Trees, Bagging, Random Forests and Boosting

- Classification Trees

- Bagging: Averaging Trees

- Random Forests: Cleverer Averaging of Trees

- Boosting: Cleverest Averaging of Trees

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

Two-class Classification

- Observations are classified into two or more classes, coded by a response variable Y taking values $1, 2, \ldots, K$.

- We have a feature vector $X = (X_1, X_2, \ldots, X_p)$, and we hope to build a classification rule $C(X)$ to assign a class label to an individual with feature X.

- We have a sample of pairs (y_i, x_i), $i = 1, \ldots, N$. Note that each of the x_i are vectors $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$.

- Example: Y indicates whether an email is spam or not. X represents the relative frequency of a subset of specially chosen words in the email message.

- The technology described here estimates $C(X)$ directly, or via the probability function $P(C = k|X)$.

Classification Trees

- Represented by a series of binary splits.

- Each internal node represents a value query on one of the variables — e.g. "Is $X_3 > 0.4$". If the answer is "Yes", go right, else go left.

- The terminal nodes are the decision nodes. Typically each terminal node is dominated by one of the classes.

- The tree is grown using training data, by recursive splitting.

- The tree is often pruned to an optimal size, evaluated by cross-validation.

- New observations are classified by passing their X down to a terminal node of the tree, and then using majority vote.

Classification Tree
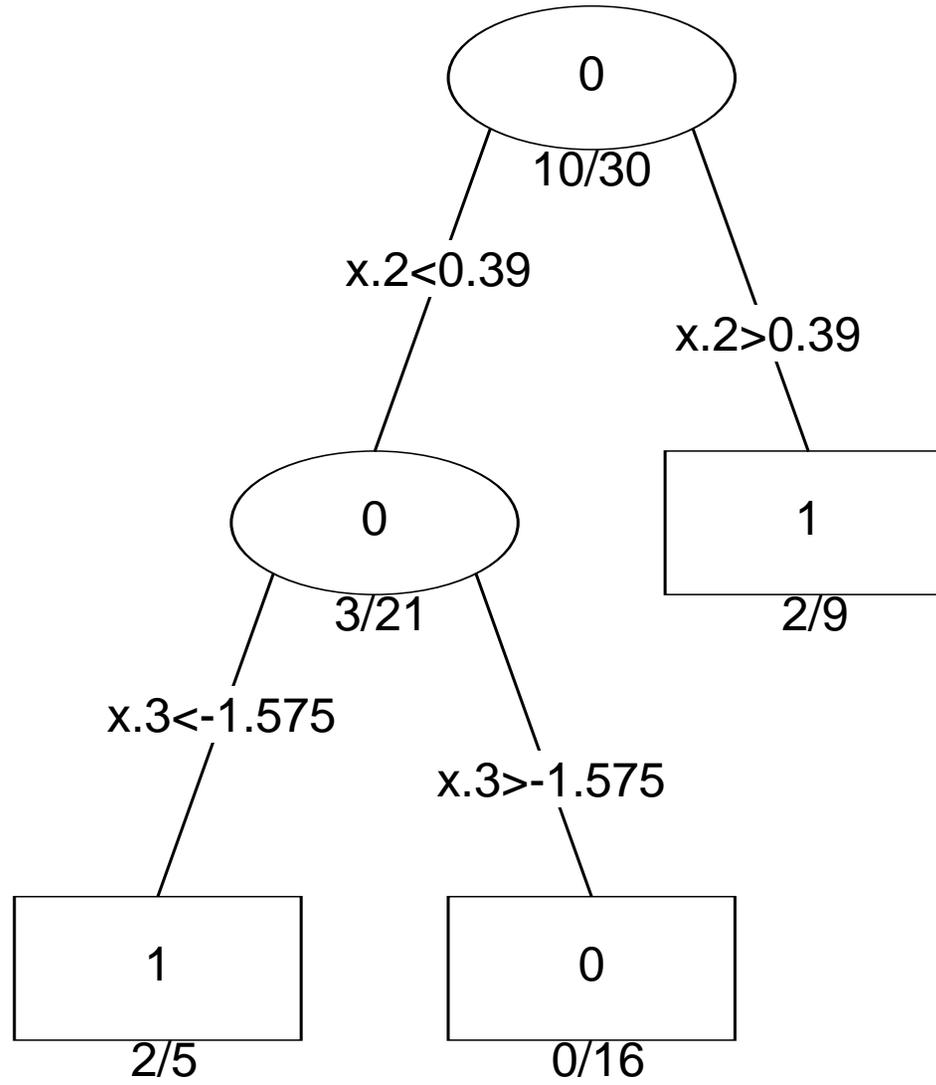
Properties of Trees

✔ Can handle huge datasets

✔ Can handle mixed predictors—quantitative and qualitative

✔ Easily ignore redundant variables

✔ Handle missing data elegantly

✔ Small trees are easy to interpret

✘ large trees are hard to interpret

✘ Often prediction performance is poor

Example: Predicting e-mail spam

- data from 4601 email messages

- goal: predict whether an email message is spam (junk email) or good.

- input features: relative frequencies in a message of 57 of the most commonly occurring words and punctuation marks in all the training the email messages.

- for this problem not all errors are equal; we want to avoid filtering out good email, while letting spam get through is not desirable but less serious in its consequences.

- we coded `spam` as 1 and `email` as 0.

- A system like this would be trained for each user separately (e.g. their word lists would be different)

Predictors

- 48 quantitative predictors—the percentage of words in the email that match a given word. Examples include `business`, `address`, `internet`, `free`, and `george`. The idea was that these could be customized for individual users.

- 6 quantitative predictors—the percentage of characters in the email that match a given character. The characters are `ch;`, `ch(`, `ch[`, `ch!`, `ch$`, and `ch#`.

- The average length of uninterrupted sequences of capital letters: `CAPAVE`.

- The length of the longest uninterrupted sequence of capital letters: `CAPMAX`.

- The sum of the length of uninterrupted sequences of capital letters: `CAPTOT`.

Details

- A test set of size 1536 was randomly chosen, leaving 3065 observations in the training set.

- A full tree was grown on the training set, with splitting continuing until a minimum bucket size of 5 was reached.

- This bushy tree was pruned back using cost-complexity pruning, and the tree size was chosen by 10-fold cross-validation.

- We then compute the test error and ROC curve on the test data.

Some important features

39% of the training data were spam.

Average percentage of words or characters in an email message equal to the indicated word or character. We have chosen the words and characters showing the largest difference between `spam` and `email`.

	george	you	your	hp	free	hpl
spam	0.00	2.26	1.38	0.02	0.52	0.01
email	1.27	1.27	0.44	0.90	0.07	0.43

	!	our	re	edu	remove
spam	0.51	0.51	0.13	0.01	0.28
email	0.11	0.18	0.42	0.29	0.01

ROC curve for pruned tree on SPAM data
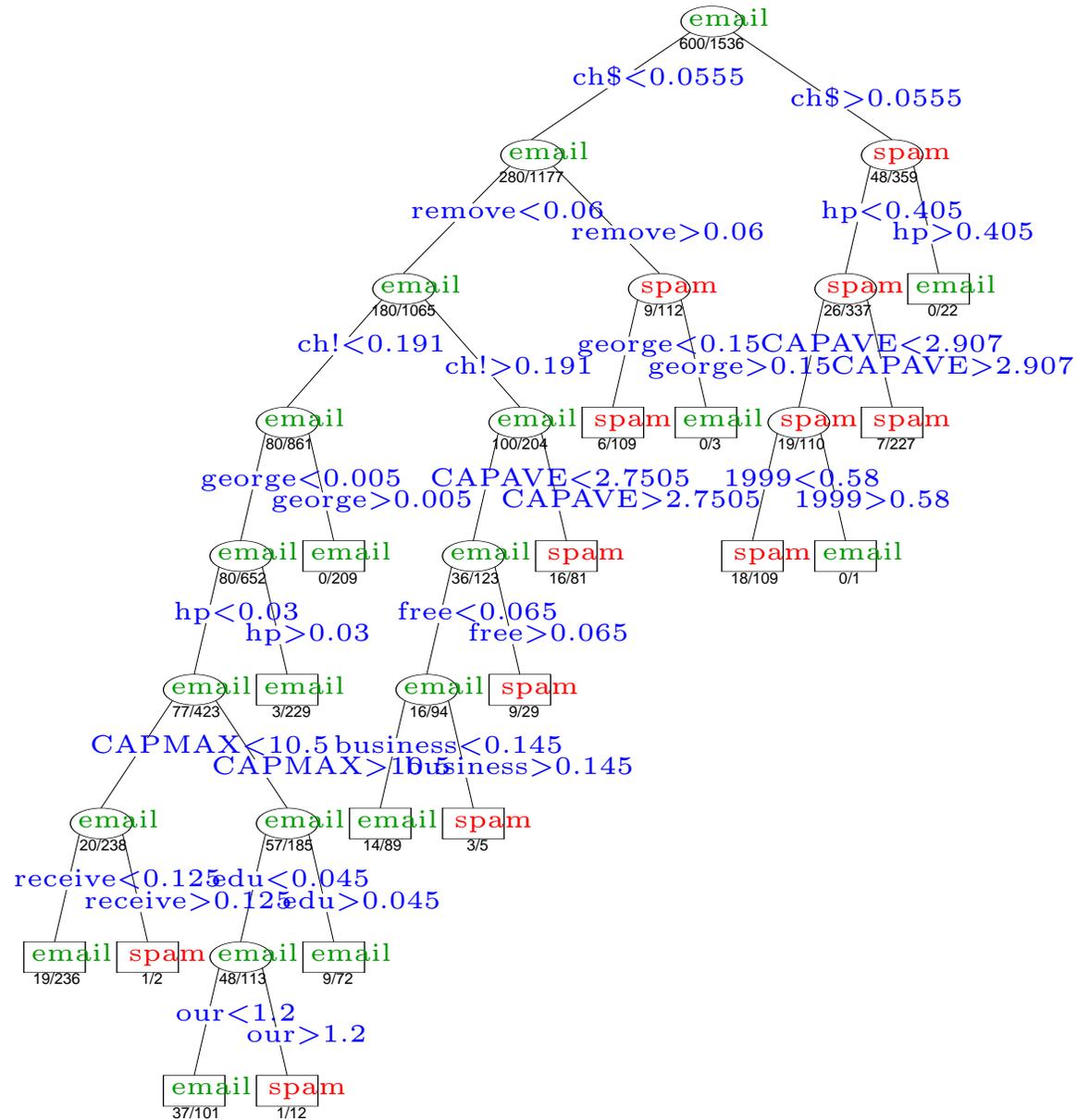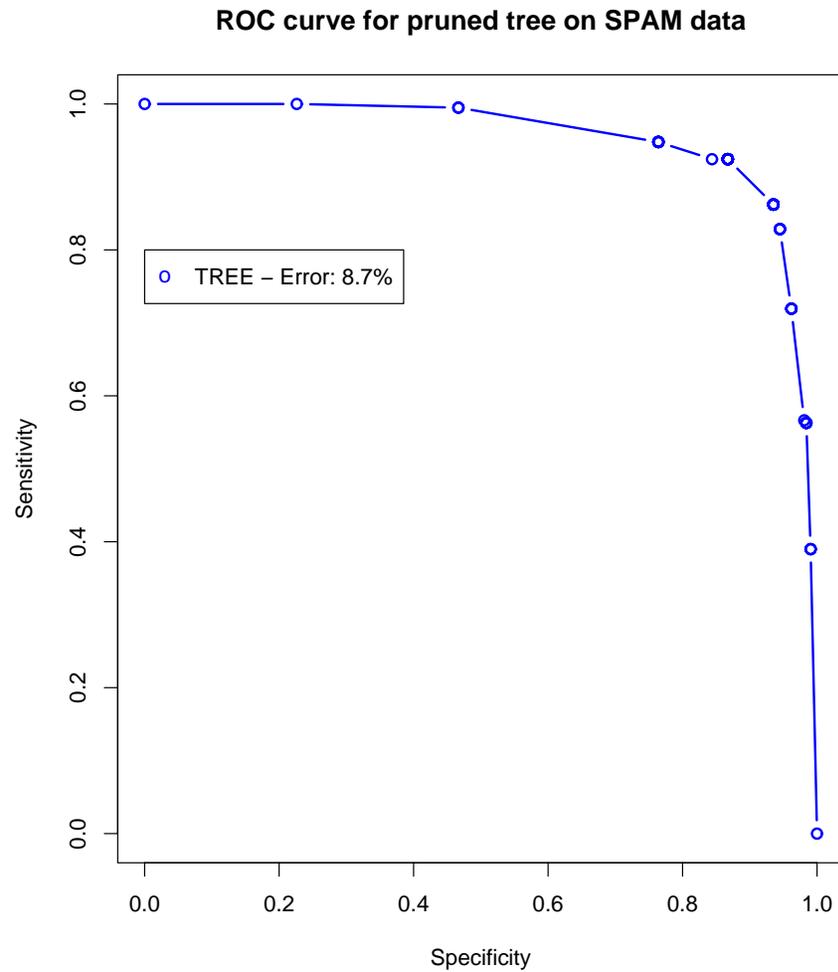
SPAM Data

Overall error rate on test data: 8.7%.

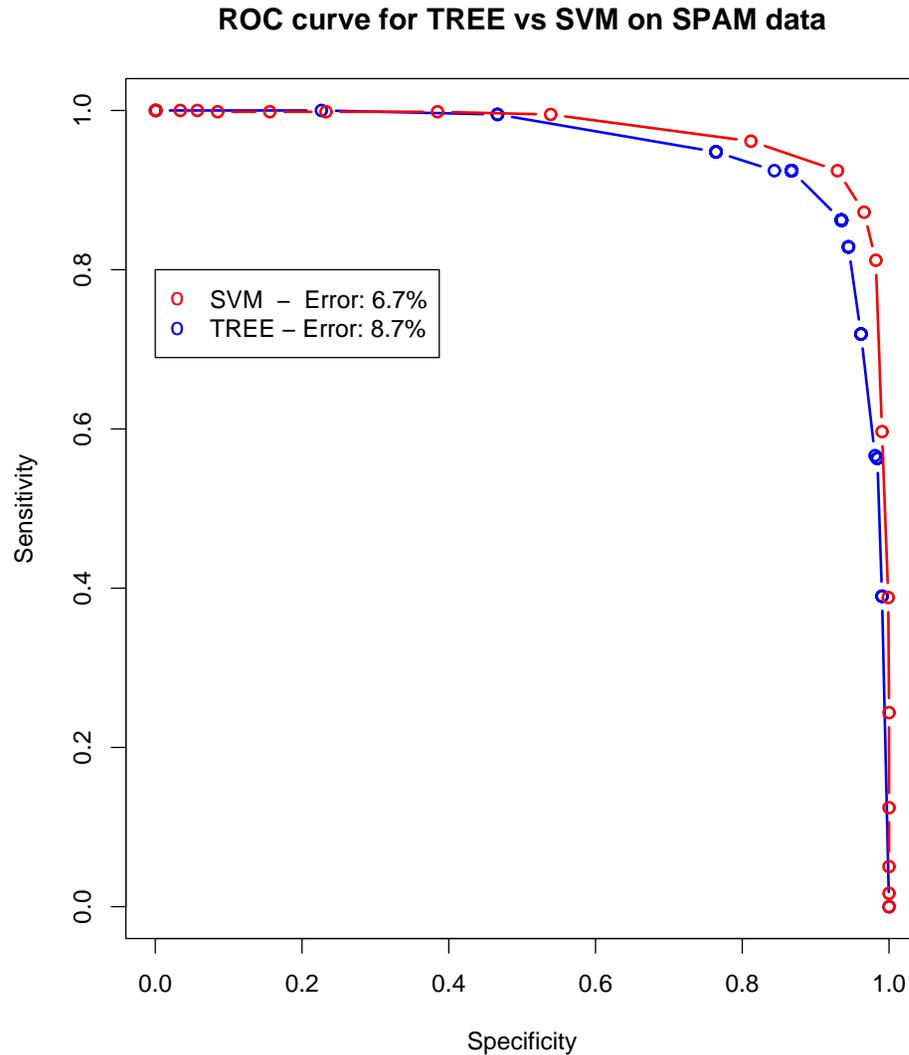ROC curve obtained by varying the threshold c_0 of the classifier:

$C(X) = +1$ if $\hat{P}(+1|X) > c_0$.

Sensitivity: proportion of true spam identified

Specificity: proportion of true email identified.

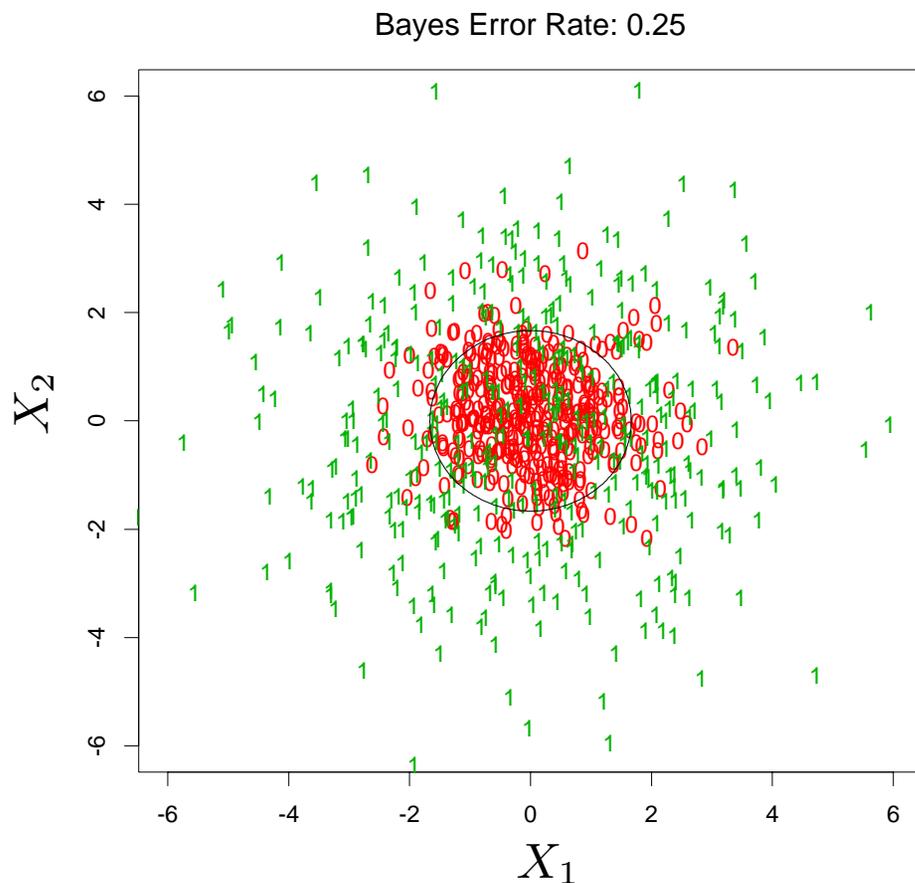We may want specificity to be high, and suffer some spam:

Specificity : $95\% \implies$ Sensitivity : 79%

ROC curve for TREE vs SVM on SPAM data

TREE vs SVM

Comparing ROC curves on the test data is a good way to compare classifiers. SVM dominates TREE here.

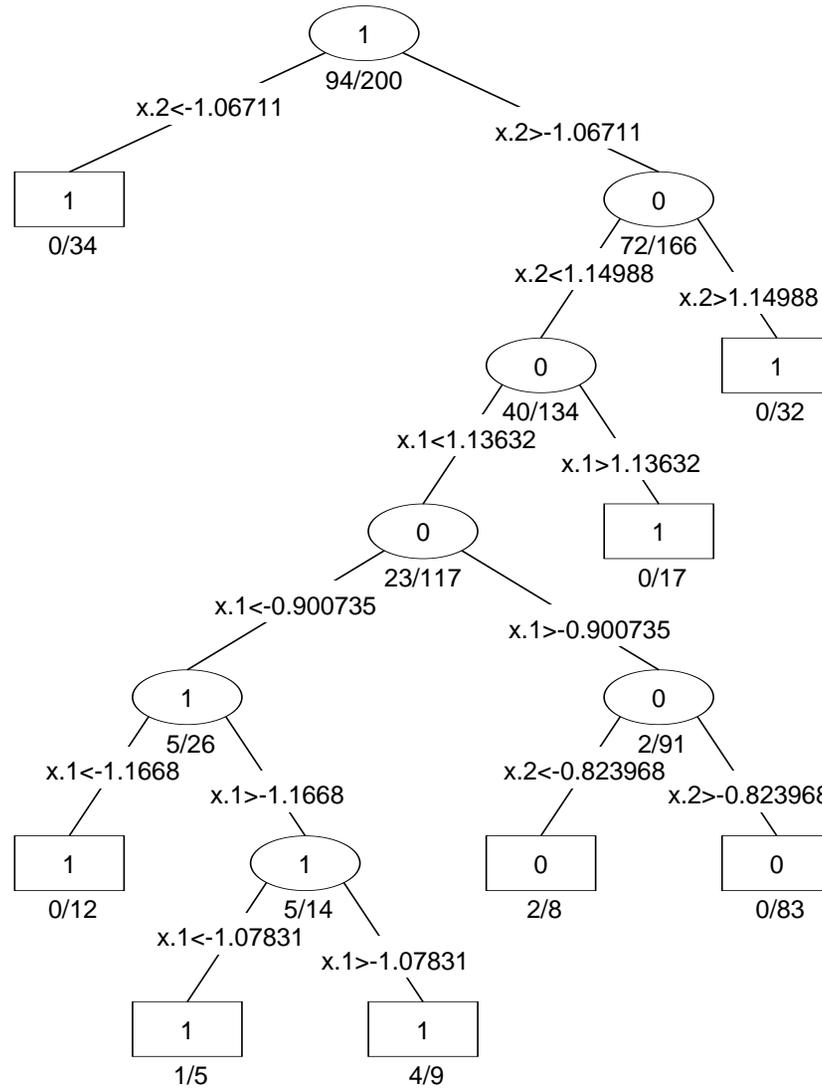## Toy Classification Problem

Bayes Error Rate: 0.25

- Data X and Y, with Y taking values $+1$ or -1.

- Here $X = (X_1, X_2)$

- The black boundary is the Bayes Decision Boundary - the best one can do.

- 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 $P(Y = +1|X)$.
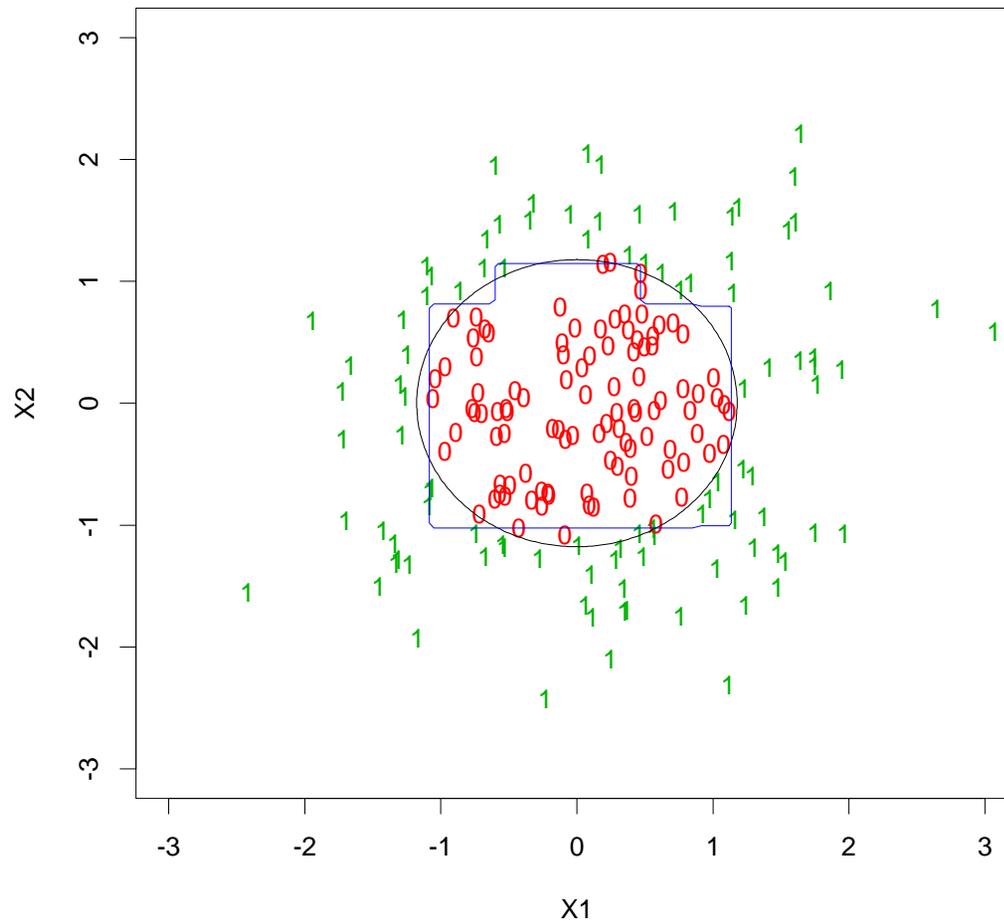
Toy Example - No Noise
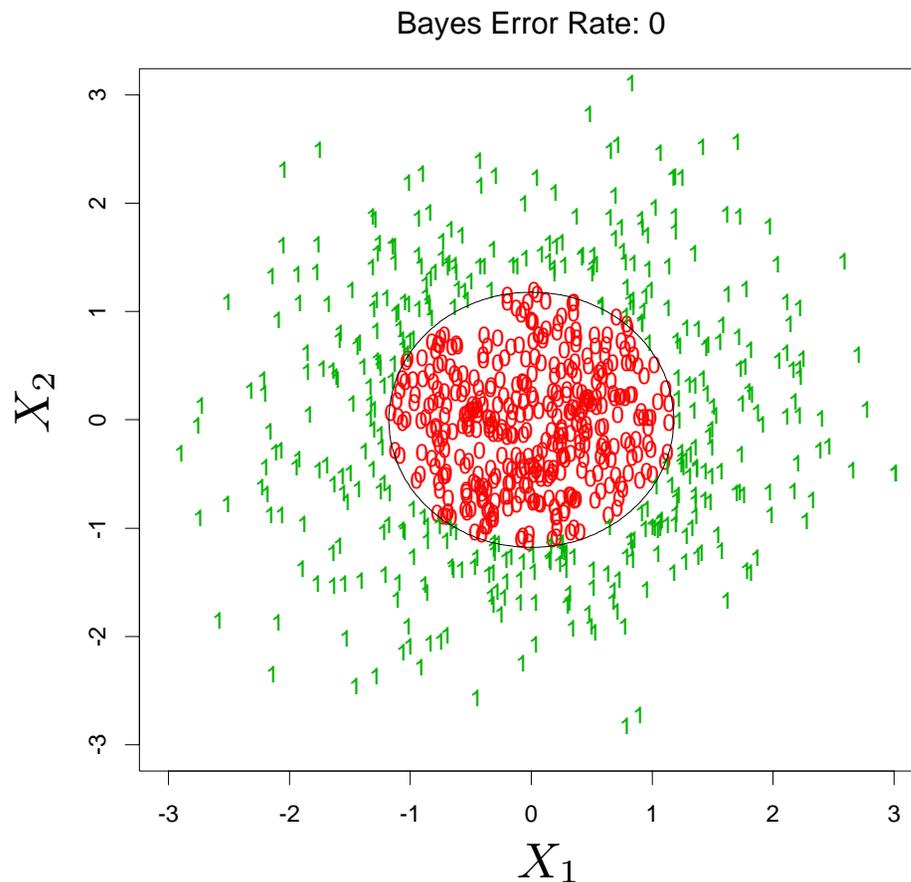
Bayes Error Rate: 0

- Deterministic problem; noise comes from sampling distribution of X.

- Use a training sample of size 200.

- Here Bayes Error is 0%.

Classification Tree

1
94/200

x.2<-1.06711

x.2>-1.06711

1
0/34

0
72/166

x.2<1.14988

x.2>1.14988

0
40/134

1
0/32

x.1<1.13632

x.1>1.13632

0
23/117

1
0/17

x.1<-0.900735

x.1>-0.900735

1
5/26

0
2/91

x.1<-1.1668

x.1>-1.1668

x.2<-0.823968

x.2>-0.823968

1
0/12

1
5/14

0
2/8

0
0/83

x.1<-1.07831

x.1>-1.07831

1
1/5

1
4/9

Decision Boundary: Tree

Error Rate: 0.073

When the nested spheres are in 10-dimensions, Classification Trees 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 given procedure over many samples, to reduce its variance — a poor man's Bayes. See

 pp 246.

Suppose $C(\mathcal{S}, x)$ is a classifier, such as a tree, based on 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
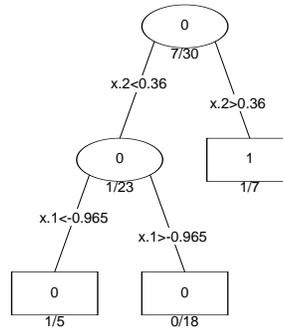
$$\hat{C}_{bag}(x) = \text{Majority Vote } \{C(\mathcal{S}^{*b}, x)\}_{b=1}^{B}.$$

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.
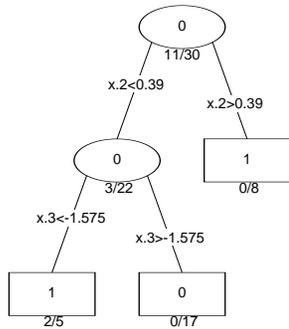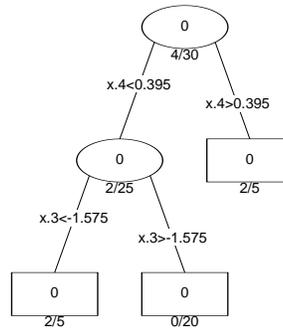
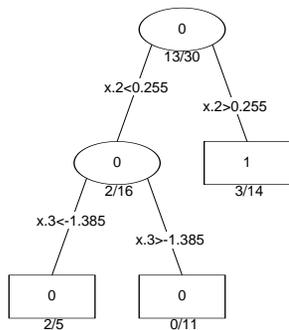## Original Tree
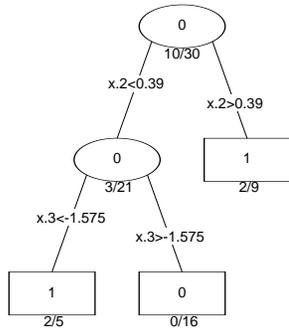
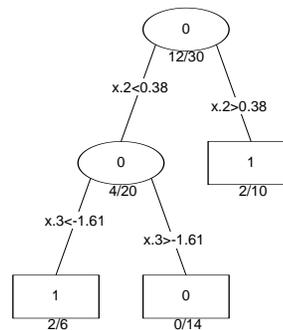## Bootstrap Tree 1

Bootstrap Tree 2

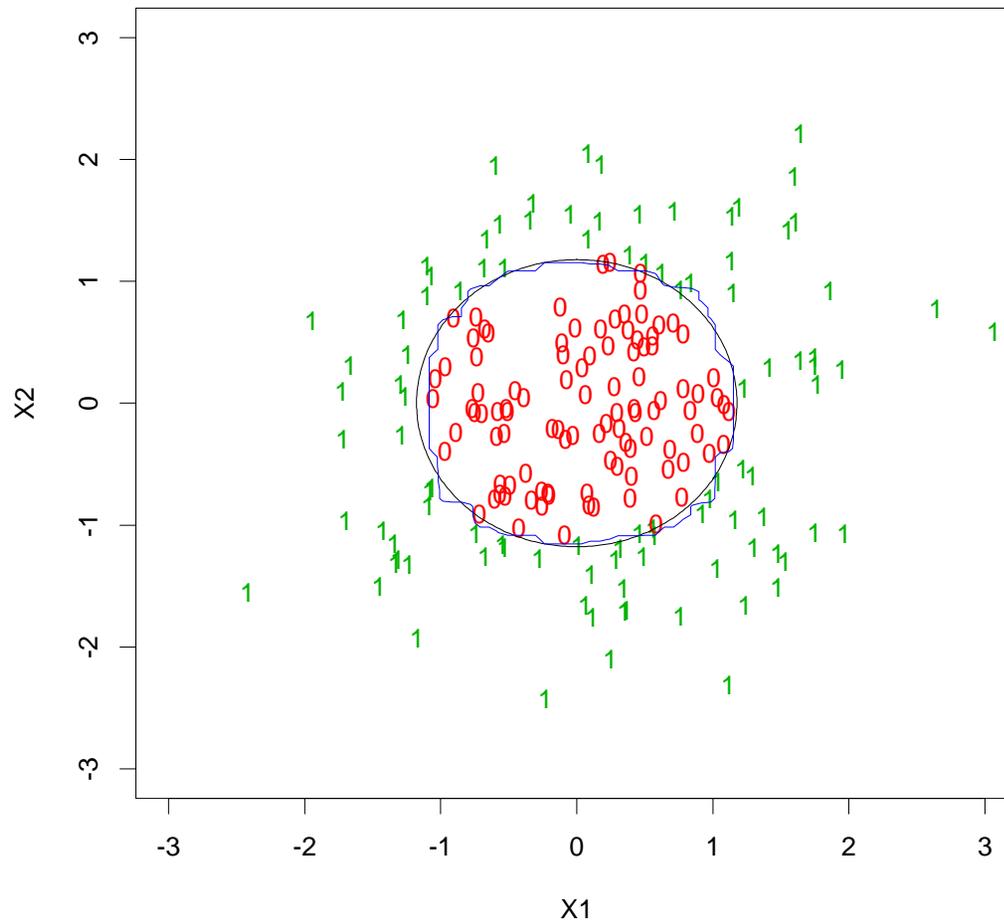

Bootstrap Tree 3

Bootstrap Tree 4

Bootstrap Tree 5

Decision Boundary: Bagging

Error Rate: 0.032

Bagging averages many trees, and produces smoother decision boundaries.

Random forests

- refinement of bagged trees; quite popular

- 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" error rate.

- random forests tries to improve on bagging by "de-correlating" the trees. Each tree has the same expectation.

ROC curve for TREE, SVM and Random Forest on SPAM data

Legend:
- Random Forest – Error: 5.0%
- SVM – Error: 6.7%
- TREE – Error: 8.7%
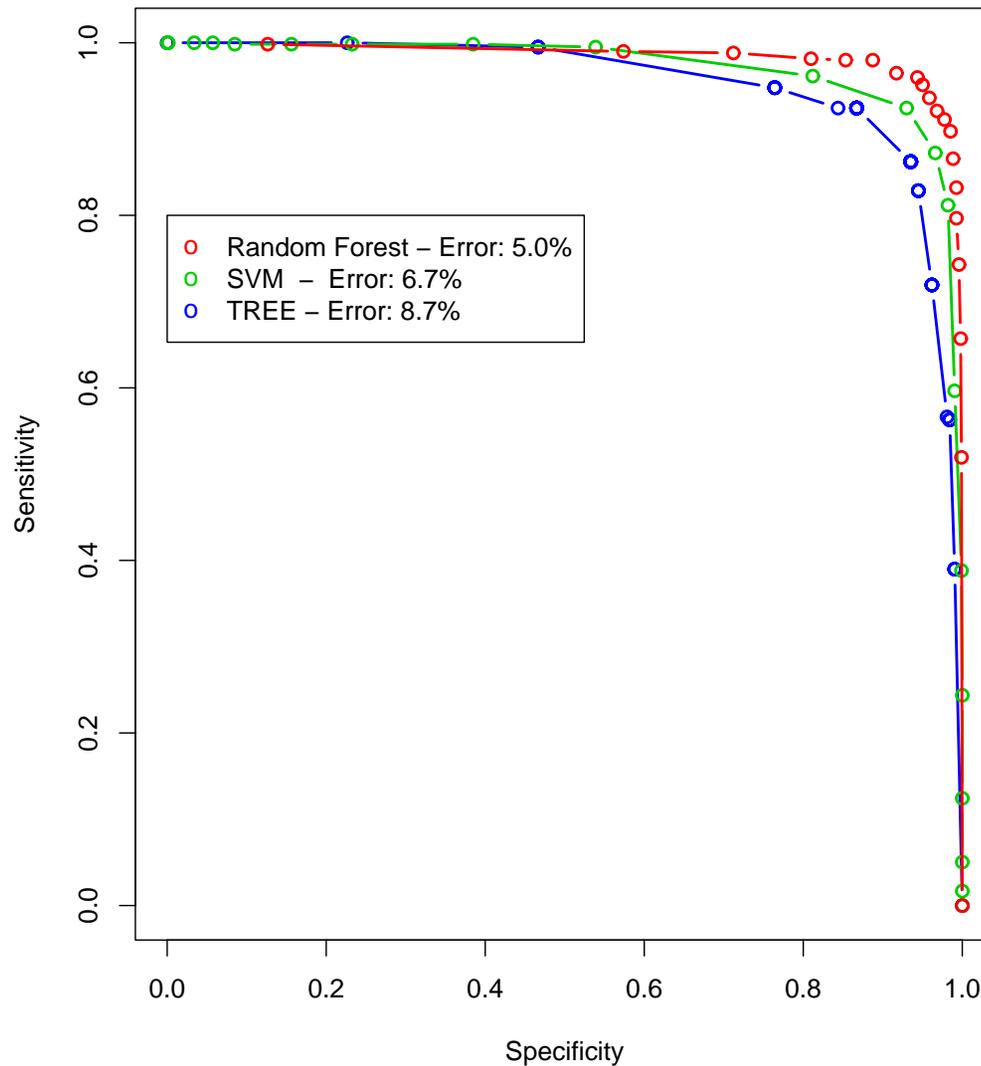
TREE, SVM and RF

Random Forest dominates both other methods on the SPAM data — 5.0% error. Used 500 trees with default settings for `random Forest` package in `R`.

$C_M(x)$

Weighted Sample $\cdots\rightarrow$ $C_M(x)$

Boosting

Weighted Sample $\cdots\rightarrow$ $C_3(x)$
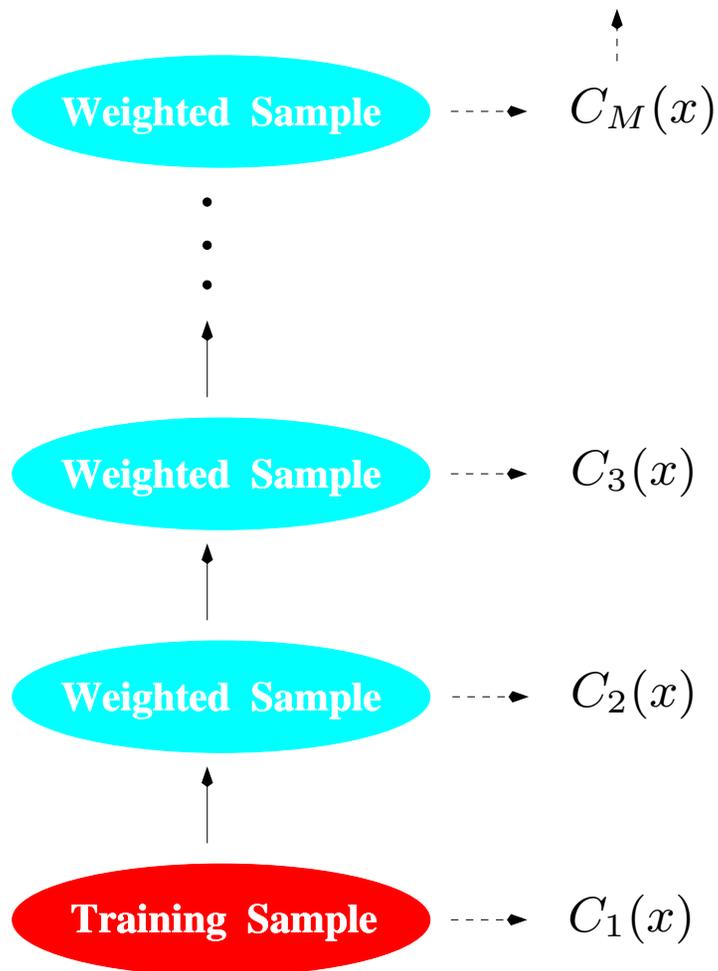
- Average many trees, each grown to re-weighted versions of the training data.

Weighted Sample $\cdots\rightarrow$ $C_2(x)$

- Final Classifier is weighted average of classifiers:

Training Sample $\cdots\rightarrow$ $C_1(x)$

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

100 Node Trees
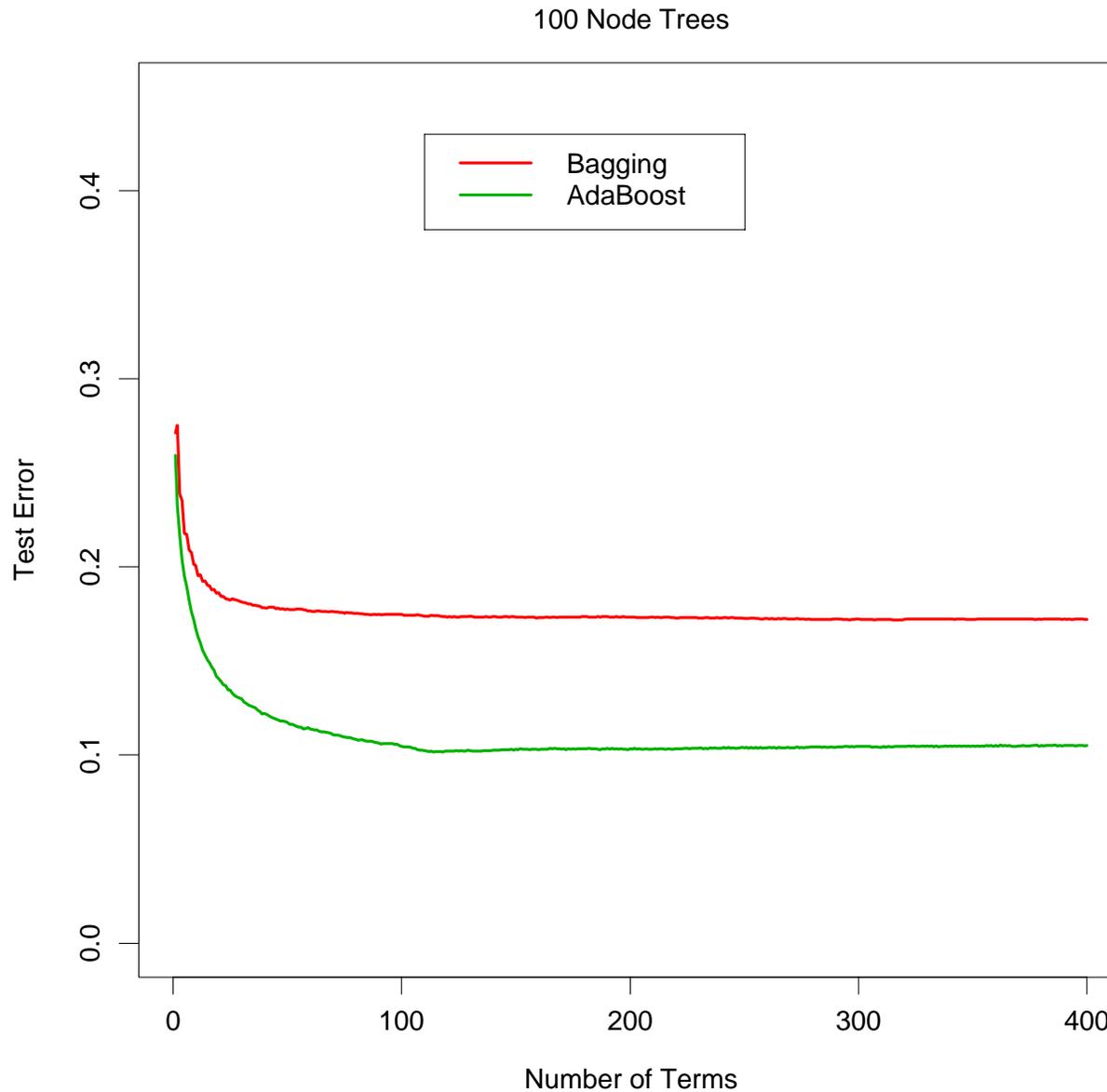
Boosting vs Bagging

- 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.
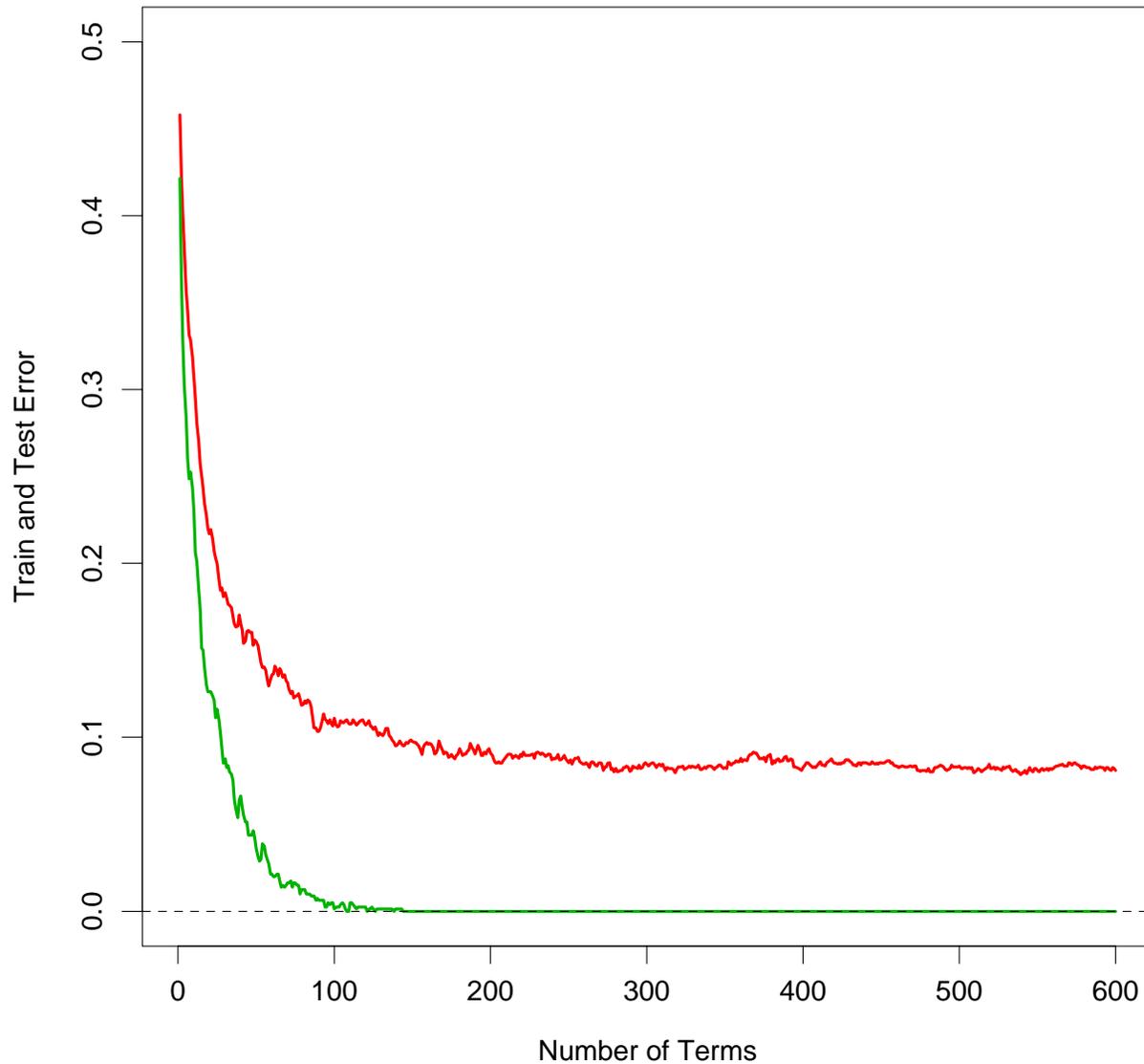
   (b) Compute weighted error of newest tree

   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq C_m(x_i))}{\sum_{i=1}^{N} w_i}.$$

   (c) Compute $\alpha_m = \log[(1 - \text{err}_m)/\text{err}_m]$.

   (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.

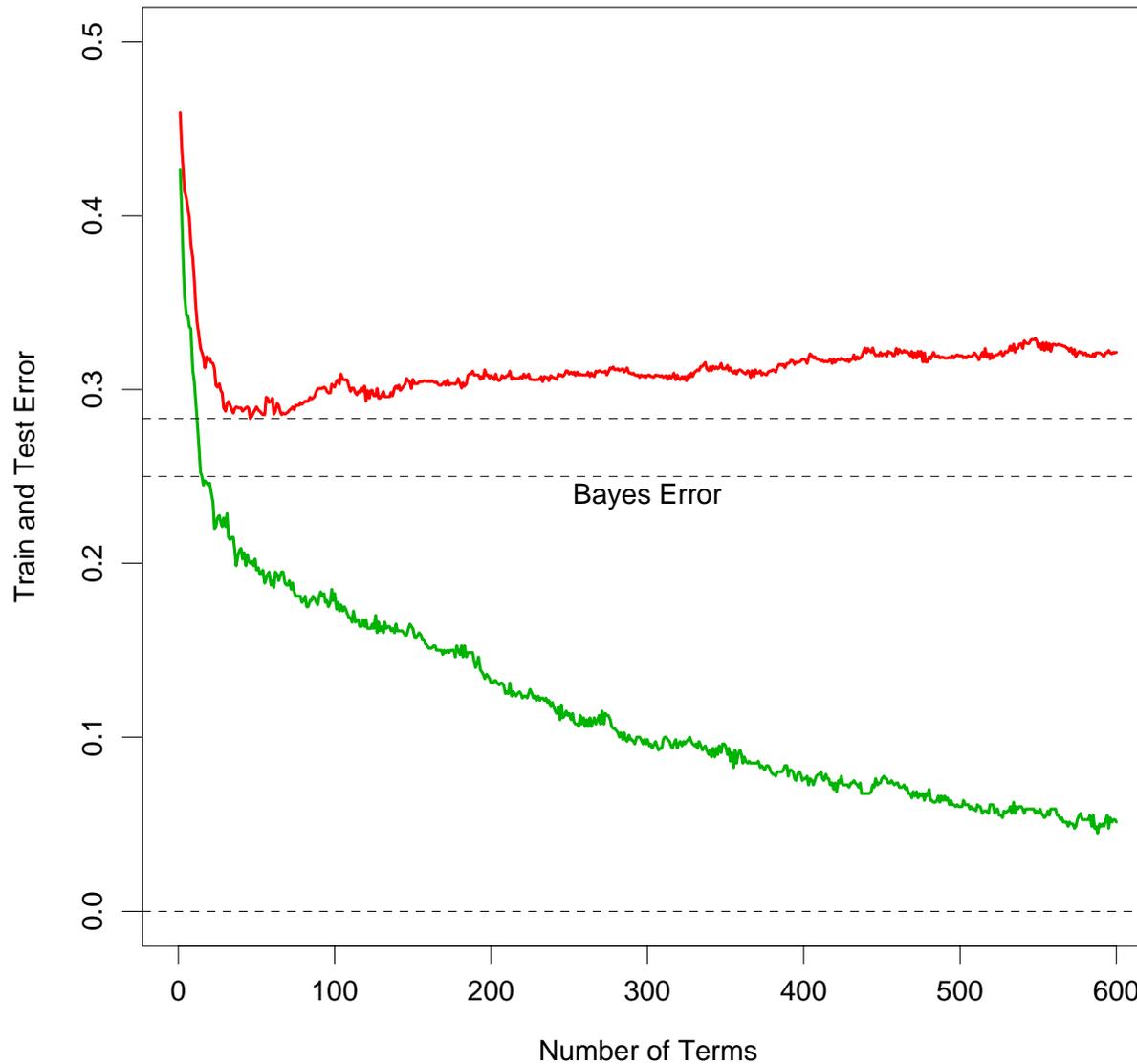3. Output $C(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m C_m(x)\right].$
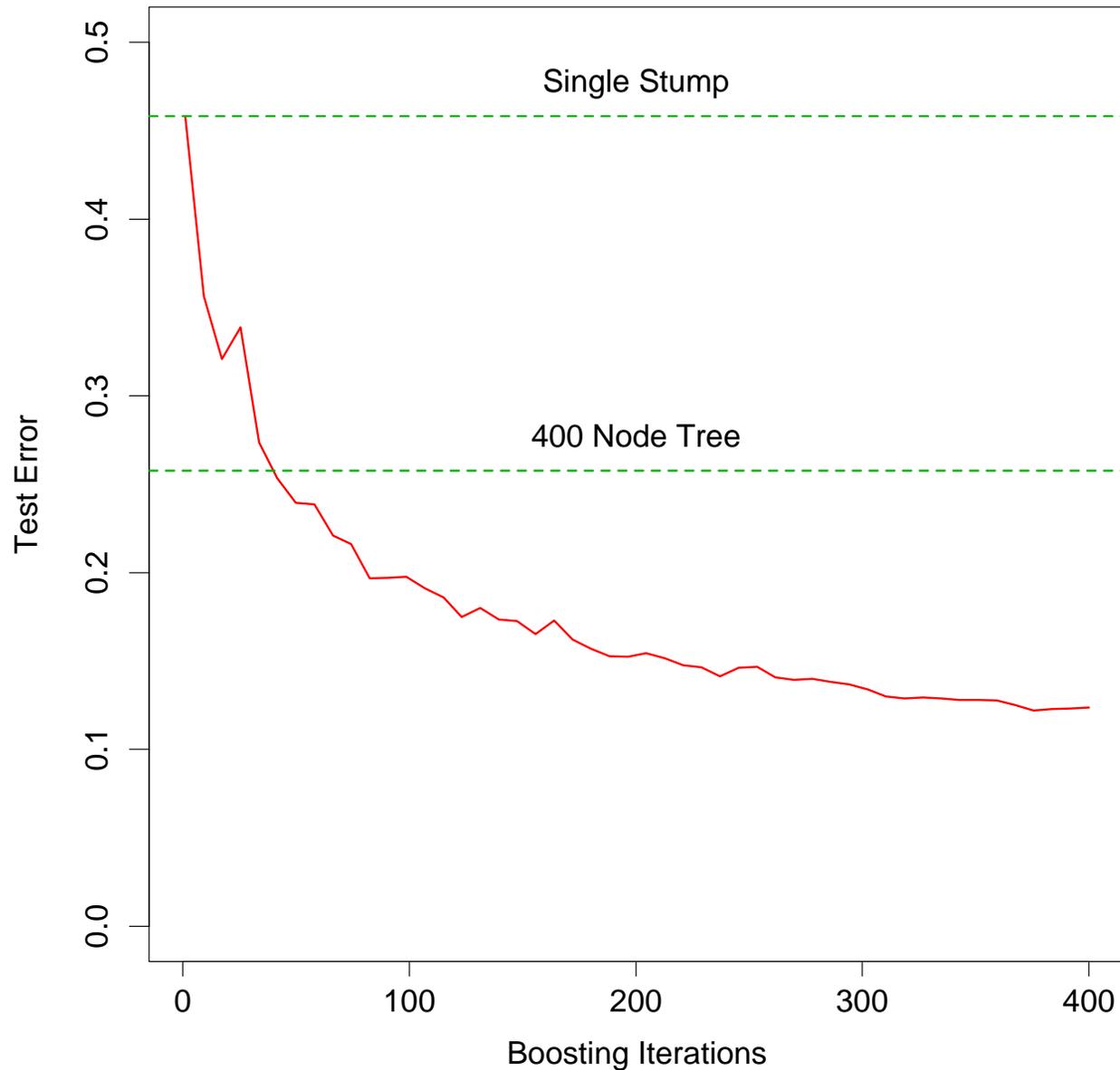
Boosting Stumps

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.

- Further iterations continue to improve test error in many examples.

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.

Additive Trees

- Simple example: stagewise least-squares?

- Fix the past $M - 1$ functions, and update the Mth using a tree:

$$\min_{f_M \in Tree(x)} E(Y - \sum_{m=1}^{M-1} f_m(x) - f_M(x))^2$$

- If we define the current residuals to be

$$R = Y - \sum_{m=1}^{M-1} f_m(x)$$

then at each stage we fit a tree to the residuals

$$\min_{f_M \in Tree(x)} E(R - f_M(x))^2$$

Stagewise Least Squares

Suppose we have available a basis family $b(x; \gamma)$ parametrized by γ.

- After $m - 1$ steps, suppose we have the model
  $f_{m-1}(x) = \sum_{j=1}^{m-1} \beta_j b(x; \gamma_j)$.

- At the mth step we solve

$$\min_{\beta, \gamma} \sum_{i=1}^{N} \left(y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma) \right)^2$$

- Denoting the residuals at the mth stage by
  $r_{im} = y_i - f_{m-1}(x_i)$, the previous step amounts to

$$\min_{\beta, \gamma} (r_{im} - \beta b(x_i; \gamma))^2,$$

- Thus the term $\beta_m b(x; \gamma_m)$ that best fits the current residuals is added to the expansion at each step.

Adaboost: Stagewise Modeling

- AdaBoost builds an additive logistic regression model

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

  by stagewise fitting using the loss function

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

- Given the current $f_{M-1}(x)$, our solution for (β_m, G_m) is

$$\arg\min_{\beta,G} \sum_{i=1}^{N} \exp[-y_i(f_{m-1}(x_i) + \beta\,G(x))]$$

  where $G_m(x) \in \{-1, 1\}$ is a tree classifier and β_m is a coefficient.

- With $w_i^{(m)} = \exp(-y_i\, f_{m-1}(x_i))$, this can be re-expressed as
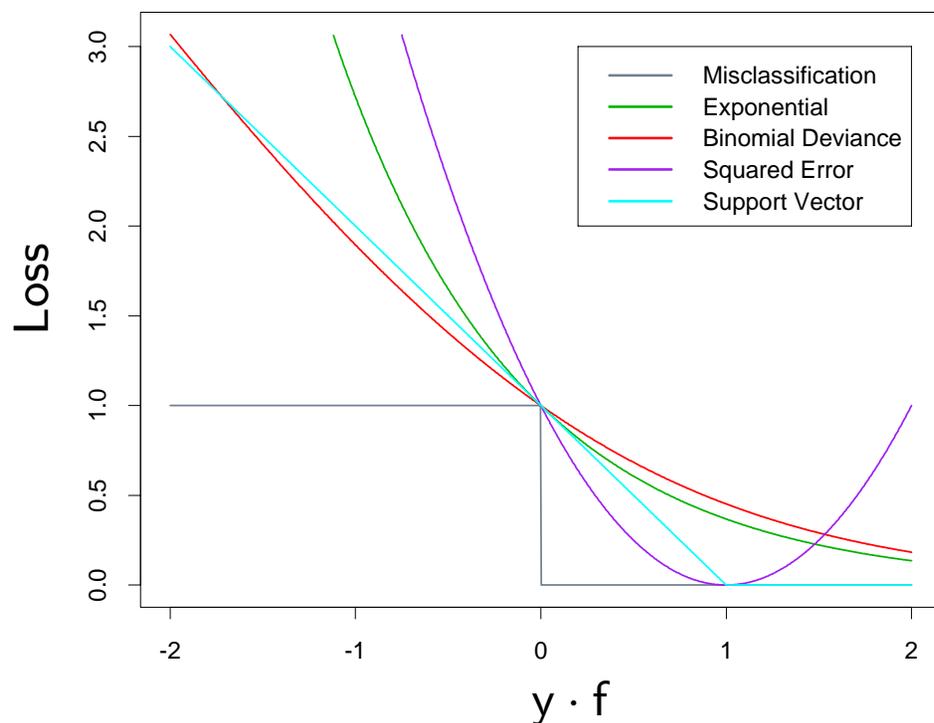
$$\arg\min_{\beta,G} \sum_{i=1}^{N} w_i^{(m)} \exp(-\beta\, y_i\, G(x_i))$$

- We can show that this leads to the Adaboost algorithm; See

 pp 305.

Why Exponential Loss?

- $e^{-yF(x)}$ 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 Stagewise Algorithm

We can do the same for more general loss functions, not only least squares.

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 $f_m(x) = f_{m-1}(x) + \nu \beta_m b(x; \gamma_m)$

Here ν is a shrinkage factor, and often $\nu < 0.1$. 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.
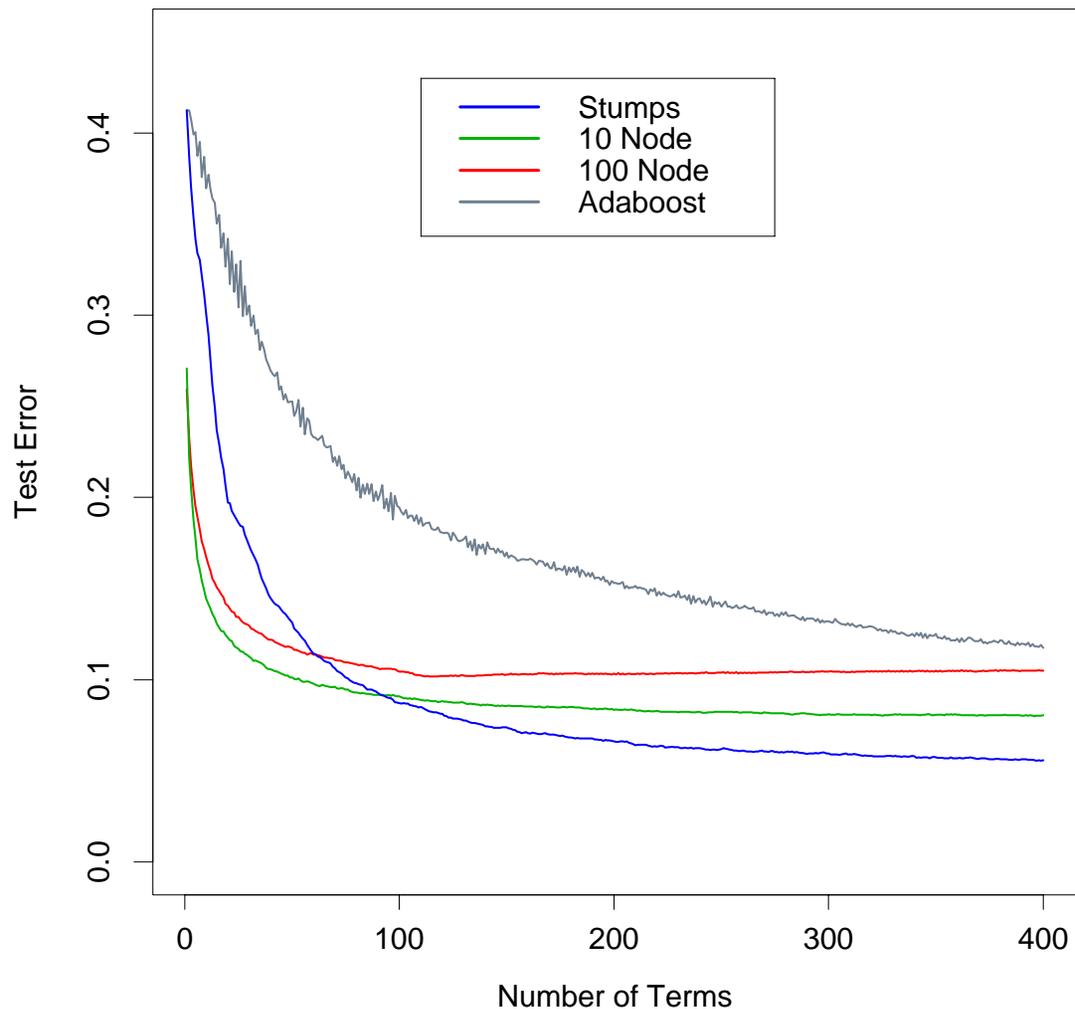
Tree Size

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

$$\eta(X) = \sum_j \eta_j(X_j)$$

$$+ \sum_{jk} \eta_{jk}(X_j, X_k)$$

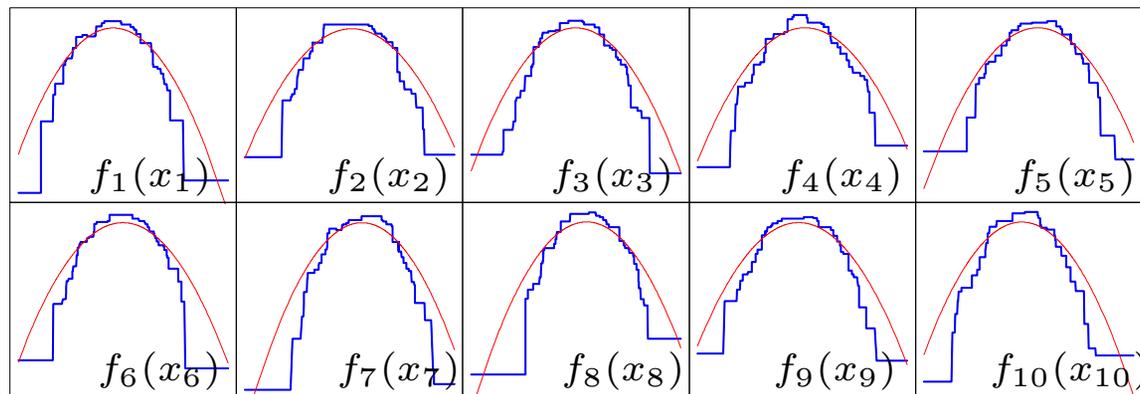$$+ \sum_{jkl} \eta_{jkl}(X_j, X_k, X_l)$$

$$+ \cdots$$

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

Spam Example Results

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

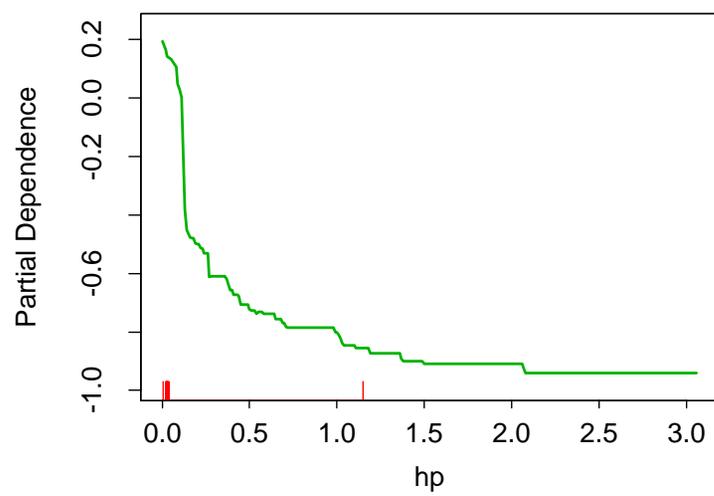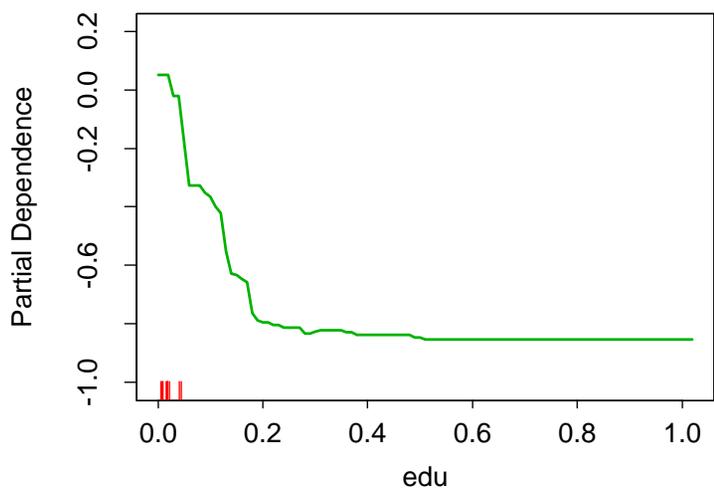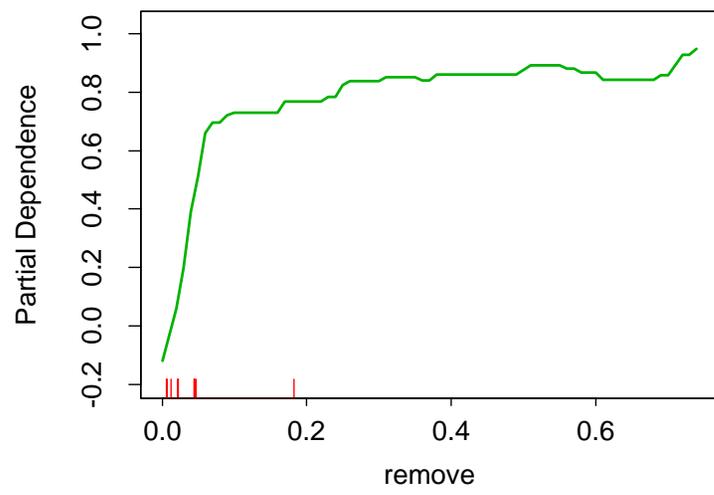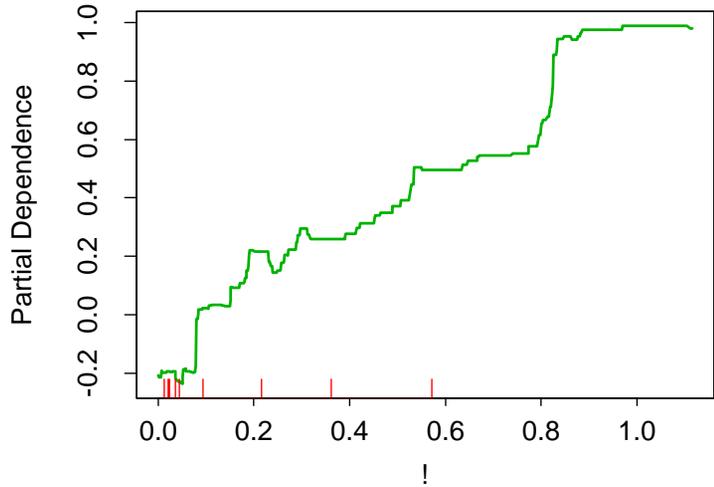$$f(x) = \log \frac{\Pr(\texttt{spam}|x)}{\Pr(\texttt{email}|x)}$$

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

Gradient Boosting achieves a test error of 4%, compared to 5.3% for an additive GAM, 5.0% for Random Forests, and 8.7% for CART.

Spam: Variable Importance
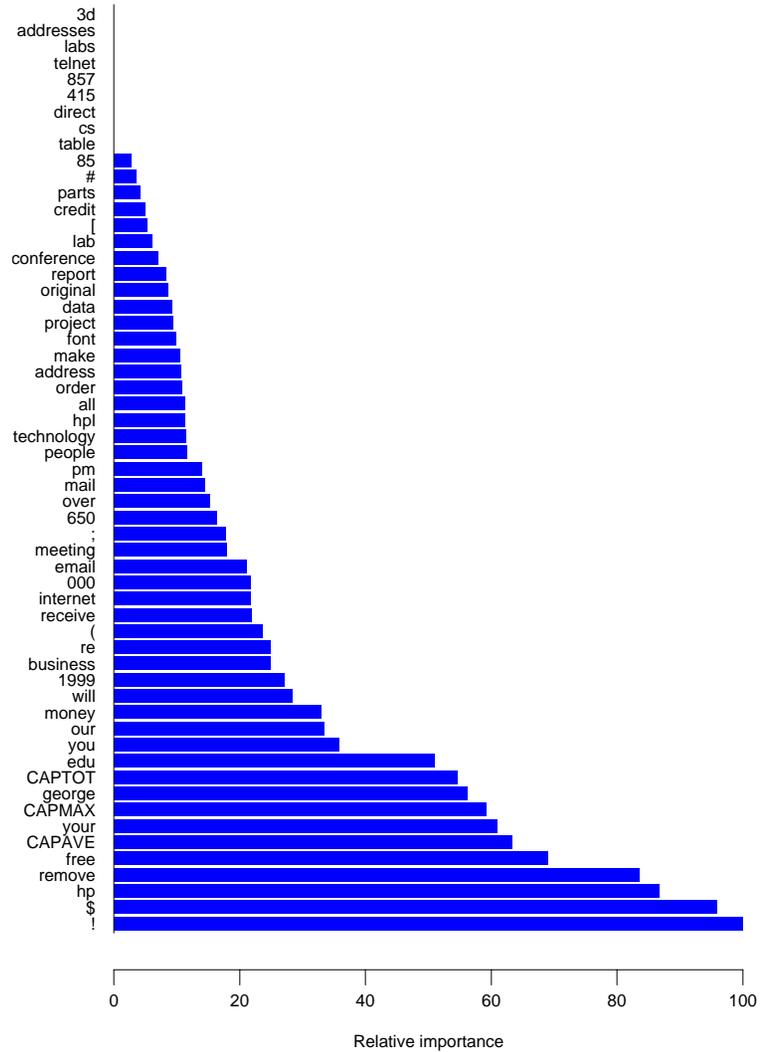
Spam: Partial Dependence

Comparison of Learning Methods

Some characteristics of different learning methods.

Key: ●= good, ●=fair, and ●=poor.

Characteristic	Neural Nets	SVM	CART	GAM	KNN, Kernel	Gradient Boost
Natural handling of data of "mixed" type	poor	poor	good	good	poor	good
Handling of missing values	poor	poor	good	poor	good	good
Robustness to outliers in input space	poor	poor	good	fair	good	good
Insensitive to monotone transformations of inputs	poor	poor	good	good	poor	good
Computational scalability (large N)	poor	poor	good	good	poor	good
Ability to deal with irrelevant inputs	poor	poor	good	fair	poor	good
Ability to extract linear combinations of features	good	good	poor	poor	fair	fair
Interpretability	poor	poor	fair	good	poor	good
Predictive power	good	good	poor	poor	good	good

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.