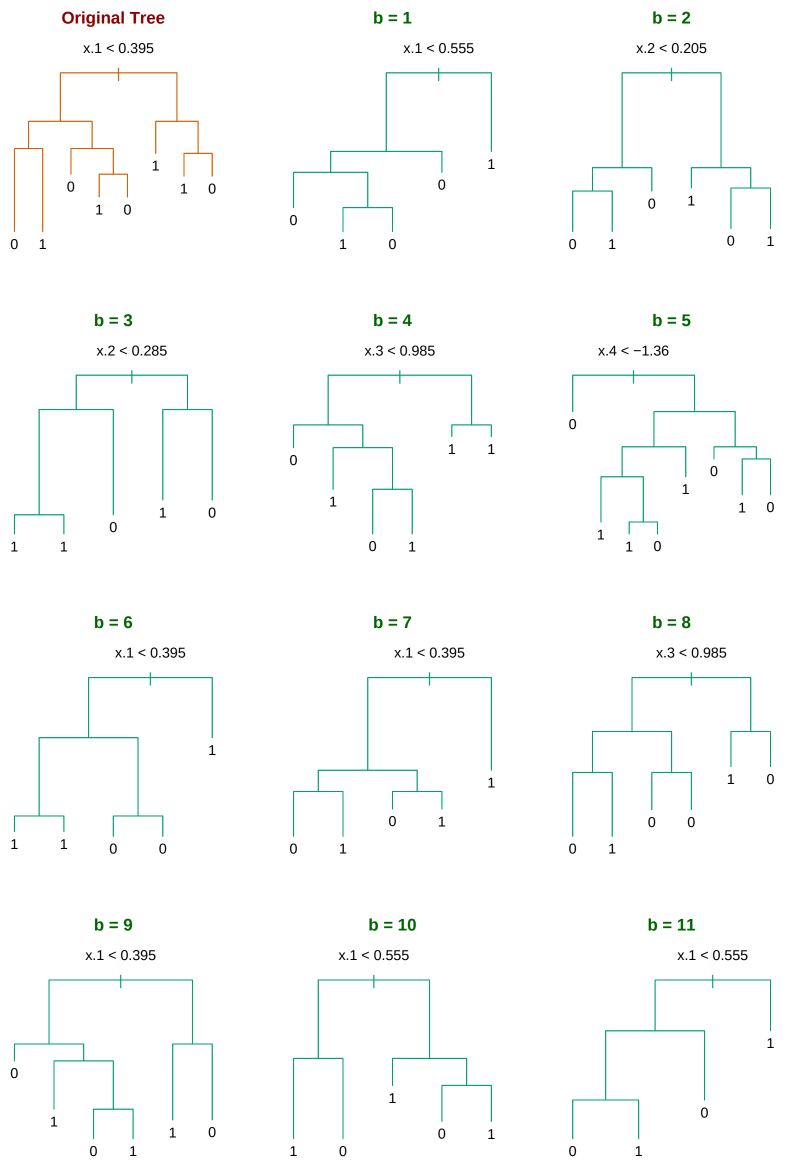
In general *Boosting* $\succ$ *Random Forests* $\succ$ *Bagging* $\succ$ *Single Tree*.

Bagging

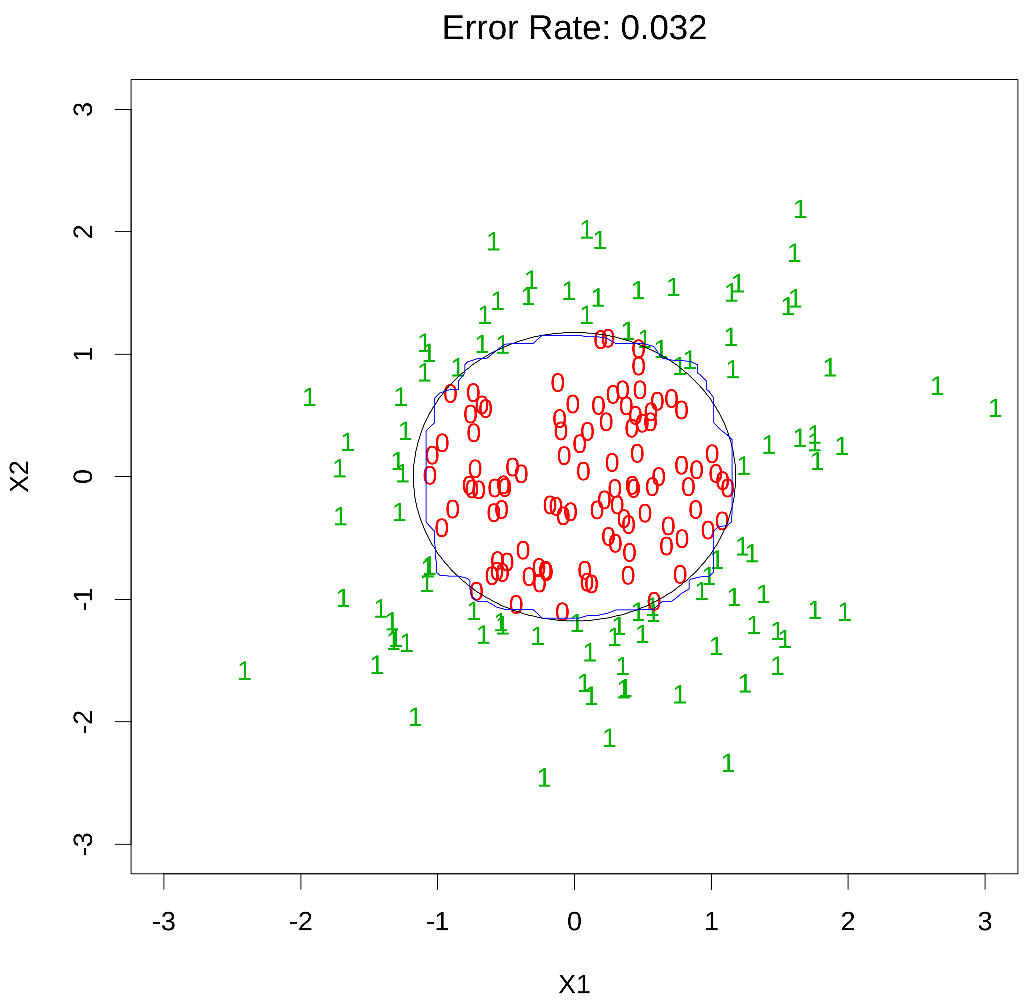
- Bagging or *bootstrap aggregation* averages a noisy fitted function refit to many bootstrap samples, to reduce its variance. Bagging is a poor man's Bayes; See  pp 282.
- Suppose $C(\mathcal{S}, x)$ is a fitted classifier, such as a tree, fit to our training data $\mathcal{S}$, producing a predicted class label at input point x.
- To bag C, we draw bootstrap samples $\mathcal{S}^{*1}, \ldots \mathcal{S}^{*B}$ each of size N with replacement from the training data. Then

$$C_{bag}(x) = \text{Majority Vote } \{C(\mathcal{S}^{*b}, x)\}_{b=1}^{B}.$$

- Bagging can dramatically reduce the variance of unstable procedures (like trees), leading to improved prediction. However any simple structure in C (e.g a tree) is lost.

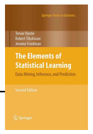

Original Tree
x.1 < 0.395
b = 1
x.1 < 0.555
b = 2
x.2 < 0.205
b = 3
x.2 < 0.285
b = 4
x.3 < 0.985
b = 5
x.4 < −1.36
b = 6
x.1 < 0.395
b = 7
x.1 < 0.395
b = 8
x.3 < 0.985
b = 9
x.1 < 0.395
b = 10
x.1 < 0.555
b = 11
x.1 < 0.555

Decision Boundary: Bagging

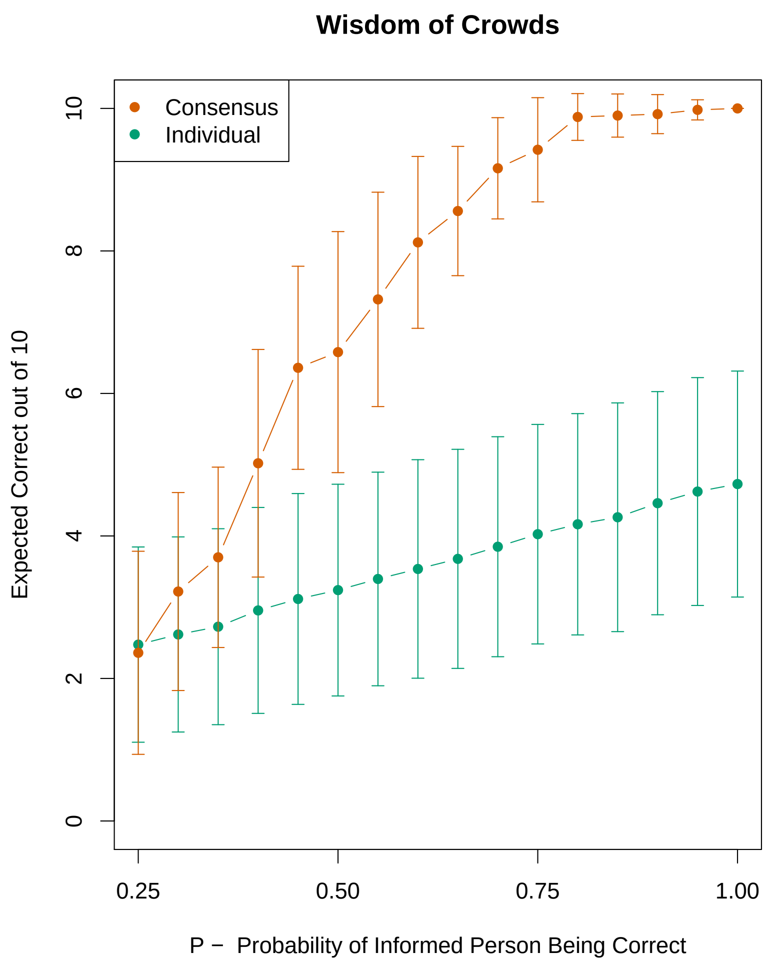


- Bagging averages many trees, and produces *smoother* decision boundaries.
- Bagging reduces error by variance reduction.
- Bias is slightly increased because bagged trees are slightly shallower.

Random forests

- refinement of bagged trees; very popular—  chapter 15.
- at each tree split, a random sample of m features is drawn, and only those m features are considered for splitting. Typically $m = \sqrt{p}$ or $\log_2 p$, where p is the number of features
- For each tree grown on a bootstrap sample, the error rate for observations left out of the bootstrap sample is monitored. This is called the "out-of-bag" or OOB error rate.
- random forests tries to improve on bagging by "de-correlating" the trees, and reduce the variance.
- Each tree has the same expectation, so increasing the number of trees does not alter the bias of bagging or random forests.

Wisdom of Crowds

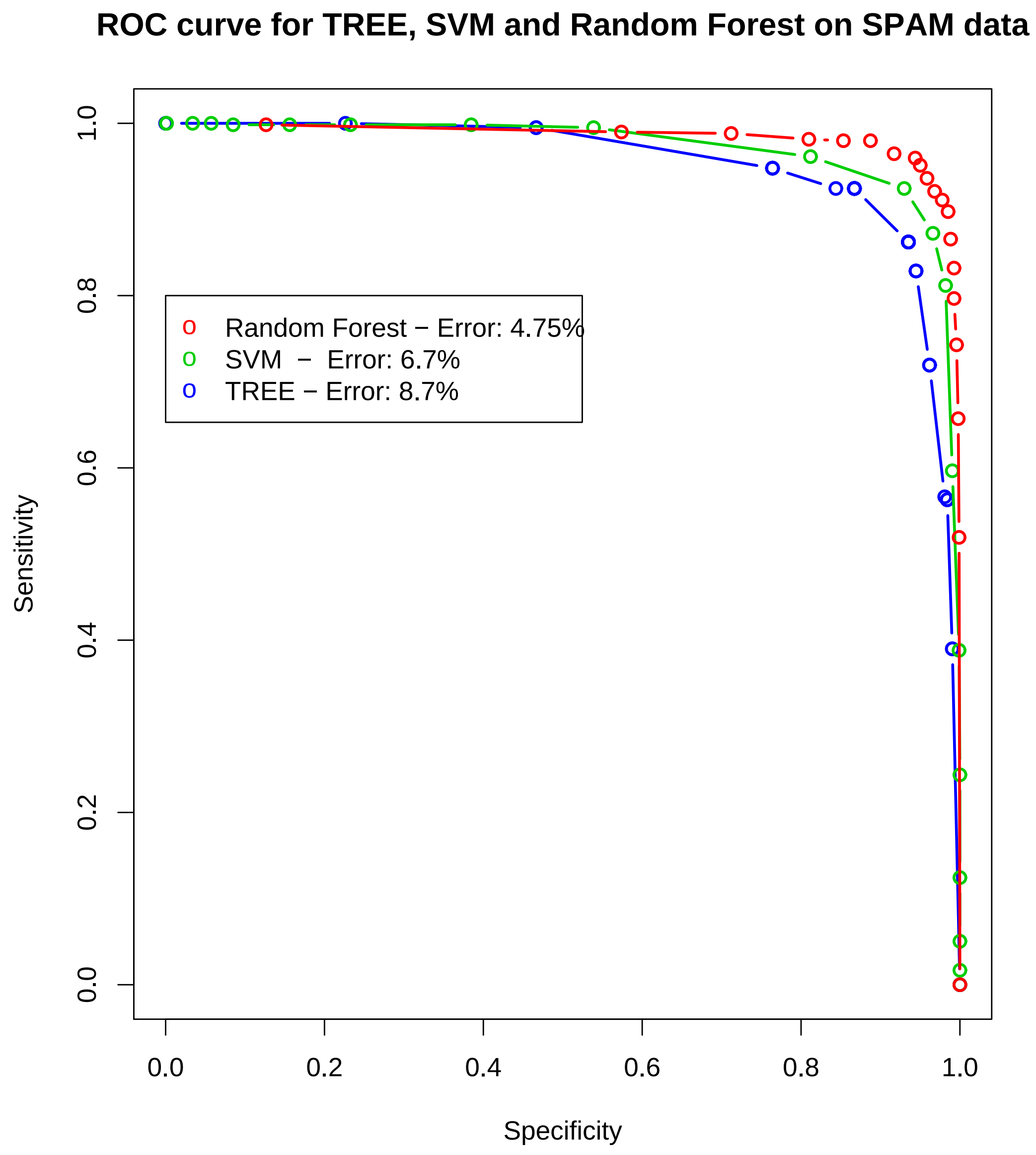


- 10 movie categories, 4 choices in each category
- 50 people, for each question 15 *informed* and 35 *guessers*
- For informed people,

$$\begin{aligned} \Pr(\text{correct}) &= p \\ \Pr(\text{incorrect}) &= (1-p)/3 \end{aligned}$$

- For guessers, $\Pr(\text{all choices}) = 1/4$.

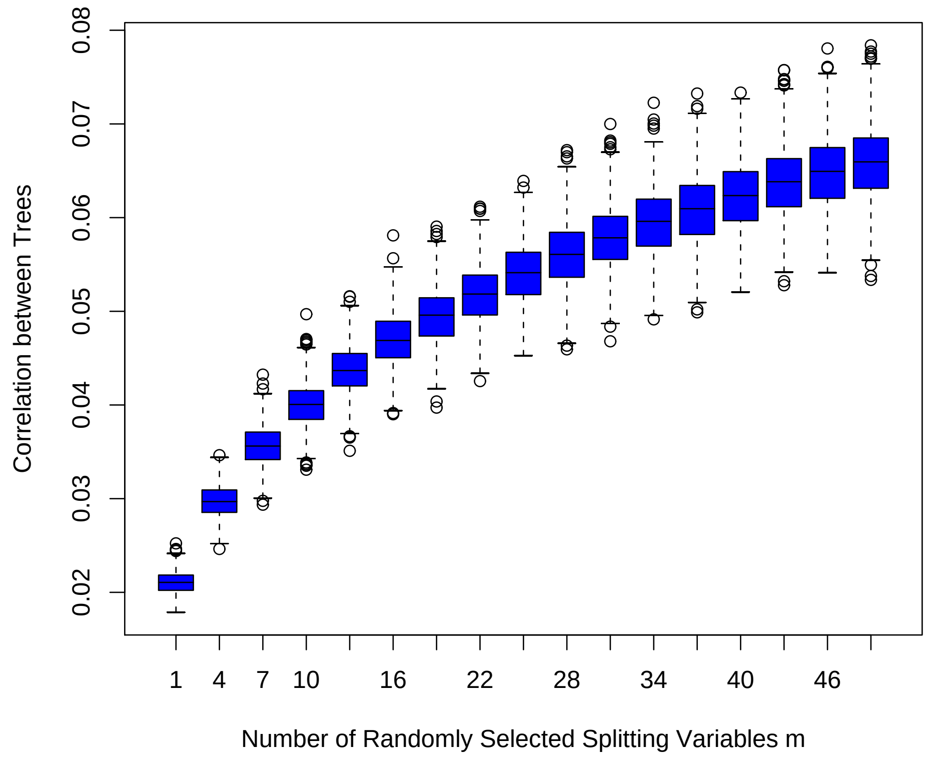
Consensus (orange) improve over individuals (teal) even for moderate p.



TREE, SVM and RF

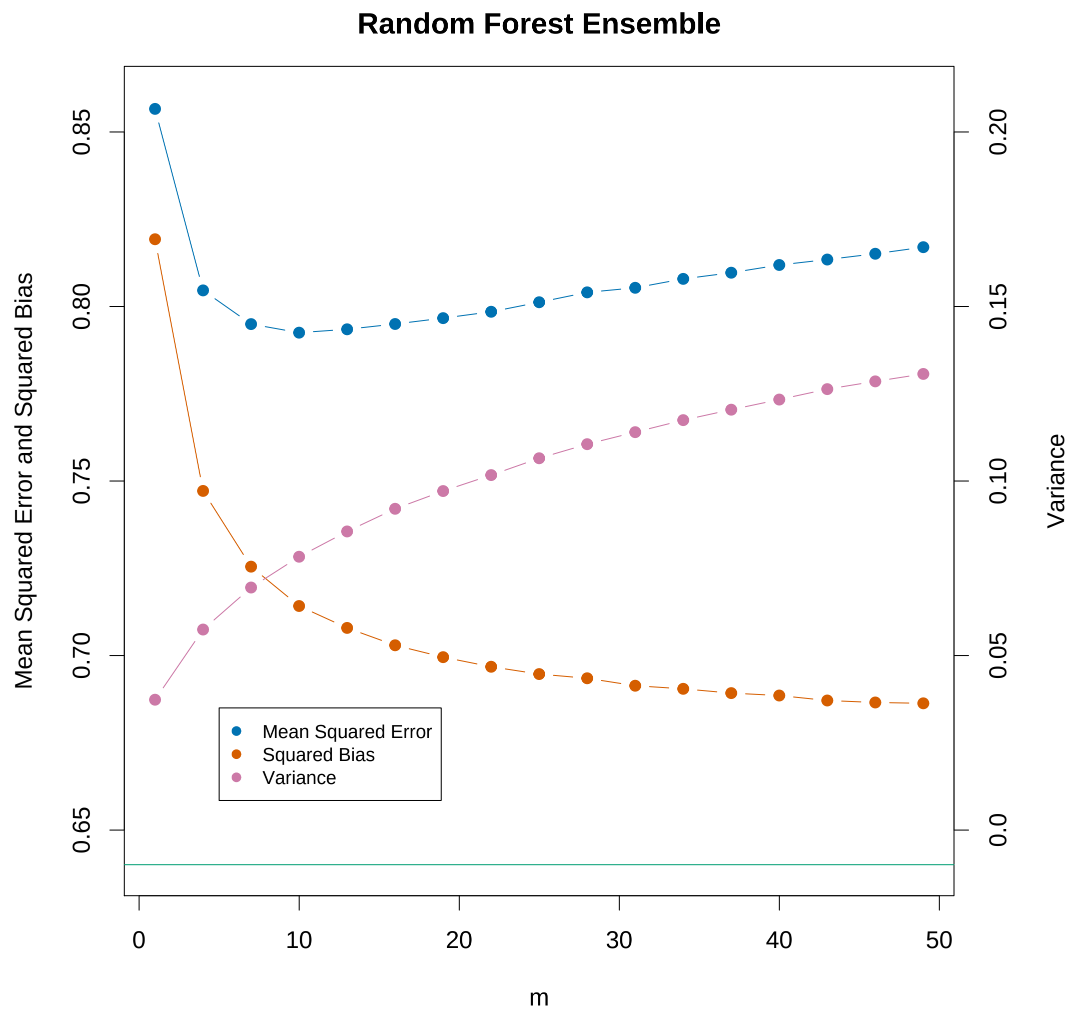
Random Forest dominates both other methods on the SPAM data — 4.75% error. Used 500 trees with default settings for `random Forest` package in `R`.

RF: decorrelation



- $\text{Var}\hat{f}_{RF}(x) = \rho(x)\sigma^2(x)$
- The correlation is because we are growing trees to the *same data*.
- The correlation decreases as m decreases—m is the number of variables sampled at each split.

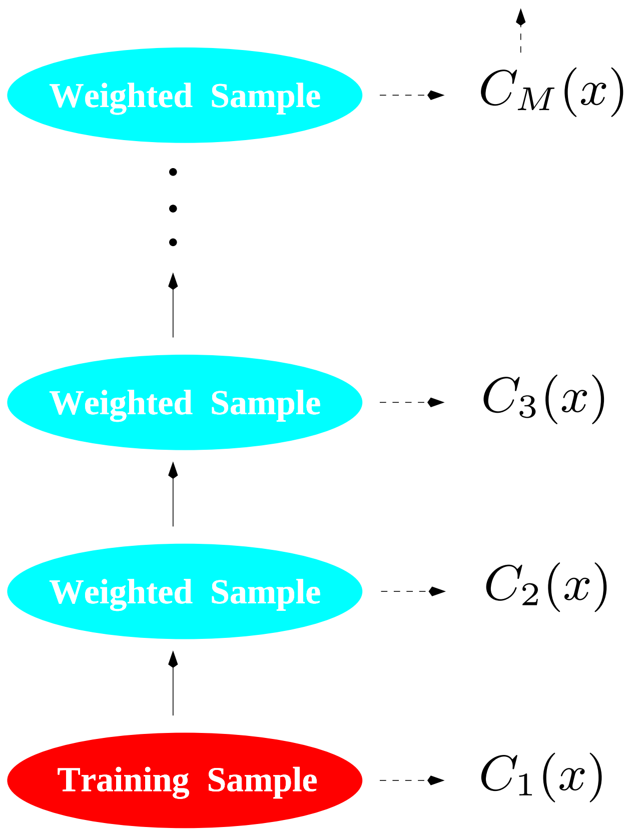
Based on a simple experiment with random forest regression models.



Bias & Variance

- The smaller m, the lower the variance of the random forest ensemble
- Small m is also associated with higher bias, because important variables can be missed by the sampling.

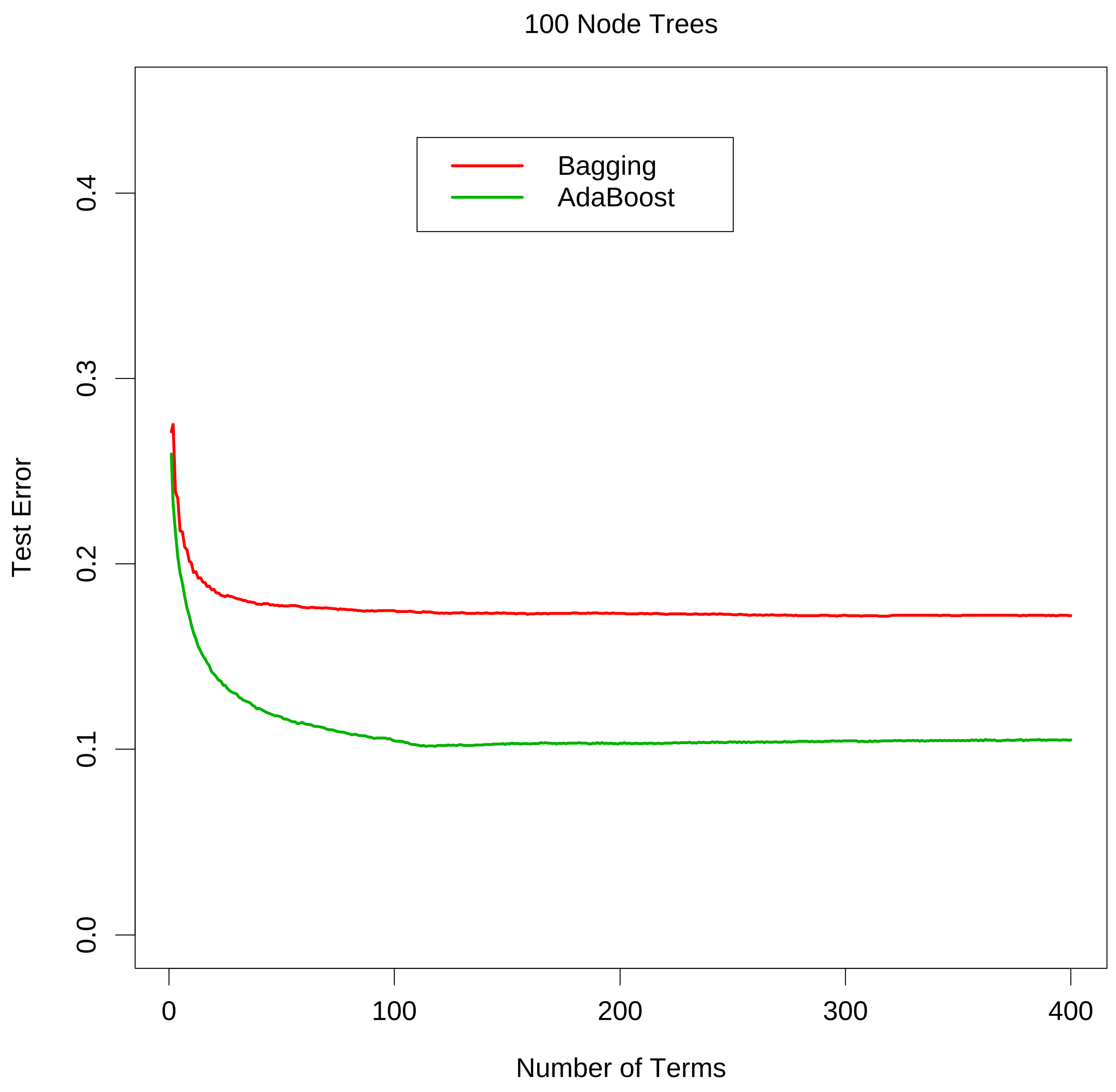
Boosting



- Breakthrough invention of Freund & Schapire (1997)
- Average many trees, each grown to re-weighted versions of the training data.
- Weighting *decorrelates* the trees, by focussing on regions missed by past trees.
- Final Classifier is weighted average of classifiers:

$$C(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m C_m(x)\right]$$

Note: $C_m(x) \in \{-1, +1\}$.

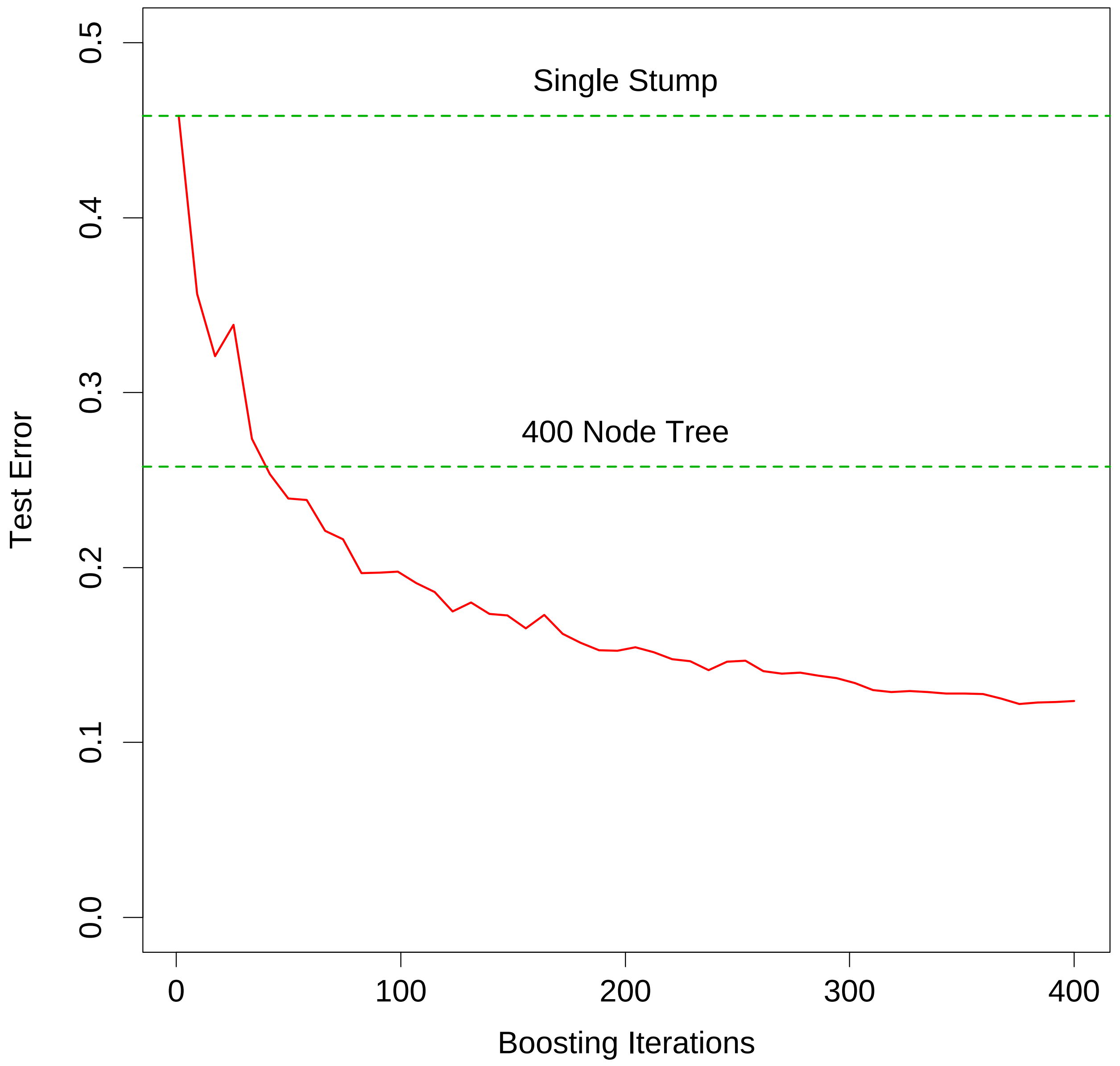


Boosting vs Bagging

- 2000 points from Nested Spheres in R^{10}
- Bayes error rate is 0%.
- Trees are grown *best first* without pruning.
- Leftmost term is a single tree.

AdaBoost (Freund & Schapire, 1996)

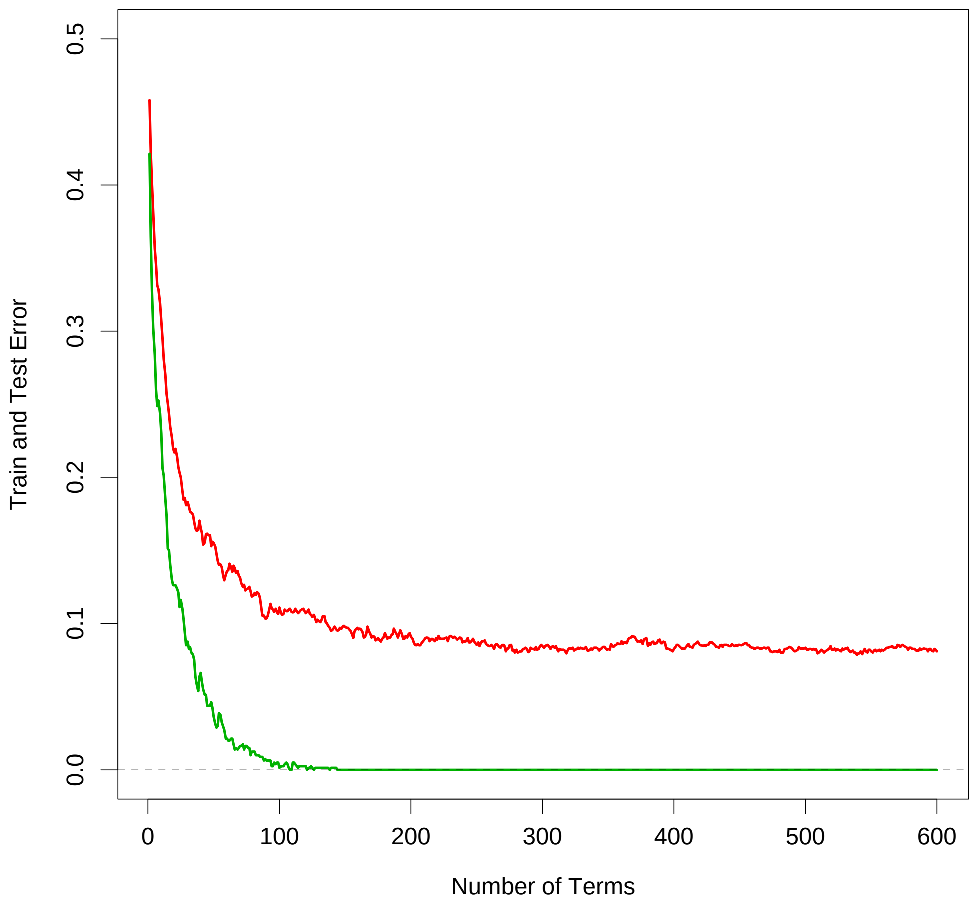
1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M repeat steps (a)–(d):
   (a) Fit a classifier $C_m(x)$ to the training data using weights w_i.
   (b) Compute weighted error of newest tree
   $$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq C_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log[(1 - \mathrm{err}_m)/\mathrm{err}_m]$.
   (d) Update weights for $i = 1, \ldots, N$:
   $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq C_m(x_i))]$
   and renormalize to w_i to sum to 1.
3. Output $C(x) = \mathrm{sign}\left[\sum_{m=1}^{M} \alpha_m C_m(x)\right]$.



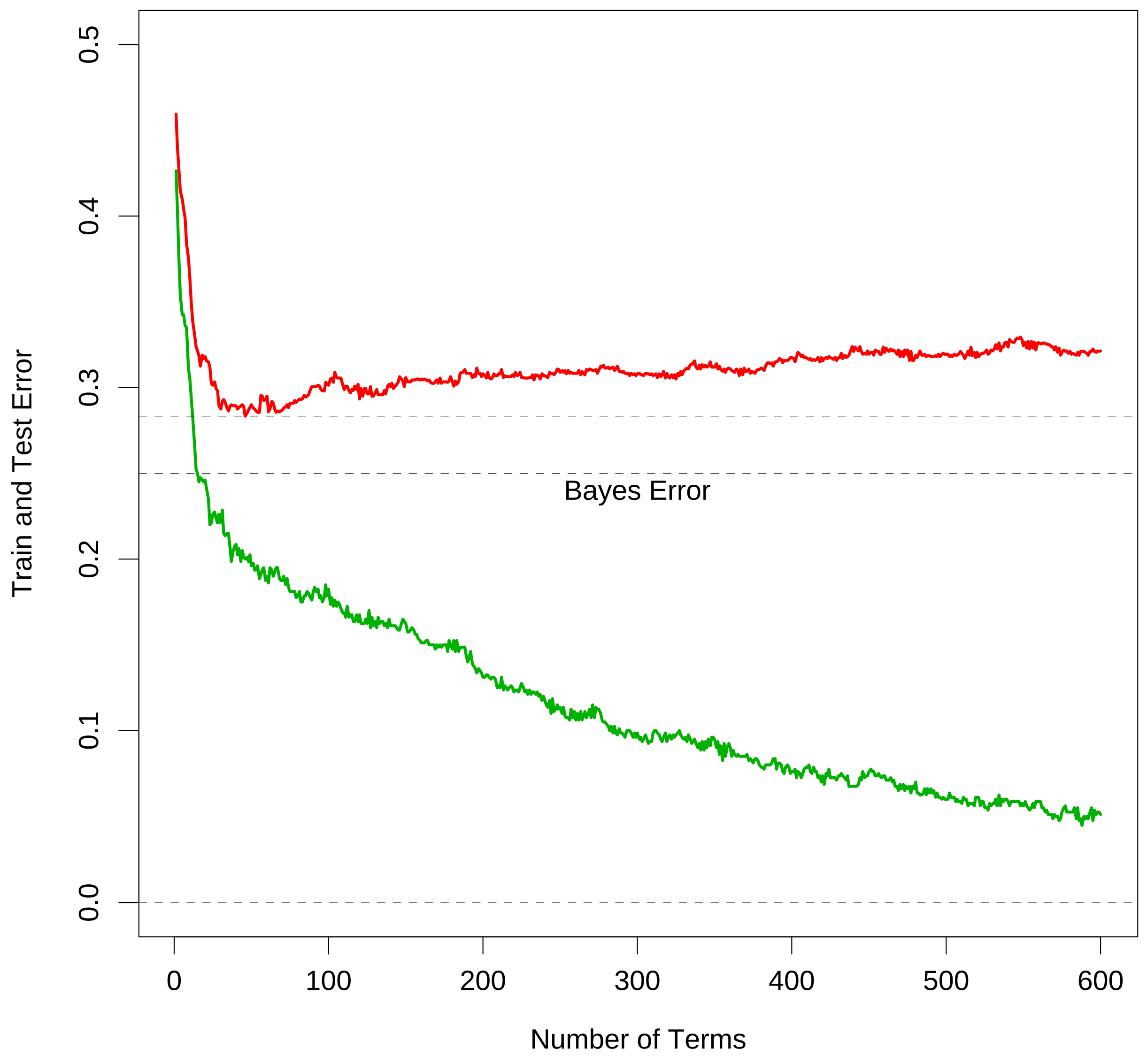
Boosting Stumps

- A stump is a two-node tree, after a single split.
- Boosting stumps works remarkably well on the nested-spheres problem.

Training Error



- Nested spheres in 10-Dimensions.
- Bayes error is 0%.
- Boosting drives the training error to zero (green curve).
- Further iterations continue to improve test error in many examples (red curve).



Noisy Problems

- Nested Gaussians in 10-Dimensions.
- Bayes error is 25%.
- Boosting with stumps
- Here the test error does increase, but quite slowly.

Stagewise Additive Modeling

Boosting builds an additive model

$$F(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m).$$

Here $b(x, \gamma_m)$ is a tree, and γ_m parametrizes the splits.

We do things like that in statistics all the time!

- GAMs: $F(x) = \sum_j f_j(x_j)$
- Basis expansions: $F(x) = \sum_{m=1}^{M} \theta_m h_m(x)$

Traditionally the parameters f_m, θ_m are fit *jointly* (i.e. least squares, maximum likelihood).

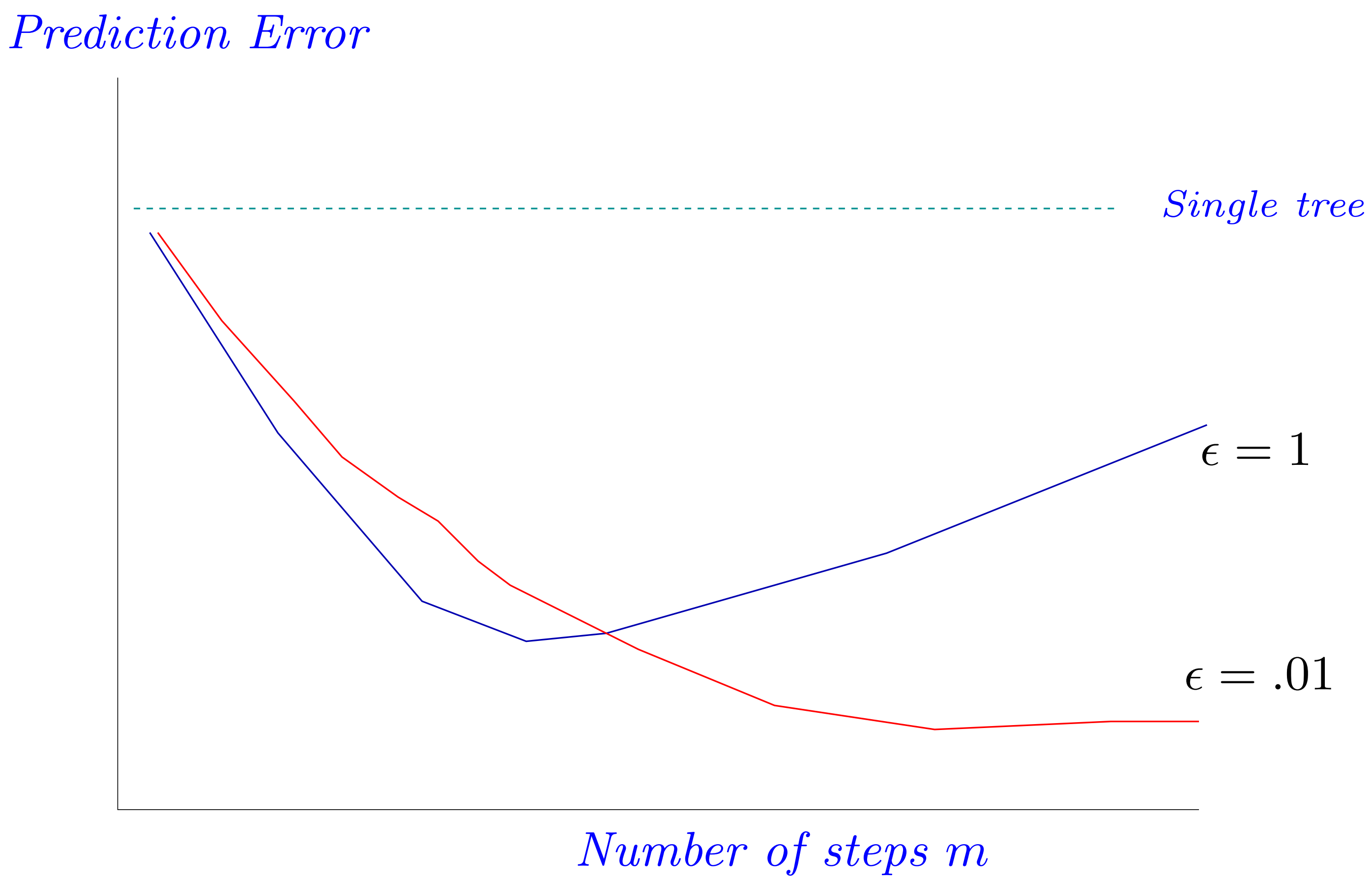
With boosting, the parameters (β_m, γ_m) are fit in a *stagewise* fashion. This slows the process down, and overfits less quickly.

Example: Least Squares Boosting

1. Start with function $F_0(x) = 0$ and residual $r = y$, $m = 0$
2. $m \leftarrow m + 1$
3. Fit a CART regression tree to r giving $g(x)$
4. Set $f_m(x) \leftarrow \epsilon \cdot g(x)$
5. Set $F_m(x) \leftarrow F_{m-1}(x) + f_m(x)$, $r \leftarrow r - f_m(x)$ and repeat step 2–5 many times

- $g(x)$ typically shallow tree; e.g. constrained to have $k = 3$ splits only (depth).
- Stagewise fitting (no backfitting!). Slows down (over-) fitting process.
- $0 < \epsilon \leq 1$ is shrinkage parameter; slows overfitting even further.

Least Squares Boosting and ϵ



- If instead of trees, we use linear regression terms, boosting with small ϵ very similar to lasso!

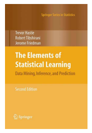
Adaboost: Stagewise Modeling

- AdaBoost builds an *additive logistic regression model*

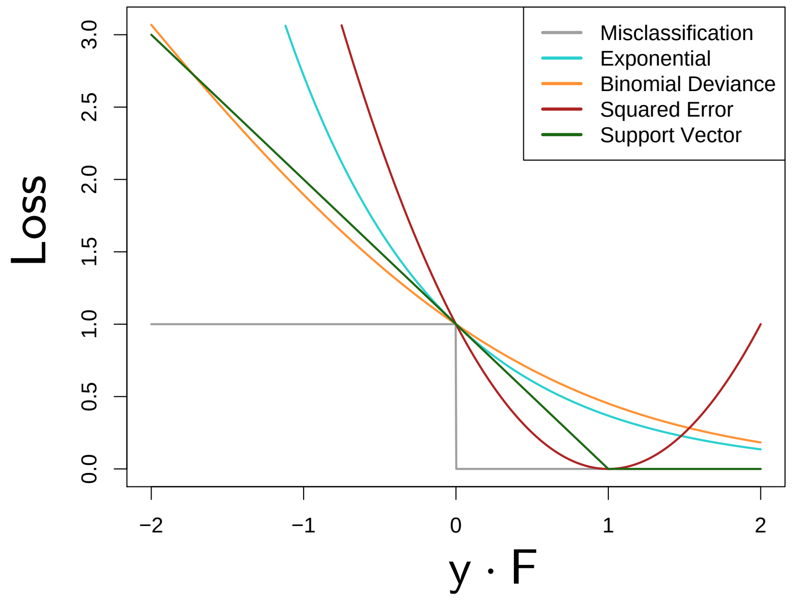
$$F(x) = \log \frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)} = \sum_{m=1}^{M} \alpha_m f_m(x)$$

  by *stagewise fitting* using the loss function

$$L(y, F(x)) = \exp(-y\,F(x)).$$

- Instead of fitting trees to residuals, the special form of the exponential loss leads to fitting trees to weighted versions of the original data. See  pp 343.

Why Exponential Loss?



- $y \cdot F$ is the *margin*; positive margin means correct classification.

- $e^{-yF(x)}$ is a monotone, smooth upper bound on misclassification loss at x.
- Leads to simple reweighting scheme.
- Has *logit* transform as population minimizer

$$F^*(x) = \frac{1}{2}\log\frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)}$$

- Other more robust loss functions, like *binomial deviance.*

General Boosting Algorithms

The boosting paradigm can be extended to general loss functions — not only *squared error* or *exponential*.

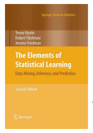
1. Initialize $F_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute
   $(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \beta b(x_i; \gamma))$.
   (b) Set $F_m(x) = F_{m-1}(x) + \beta_m b(x; \gamma_m)$.

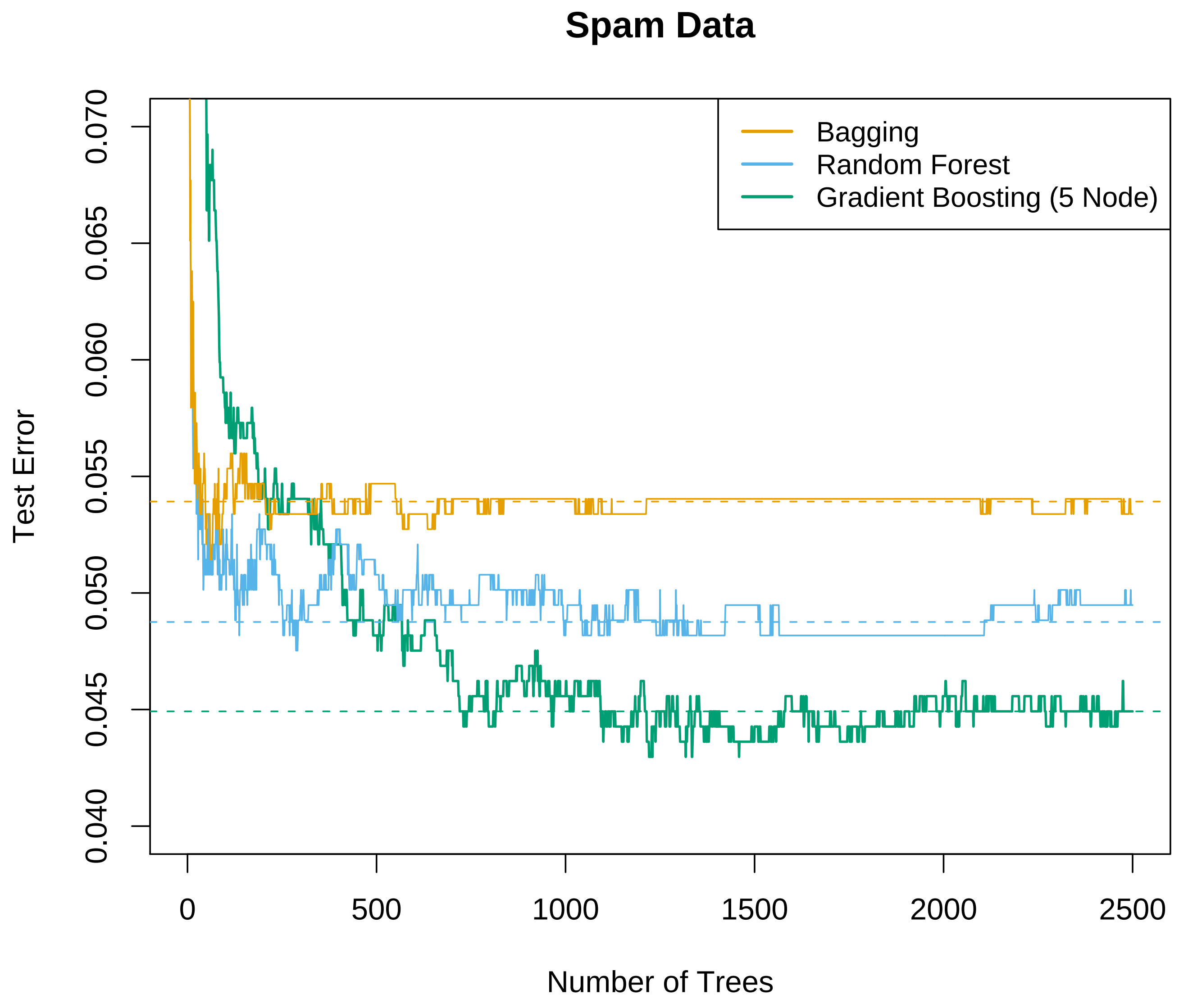
Sometimes we replace step (b) in item 2 by

(b*) Set $F_m(x) = F_{m-1}(x) + \epsilon\beta_m b(x; \gamma_m)$

Here ϵ is a *shrinkage factor*; in `gbm` package in R, the default is $\epsilon = 0.001$. Shrinkage slows the stagewise model-building even more, and typically leads to better performance.

Gradient Boosting

- General boosting algorithm that works with a variety of different loss functions. Models include regression, resistant regression, K-class classification and risk modeling.
- Gradient Boosting builds additive tree models, for example, for representing the logits in logistic regression.
- *Tree size* is a parameter that determines the order of interaction (next slide).
- Gradient Boosting inherits all the good features of trees (variable selection, missing data, mixed predictors), and improves on the weak features, such as prediction performance.
- Gradient Boosting is described in detail in , section 10.10, and is implemented in Greg Ridgeways `gbm` package in R.


Spam Data
Bagging
Random Forest
Gradient Boosting (5 Node)
Test Error
0.040
0.045
0.050
0.055
0.060
0.065
0.070
Number of Trees
0
500
1000
1500
2000
2500

Spam Example Results

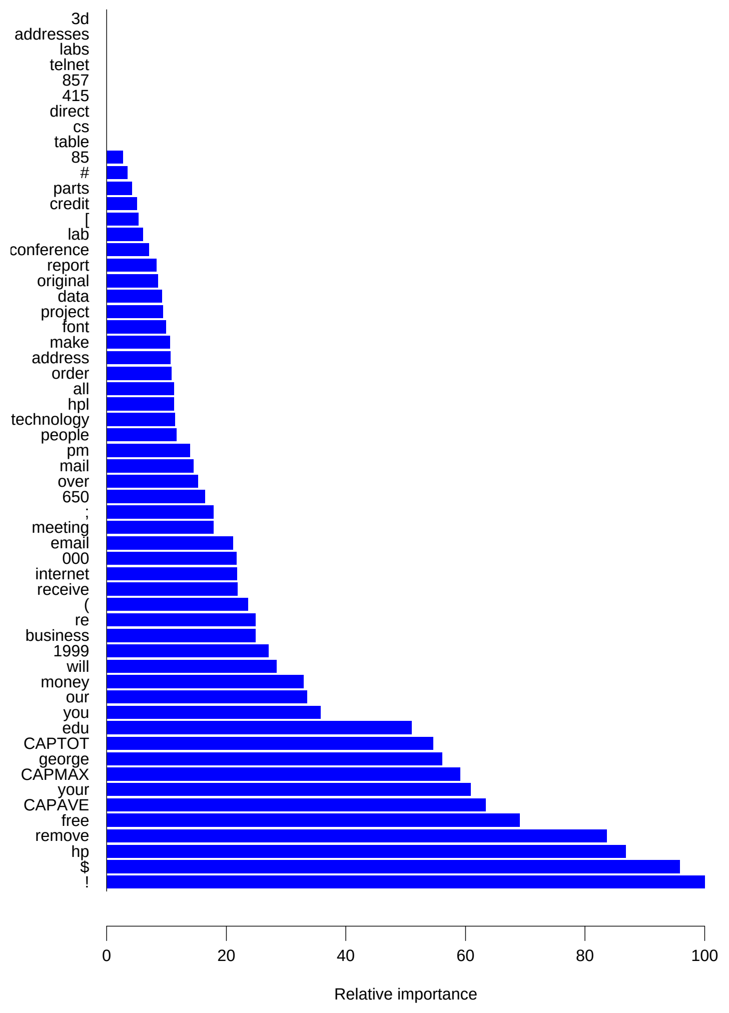
With 3000 training and 1500 test observations, Gradient Boosting fits an additive logistic model

$$f(x) = \log \frac{\Pr(\texttt{spam}|x)}{\Pr(\texttt{email}|x)}$$

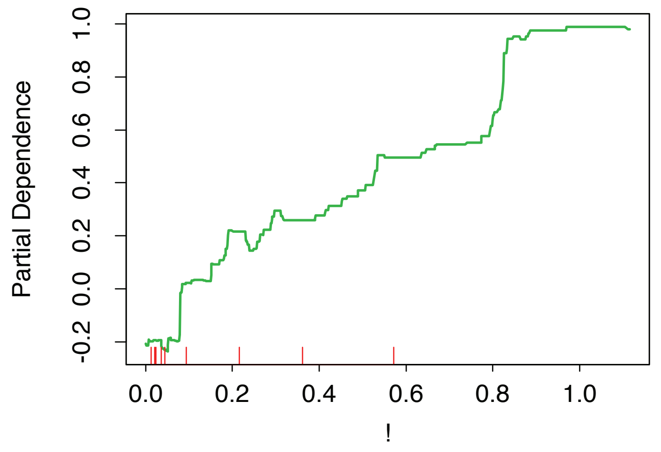
using trees with $J = 6$ terminal-node trees.

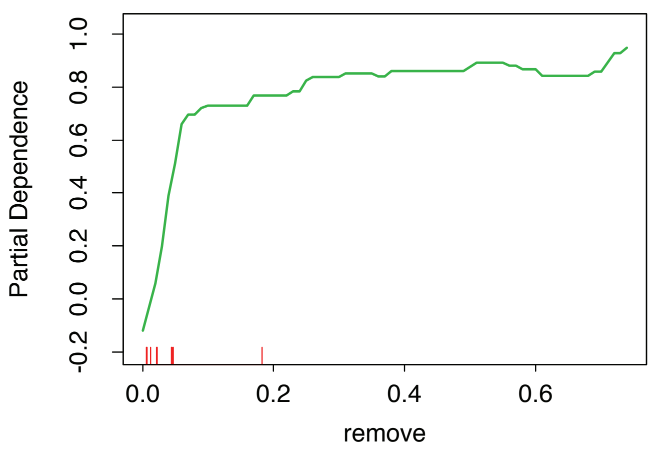
Gradient Boosting achieves a test error of 4.5%, compared to 5.3% for an additive GAM, 4.75% for Random Forests, and 8.7% for CART.

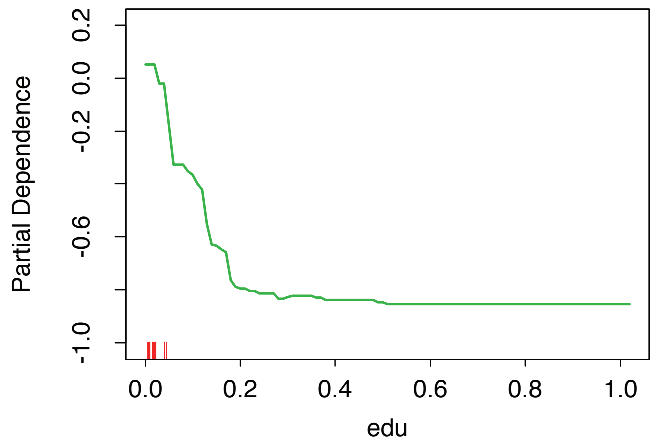
Spam: Variable Importance

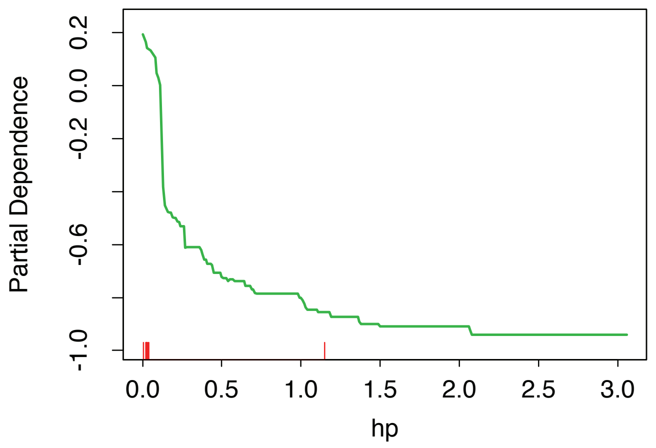


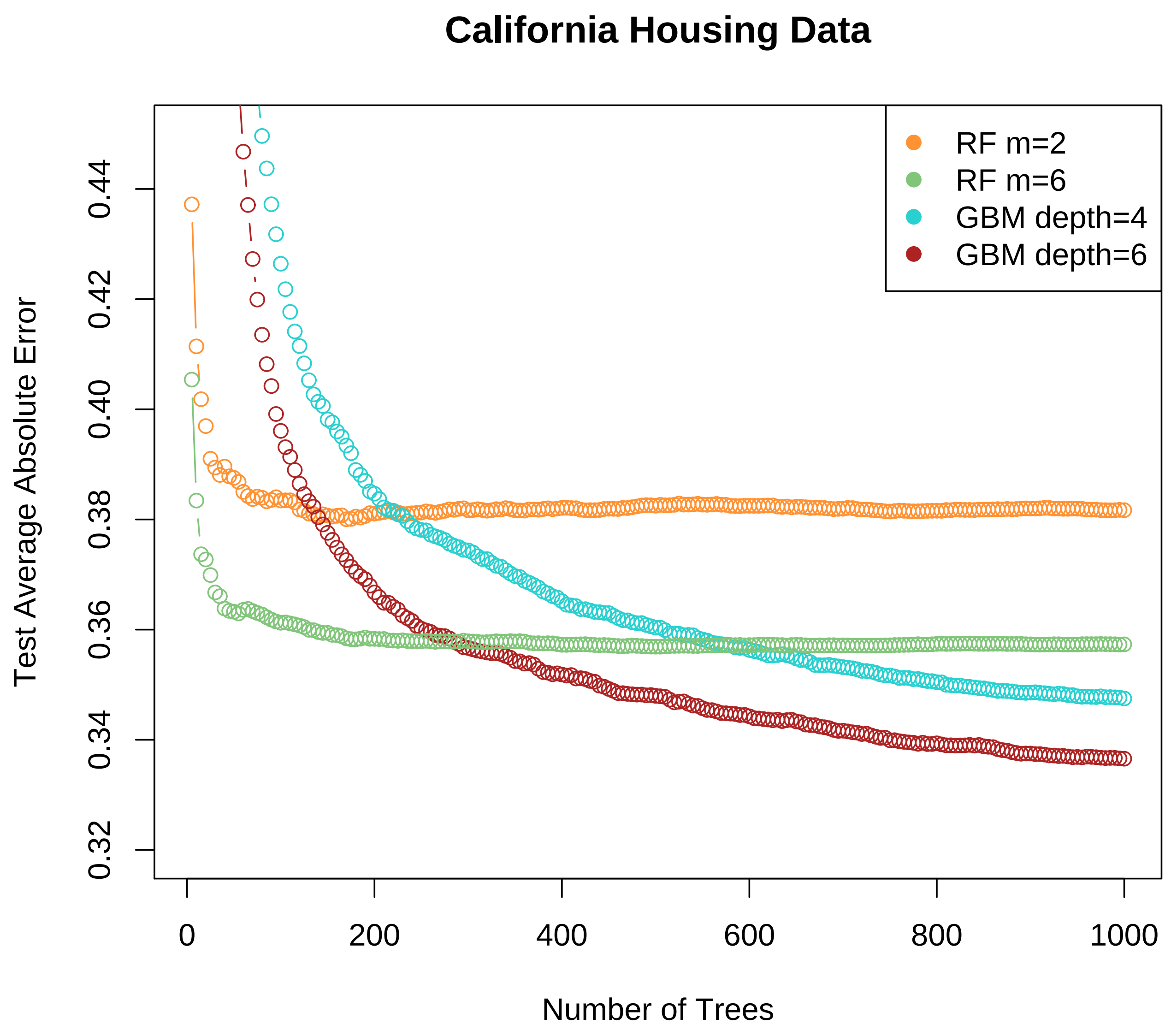
Spam: Partial Dependence

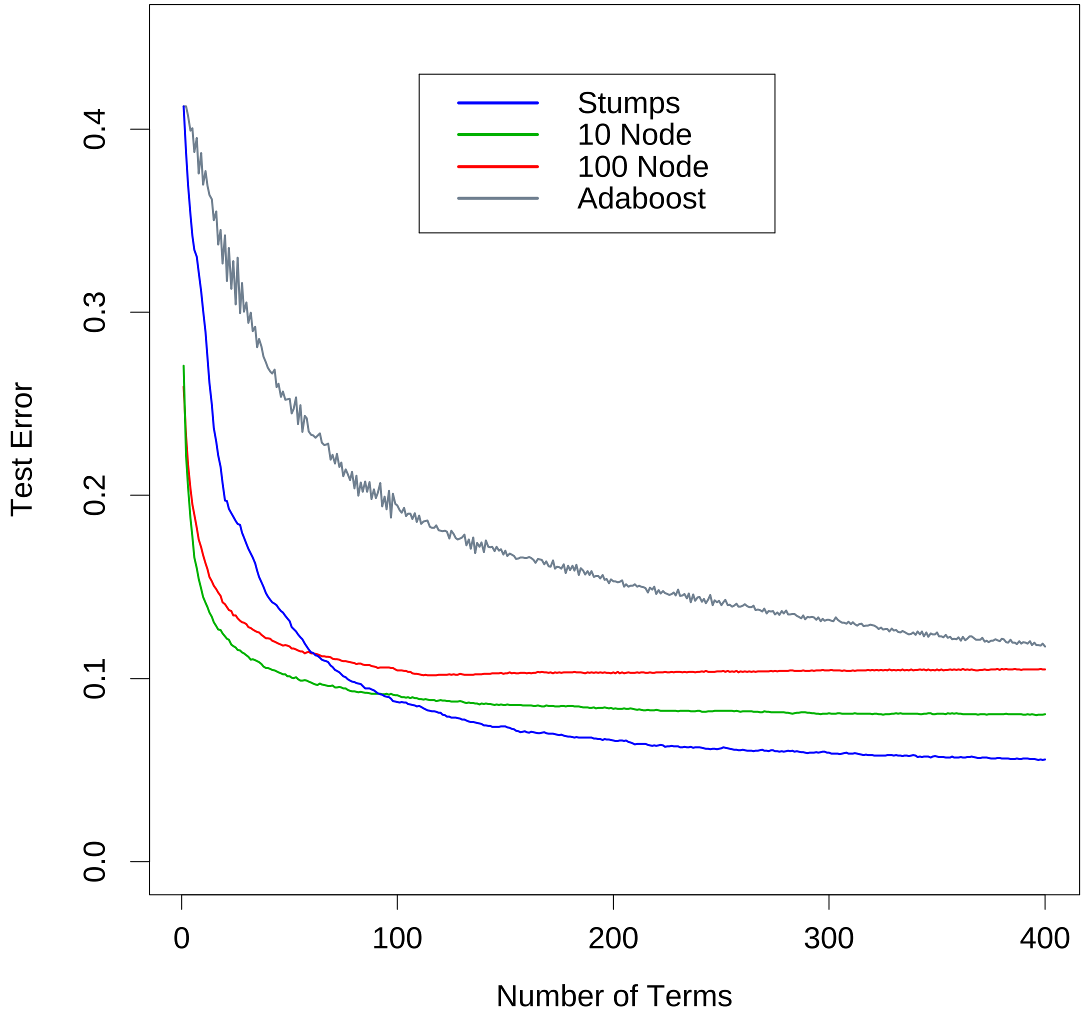









California Housing Data
Test Average Absolute Error
Number of Trees
0.32
0.34
0.36
0.38
0.40
0.42
0.44
0
200
400
600
800
1000
● RF m=2
● RF m=6
● GBM depth=4
● GBM depth=6



Tree Size

The tree size J determines the interaction order of the model:

$$\eta(X) = \sum_j \eta_j(X_j) + \sum_{jk} \eta_{jk}(X_j, X_k) + \sum_{jkl} \eta_{jkl}(X_j, X_k, X_l) + \cdots$$

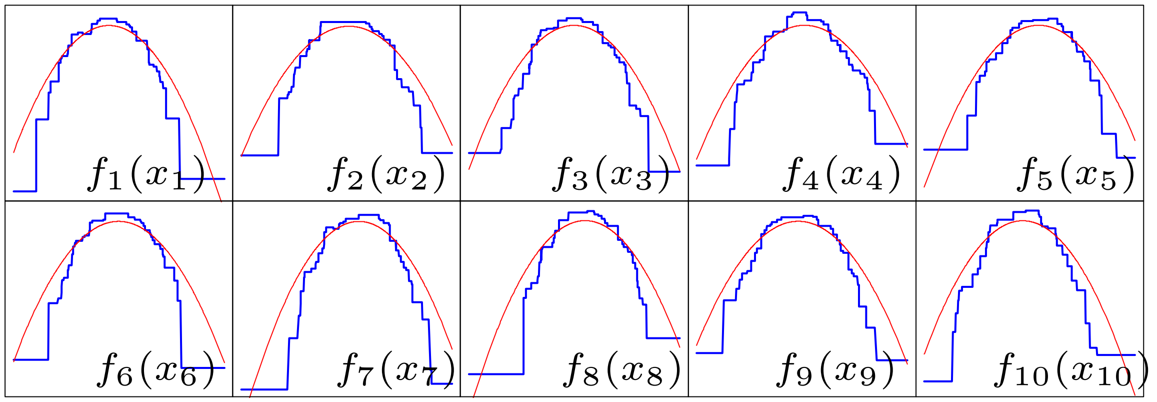
Stumps win!

Since the true decision boundary is the surface of a sphere, the function that describes it has the form

$$f(X) = X_1^2 + X_2^2 + \ldots + X_p^2 - c = 0.$$

Boosted stumps via Gradient Boosting returns reasonable approximations to these quadratic functions.

Coordinate Functions for Additive Logistic Trees



Learning Ensembles

Ensemble Learning consists of two steps:

1. Construction of a *dictionary* $\mathcal{D} = \{T_1(X), T_2(X), \ldots, T_M(X)\}$ of basis elements (weak learners) $T_m(X)$.
2. Fitting a model $f(X) = \sum_{m \in \mathcal{D}} \alpha_m T_m(X)$.

Simple examples of ensembles

- Linear regression: The ensemble consists of the coordinate functions $T_m(X) = X_m$. The fitting is done by least squares.
- *Random Forests:* The ensemble consists of trees grown to booststrapped versions of the data, with additional randomization at each split. The fitting simply averages.
- *Gradient Boosting:* The ensemble is grown in an adaptive fashion, but then simply averaged at the end.

Learning Ensembles and the Lasso

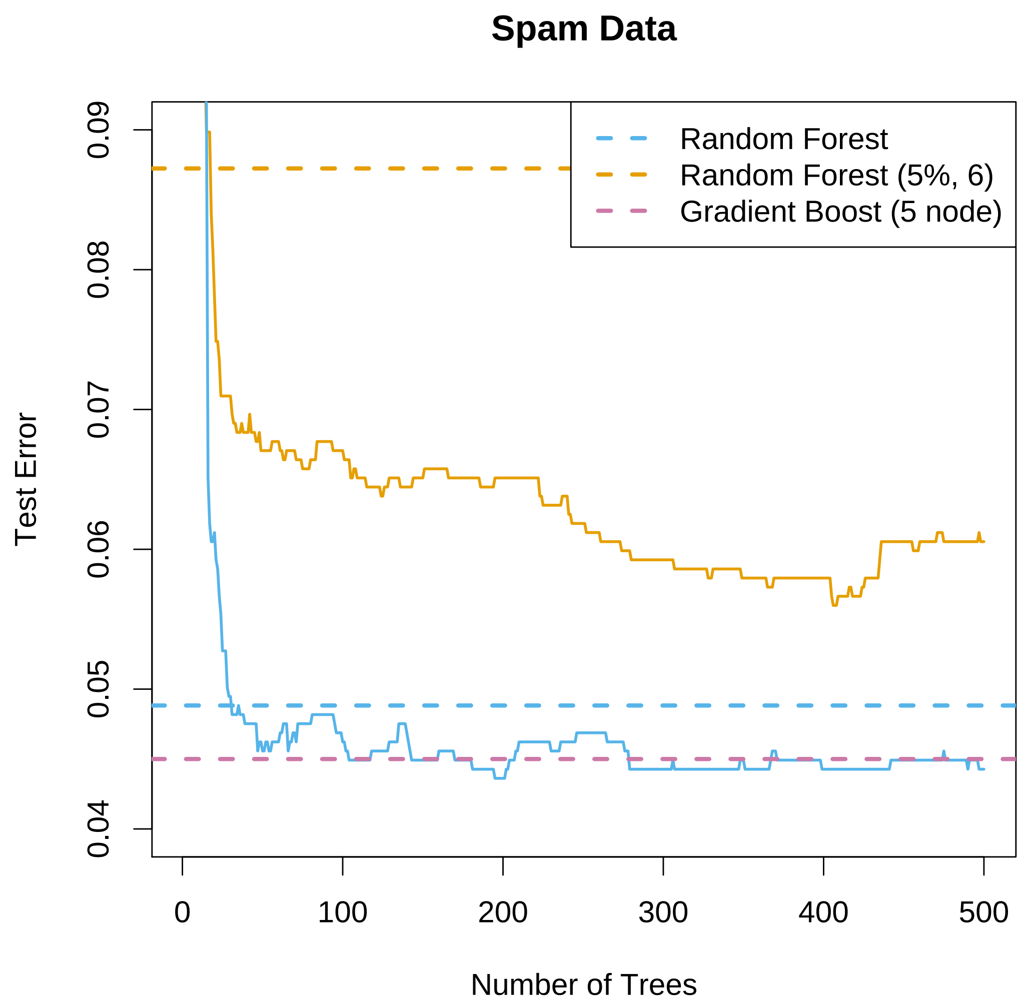
Proposed by Popescu and Friedman (2004):
Fit the model $f(X) = \sum_{m \in \mathcal{D}} \alpha_m T_m(X)$ in step (2) using Lasso regularization:

$$\alpha(\lambda) = \arg\min_{\alpha} \sum_{i=1}^{N} L[y_i, \alpha_0 + \sum_{m=1}^{M} \alpha_m T_m(x_i)] + \lambda \sum_{m=1}^{M} |\alpha_m|.$$

Advantages:

- Often ensembles are large, involving thousands of small trees. The post-processing selects a smaller subset of these and combines them efficiently.
- Often the post-processing improves the process that generated the ensemble.

Post-Processing Random Forest Ensembles



Software

- *R*: free GPL statistical computing environment available from *CRAN*, implements the *S* language. Includes:
  - *randomForest*: implementation of Leo Breimans algorithms.
  - *rpart*: Terry Therneau's implementation of classification and regression trees.
  - *gbm*: Greg Ridgeway's implementation of Friedman's gradient boosting algorithm.
- *Salford Systems*: Commercial implementation of trees, random forests and gradient boosting.
- *Splus (Insightful)*: Commerical version of S.
- *Weka*: GPL software from University of Waikato, New Zealand. Includes Trees, Random Forests and many other procedures.