Lecture 11: Boosting, Bootstrap

More Random Forests, Boosted trees, Bootstrap

Prof. Alexandra Chouldechova
95-791: Data Mining

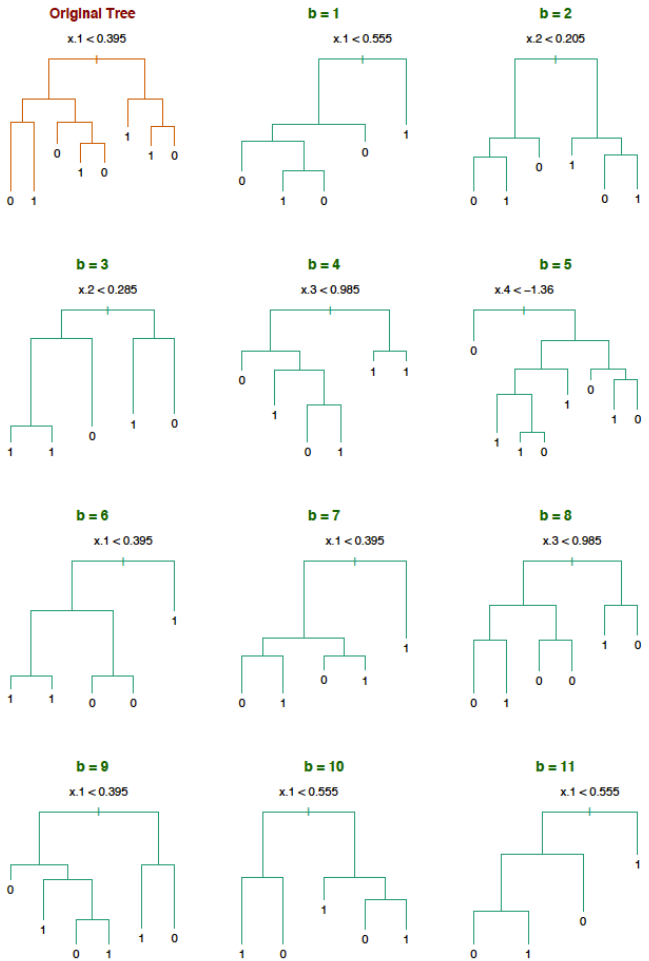
April 19, 2016

Agenda

- **Random Forests**
- **Bootstrap SE estimates, Confidence intervals**

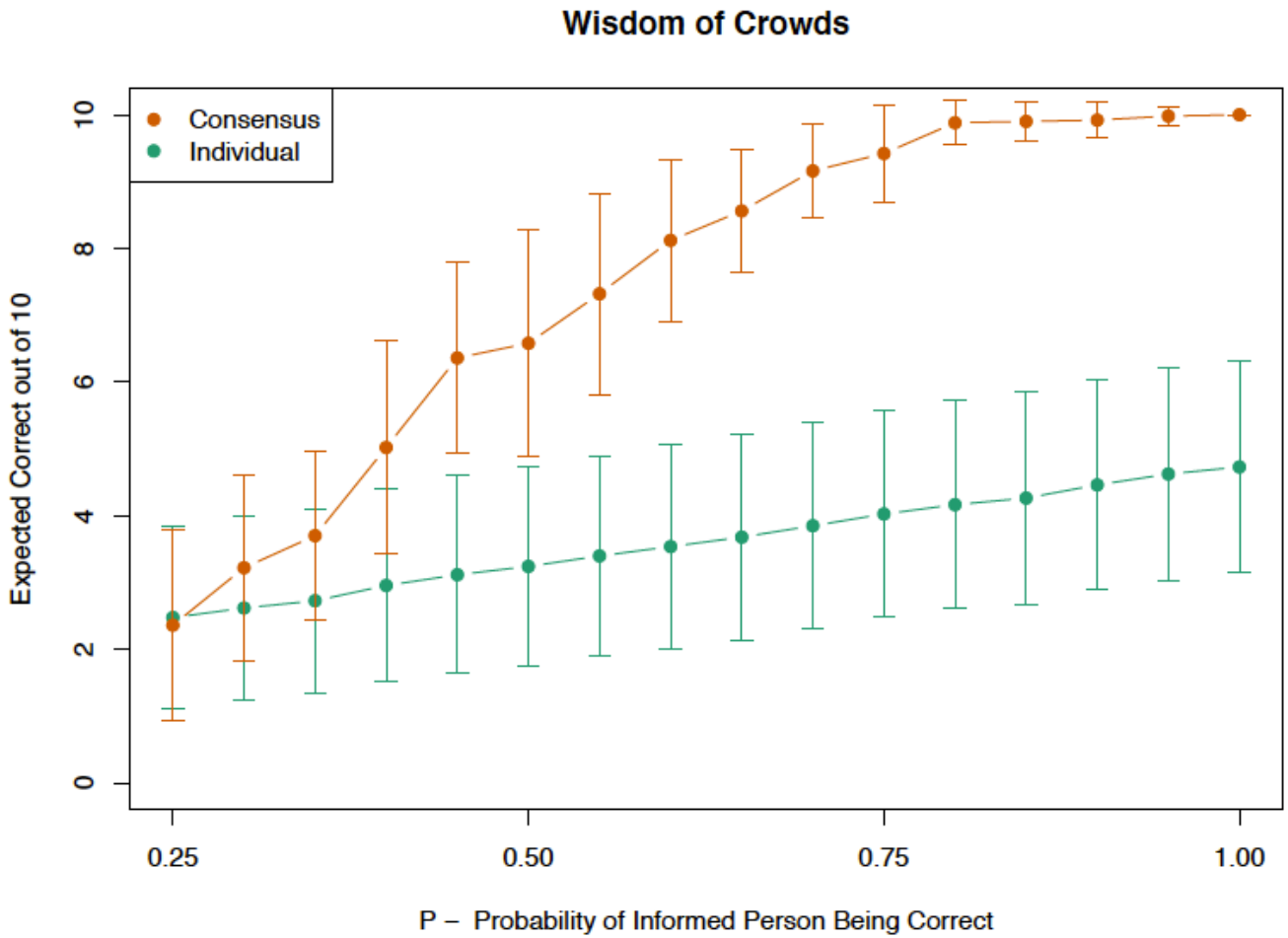
Example: bagging

Example (from ESL 8.7.1): $n = 30$ training data points, $p = 5$ features, and $K = 2$ classes. No pruning used in growing trees:



How could this possibly work?

- You may have heard of the Wisdom of crowds phenomenon
- It's a concept popularized outside of statistics to describe the idea that the collection of knowledge of a group of independent people can exceed the knowledge of any one person individually.
- Interesting example (from ESL page 287): Academy award predictions
  - 50 people are asked to predict academy award winners for 10 categories
  - Each category has 4 nominees
  - For each category, just 15 of the 50 voters are at all informed (the remaining 35 voters are guessing randomly)
  - The 15 informed voters have probability $P \geq 0.25$ of correctly guessing the winner



There are 10 award categories and 4 nominees in each. For each of the 10 categories, there are 15 (of 50) voters who are *informed*. Their probability of guessing correctly is $P \geq 0.25$. Everyone else guesses randomly.

Example: Breiman's bagging

Example from the original Breiman paper on bagging: comparing the misclassification error of the CART tree (pruning performed by cross-validation) and of the bagging classifier (with $B = 50$):

Data Set	$\bar{e}_S$	$\bar{e}_B$	**Decrease**
waveform	29.1	19.3	34%
heart	4.9	2.8	43%
breast cancer	5.9	3.7	37%
ionosphere	11.2	7.9	29%
diabetes	25.3	23.9	6%
glass	30.4	23.6	22%
soybean	8.6	6.8	21%

Voting probabilities are not estimated class probabilities

- Suppose that we wanted *probability estimates* $\hat{p}_k(x)$ out of our bagging procedure.
- What if we tried using:

$$\hat{p}_k^{\text{vote}}(x) = \frac{1}{B}\sum_{b=1}^{B}\left(\hat{f}^{\text{tree},b}(x) = k\right)$$

  This is the proportion of bootstrapped trees that voted for class k.
- This can be a bad idea
- Suppose we have two classes, and the true probability that $y_0 = 1$ when $X = x_0$ is 0.75.
- Suppose each of the bagged trees $\hat{f}^{\text{tree},b}(x)$ correctly classifies x_0 to class 1
- Then $\hat{p}_1^{\text{vote}}(x) = 1$... that's wrong
- What if we used each tree's estimated probabilities instead?

Alternative form: Probability Bagging

- Instead of just looking at the class predicted by each tree, look at the *predicted class probabilities* $\hat{p}_k^{\text{tree},b}(x)$
- Define the bagging estimate of class probabilities:

$$\hat{p}_k^{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{p}_k^{\text{tree},b}(x) \quad k = 1, \ldots K$$

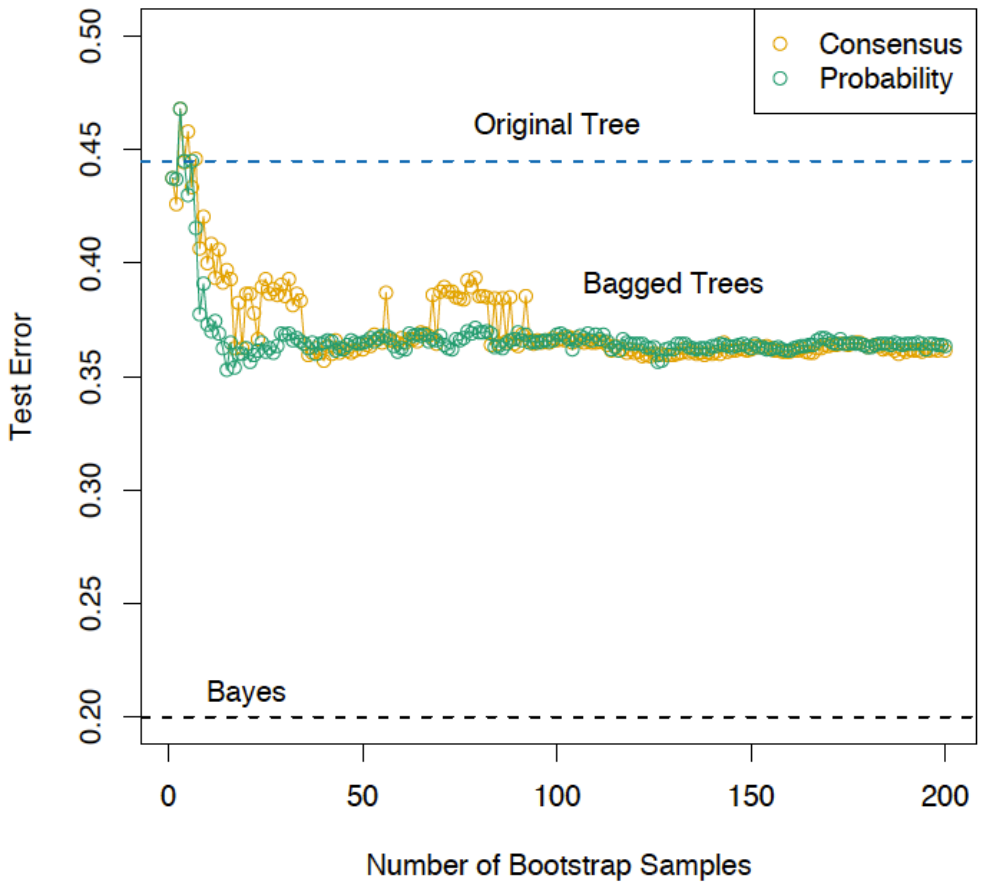
- We can use $\hat{p}_k^{\text{bag}}(x)$ itself as an *alternative* to plurality voting of the trees.
- Given an input vector x_0, we can classify it according to

$$\hat{y}_0^{\text{bag}} = \underset{k=1,\ldots K}{\text{argmax}}\ \hat{p}_k^{\text{bag}}(x)$$

- This form of bagging is preferred if we want to estimate class probabilities, and it may improve overclass classification accuracy

Comparison of the two bagging approaches

The probability form of bagging produces misclassification errors shown in green. The Consensus version is what we first introduced. It's not as well behaved.



The Test error eventually stops decreasing past a certain value of B because we hit the limit in the variance reduction bagging can provide

Out-of-Bag (OOB) Error Estimation

- Recall, each bootstrap sample contains roughly 2/3 ($\approx 63.2\%$) of the of the training observations

- The remaining observations not used to fit a given bagged tree are called the out-of-bag (OOB) observations

- Another way of thinking about it: Each observation is OOB for roughly $B/3$ of the trees. We can treat observation i as a test point each time it is OOB.

- To form the OOB estimate of test error:
  - Predict the response for the ith observation using each of the trees for which i was OOB. This gives us roughly $B/3$ predictions for each observation.
  - Calculate the error of each OOB prediction
  - Average all of the errors

Random Forests

- Random forests provide an improvement over bagged trees by incorporating a small tweak that decorrelates the individual trees
  - This further reduces variance when we average the trees

- We still build each tree on a bootstrapped training sample

- But now, each time a split in a tree is considered, the tree may only split on a predictor from a randomly selected subset of m predictors

- A fresh selection of m randomly selected predictors is presented at each split... not for each tree, but for each split of each tree

- $m \approx \sqrt{p}$ turns out to be a good choice
  - E.g., if we have 100 predictors, each split will be allowed to choose from among 10 randomly selected predictors

Bagging vs. Random Forests

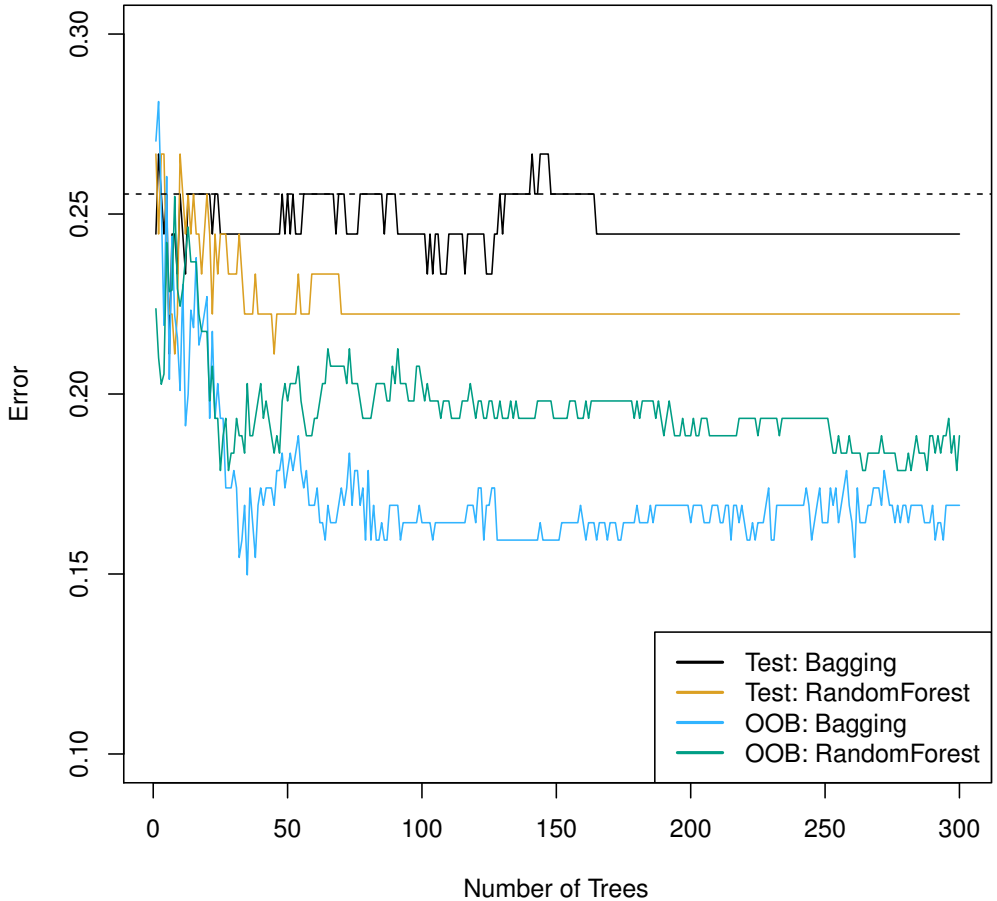


Figure 8.8 from ISL. Various fits to the `Heart` data

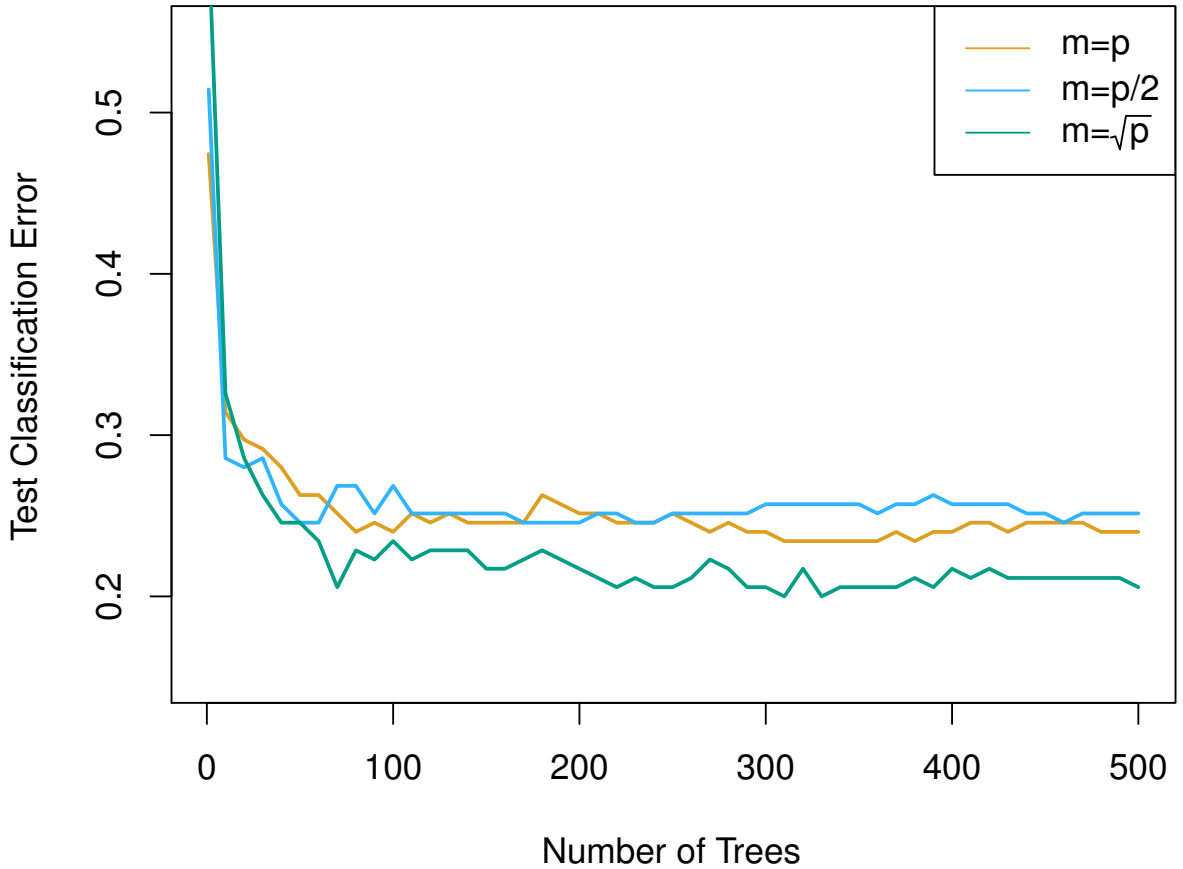
Dashed: Error from a single Classification tree

Random forest fit with $m = 4 \approx \sqrt{13} = \sqrt{p}$

A big data example: Gene expression data

- $p = 4{,}718$ genetic measurements from just 349 patients
- Each patient has a qualitative label. $K = 15$ possible labels
  - Either normal, or one of 14 types of cancer
- Split data into training and testing, fit Random forest to training set for 3 different choices of number of splitting variables, m.
- First, filter down to the 500 genes that have the highest overall variance in the training set

Test error: Gene expression data



- Curves show Test misclassification rates for a 15-class problem with $p = 500$ predictors and under 200 observations used for training
- x-axis gives number of trees (number of bootstrap samples used)
- $m = p$ corresponds to bagging.
- A single classification tree has an error rate of 45.7%.

Summary: Random forests

- Random forests have two tuning parameters:
  - $m =$ the number of predictors considered at each split
  - $B =$ the number of trees (number of bootstrapped samples)

- Increasing B helps decrease the overall variance of our estimator

- $m \approx \sqrt{p}$ is a popular choice

- Cross-validation can be used across a grid of (m, B) values to find the choice that gives the lowest CV estimate of test error

- Out-of-bag (OOB) error rates are commonly reported
  - Each observation is OOB for around $B/3$ of the trees
  - Can get a test error estimate for each observation each time it is OOB
  - Average over all the OOB errors to estimate overall test error

- RF's are parallelizable: You can distribute the computations across multiple processors and build all the trees *in parallel*

Random forests vs. Trees

- We liked trees because the model fit was very easy to understand
  - Large trees wind up hard to interpret, but small trees are highly interpretable

- With random forests, we're averaging over a bunch of bagged trees, and each tree is built by considering a small random subset of predictor variables at each split.

- This leads to a model that's *essentially* uninterpretable

- The Good: Random forests are very flexible and have a *somewhat justifiable* reputation for not overfitting
  - Clearly RF's can overfit: If we have 1 tree and consider all the variables at each split, $m = p$, we just have a single tree
  - If $m \ll p$ and the number of trees is large, RF's tend not to overfit
  - You should still use CV to get error estimates and for model tuning! Don't simply rely on the reputation RF's have for not overfitting.

Boosting

- We talked about Bagging in the context of bagged trees, but we can bag any predictor or classifier we want
- Boosting is yet another general approach that can be applied to any based learning method
- Here we'll briefly discuss Boosting decision trees
- Boosting is another way of taking a base learner (a model) and building up a more complex ensemble
- In bagging, we bootstrap multiple versions of the training data, fit a model to each sample, and then combine all of the estimates
  - The base learners used in bagging tend to have high variance, low bias
- Boosting builds up the ensemble sequentially: E.g., To boost trees, we grow small trees, one at a time, at each step trying to improve the model fit in places we've done poorly so far

Boosting algorithm: Regression trees

1. Set $\hat{f}(x) = 0$ and *residuals* $r_i = y_i$ for all i in the training set
2. For $b = 1, 2, \ldots, B$, repeat:
   1. Fit tree $\hat{f}^b$ with d splits to the training data (X, r)[1]
   2. Update $\hat{f}$ by adding *shrunken* version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

   3. Update the residuals

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

- λ is called the shrinkage parameter. Typically, $\lambda \ll 1$

[1] We're treating the residual vector r as our outcome at each step.

What's boosting trying to do?

- Boosting works best if d (the size of each tree) is small
- Given the current model, we fit a decision tree to the *residuals* from the model
- This new tree helps us perform just a little bit better in places where the current model wasn't doing well
- Unlike bagging, where each tree is large and tries to model the entire (bootstrapped) data well, each tree in boosting tries to incrementally improve the existing model
- Think of boosting as learning slowly: We use small trees, try to make incremental improvements, and further slow down the learning process by incorporating the *shrinkage parameter* λ

Boosting for classification

- The basic idea is the same: Use weak base learners, update incrementally, shrink at each step
- Details are more complicated…too complicated to present here
- The **R** package `gbm` (gradient boosted models) handles both prediction (regression) and classification problems
- If you're interested in the details, see Chapter 10 of Elements of Statistical Learning

Gene expression example: RF vs Boosted trees

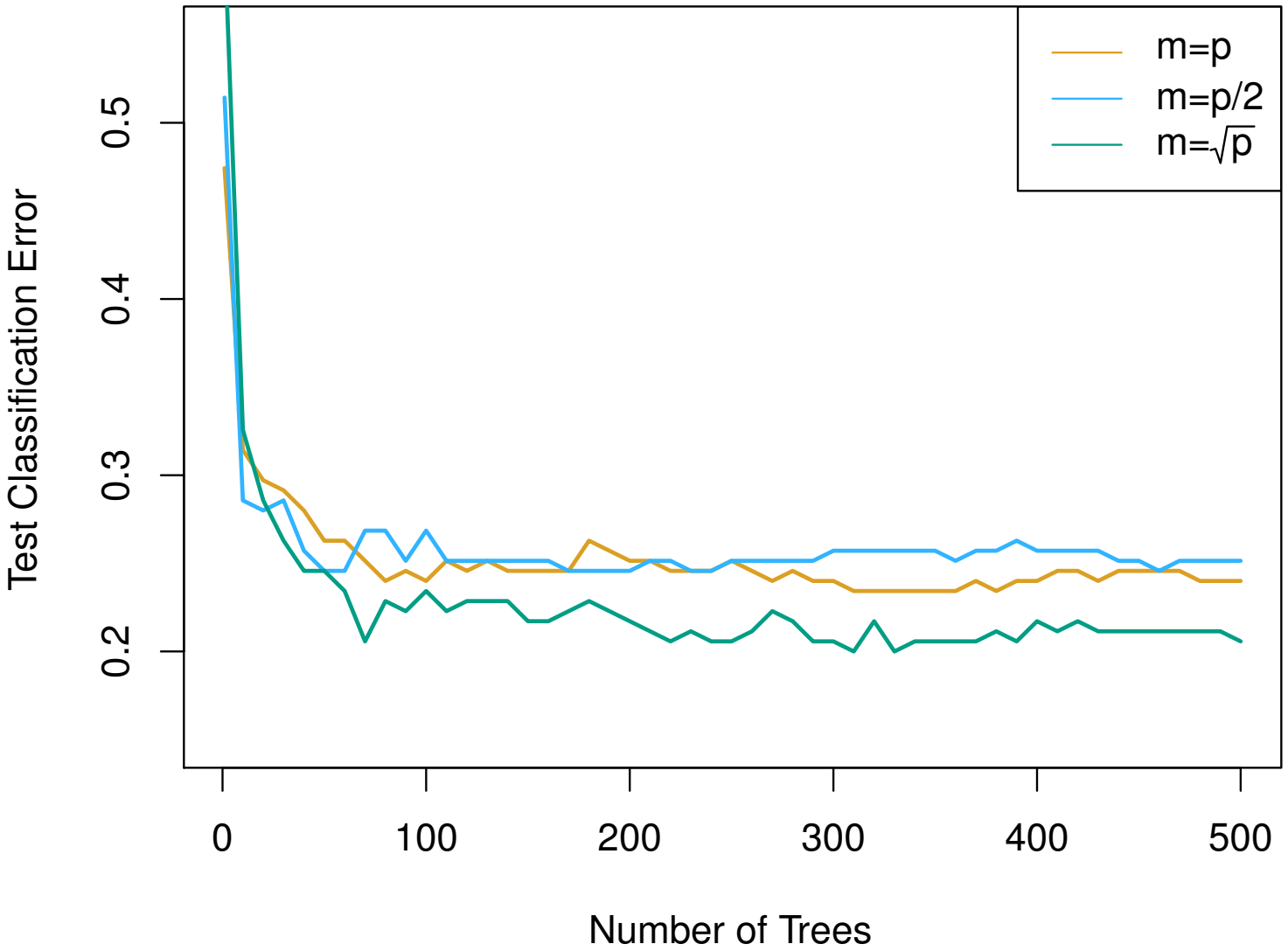


Fig 8.10 Random forests with different choices of m ($K = 15$ classes)

Gene expression example: RF vs Boosted trees

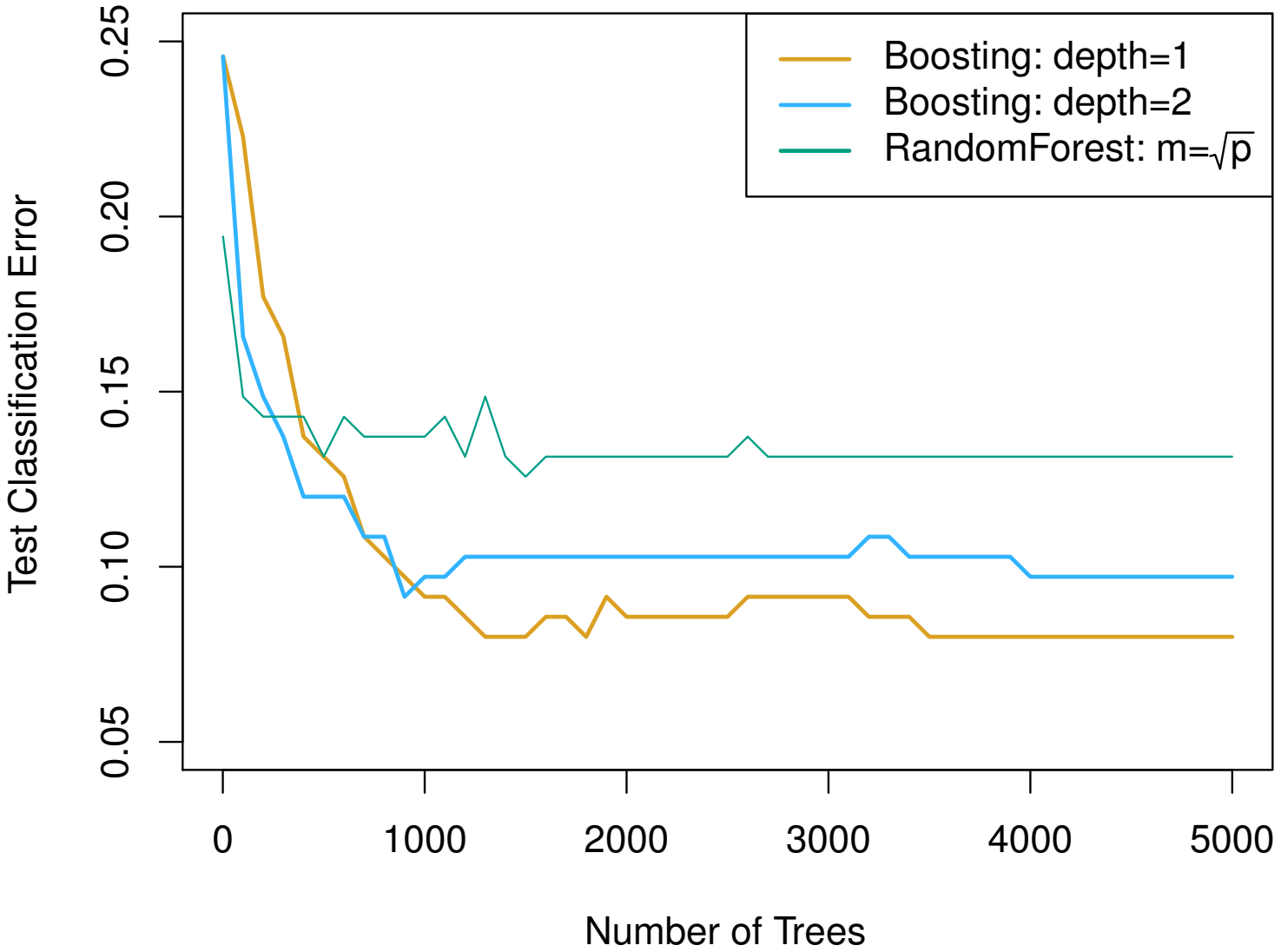
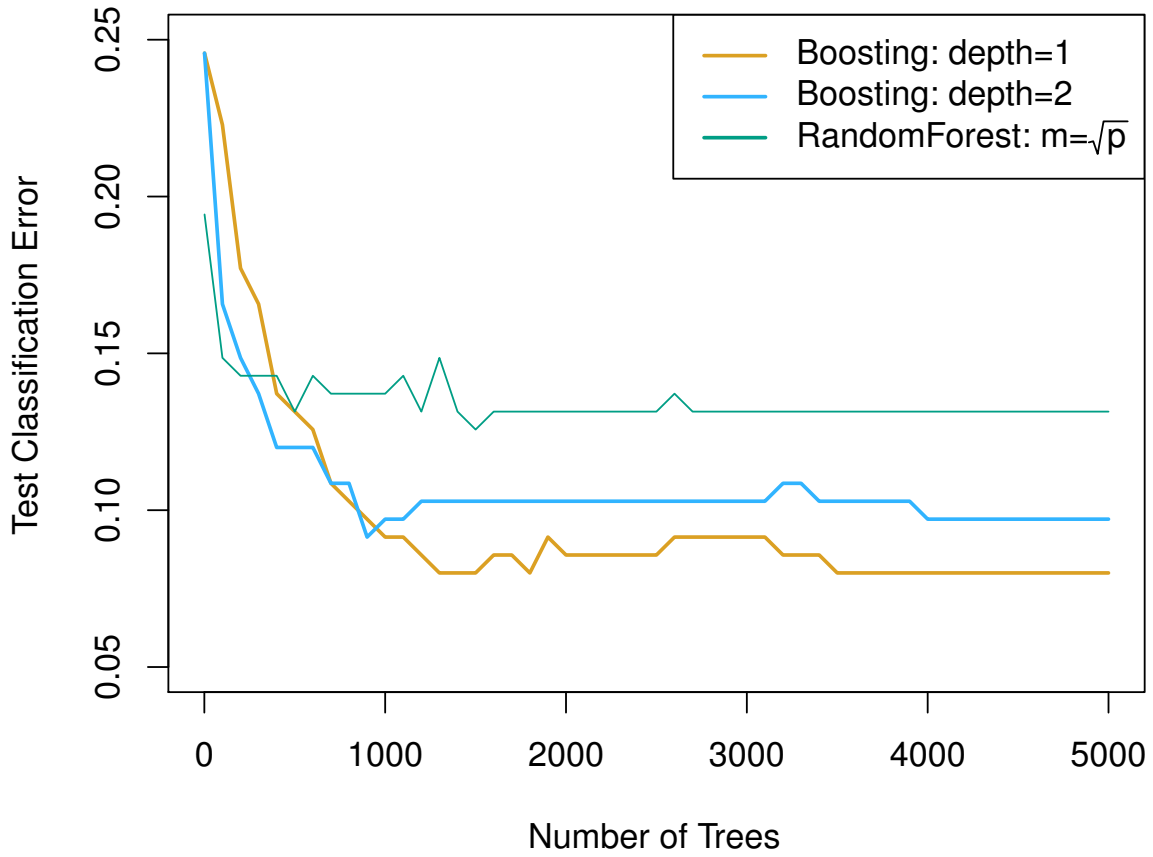


Fig 8.11 Random forests vs. Boosted trees ($K = 2$ classes)



- $K = 2$ class problem: *cancer* vs *non-cancer*
- Boosting with Depth-1 trees outperforms Depth-2 trees, and both outperform random forests…but standard errors are actually around 0.02, so differences aren't really statistically significant

Tuning parameters for boosting

- Number of trees, B: Boosting can overfit if B is too large, though overfitting happens slowly. Use cross-validation to select

- shrinkage parameter, λ: $0 < \lambda \ll 1$. This is sometimes called the *learning rate*. Common choices are $\lambda = 0.01$ or $\lambda = 0.001$. Small λ requires very large B to achieve good performance.

- Number of splits, d: $d = 1$, called *stumps*, often works well. This amounts to an additive model.
  - Often refer to d as the interaction depth: d splits can involve at most d variables

Email spam data (seen on Homework 4)

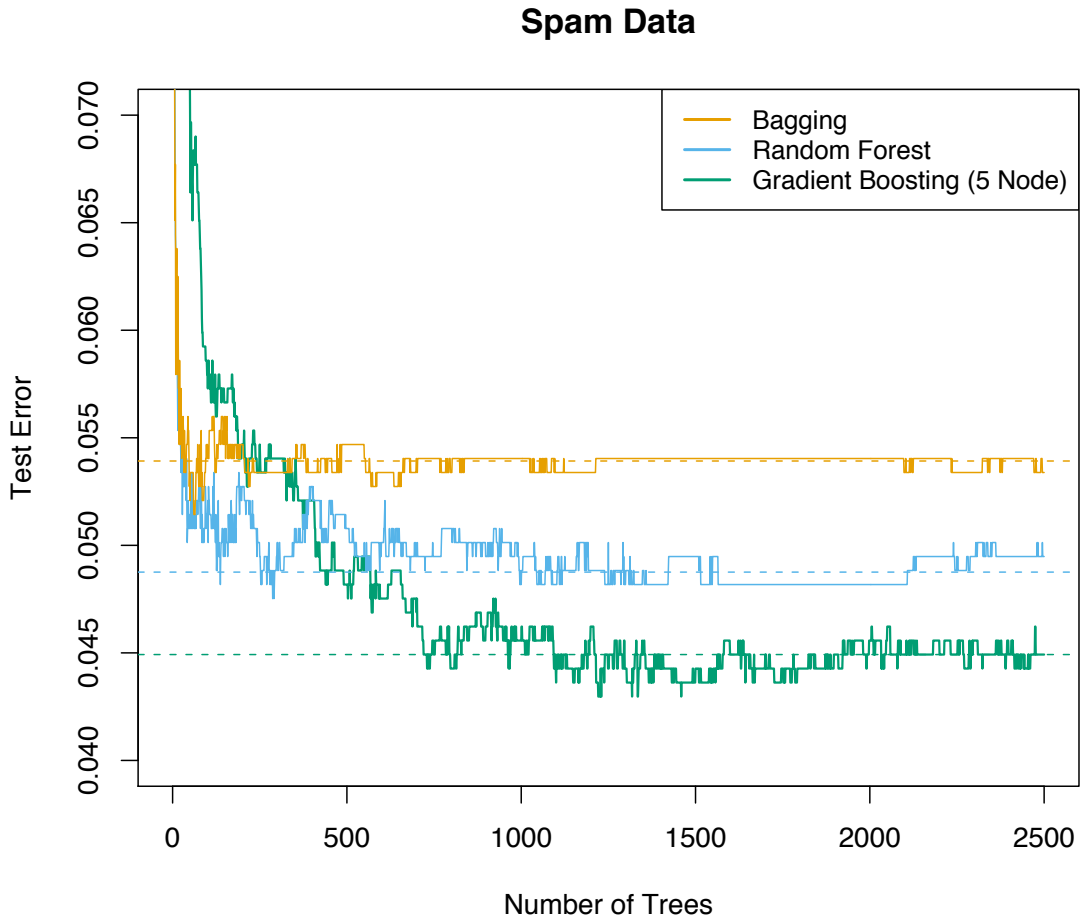


FIGURE 15.1. *Bagging, random forest, and gradient boosting, applied to the spam data. For boosting,* 5*-node trees were used, and the number of trees were chosen by* 10*-fold cross-validation (*2500 *trees). Each "step" in the figure corresponds to a change in a single misclassification (in a test set of* 1536*).*

Bagging, Boosting, Interactions

- Ensemble methods feel like *black boxes*: they make predictions by combining the results of hundreds of separate models

- Such models are able to capture complex interactions between predictors, which additive models are unable to do

- E.g., Suppose that your most profitable customers are young women and older men. A linear model would say:

$$\texttt{profit} \approx \beta_0 + \beta_1 I(\texttt{female}) + \beta_2 \texttt{age}$$

- This doesn't capture the interaction between `age` and `gender`

- Trees (and ensembles of trees) do a great job of capturing interactions

- Indeed, a tree with d splits can capture up to d-way interactions

Variable Importance

- While RF's and Boosted trees aren't interpretable in any meaningful sense, we can still extract some insight from them

- For instance, we can use variable importance plots to help answer the question: Which inputs have the biggest effect on model fit?

- There are two popular ways of measuring variable importance.

- Approach 1: For regression (resp. classification) record the total amount that the RSS (resp. Gini index) is decreased due to splits over a given predictor. Average this over all B trees.
  - A large value indicates an important predictor

- Approach 2: Randomly permute the values of the jth predictor, and measure how much this reduces the performance of your model (e.g., how much it increases MSE or accuracy)
  - A large drop in performance indicates an important predictor

- The `varImpPlot()` function in **R** calculates these quantities for you

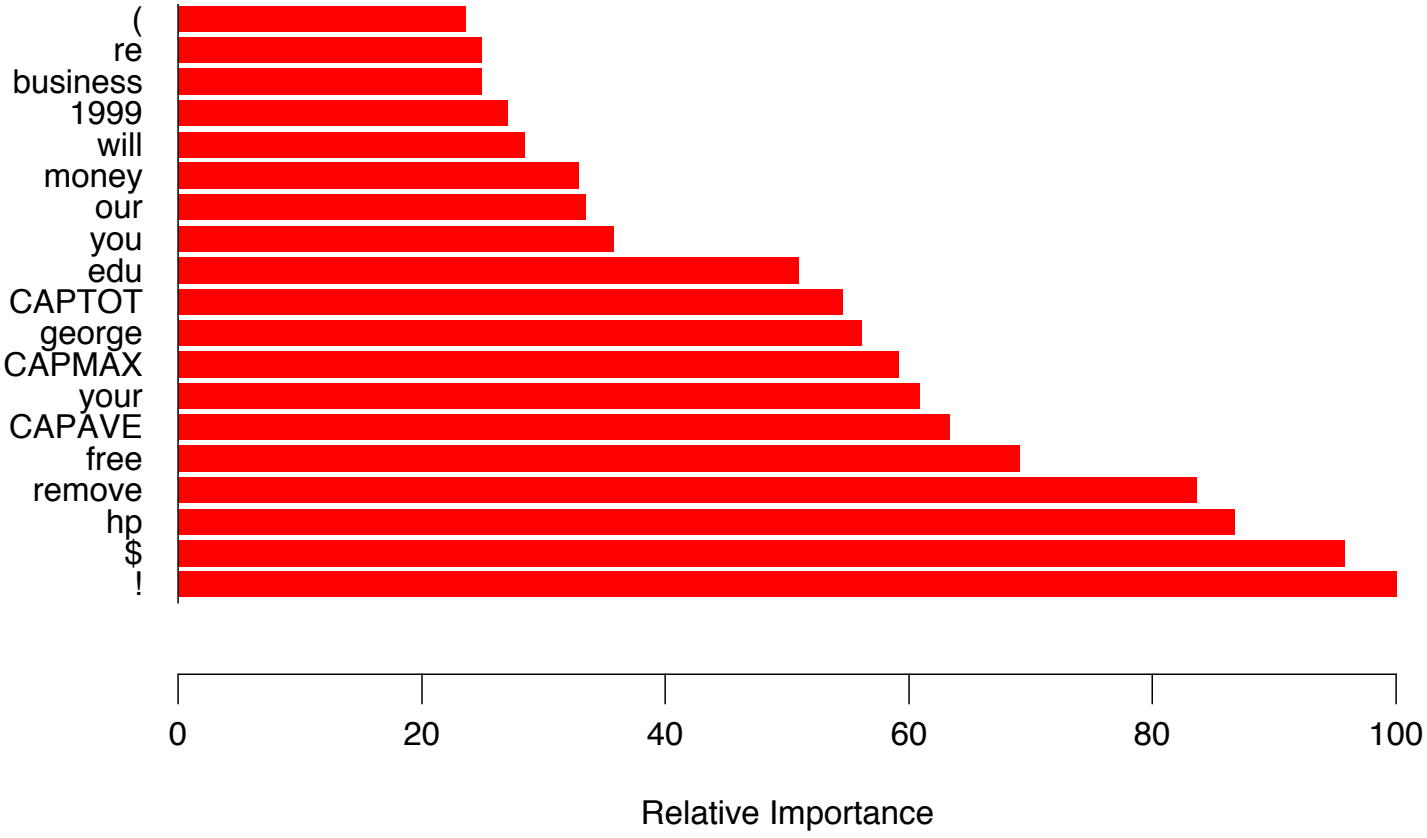


FIGURE 10.6. *Predictor variable importance spectrum for the* `spam` *data. The variable names are written on the vertical axis.*

- This helps us to see which variables are the most important…but it doesn't tell us how they affect the model. E.g., the frequency of ! is very important, but are emails with lots of !'s more likely to be spam or not spam?

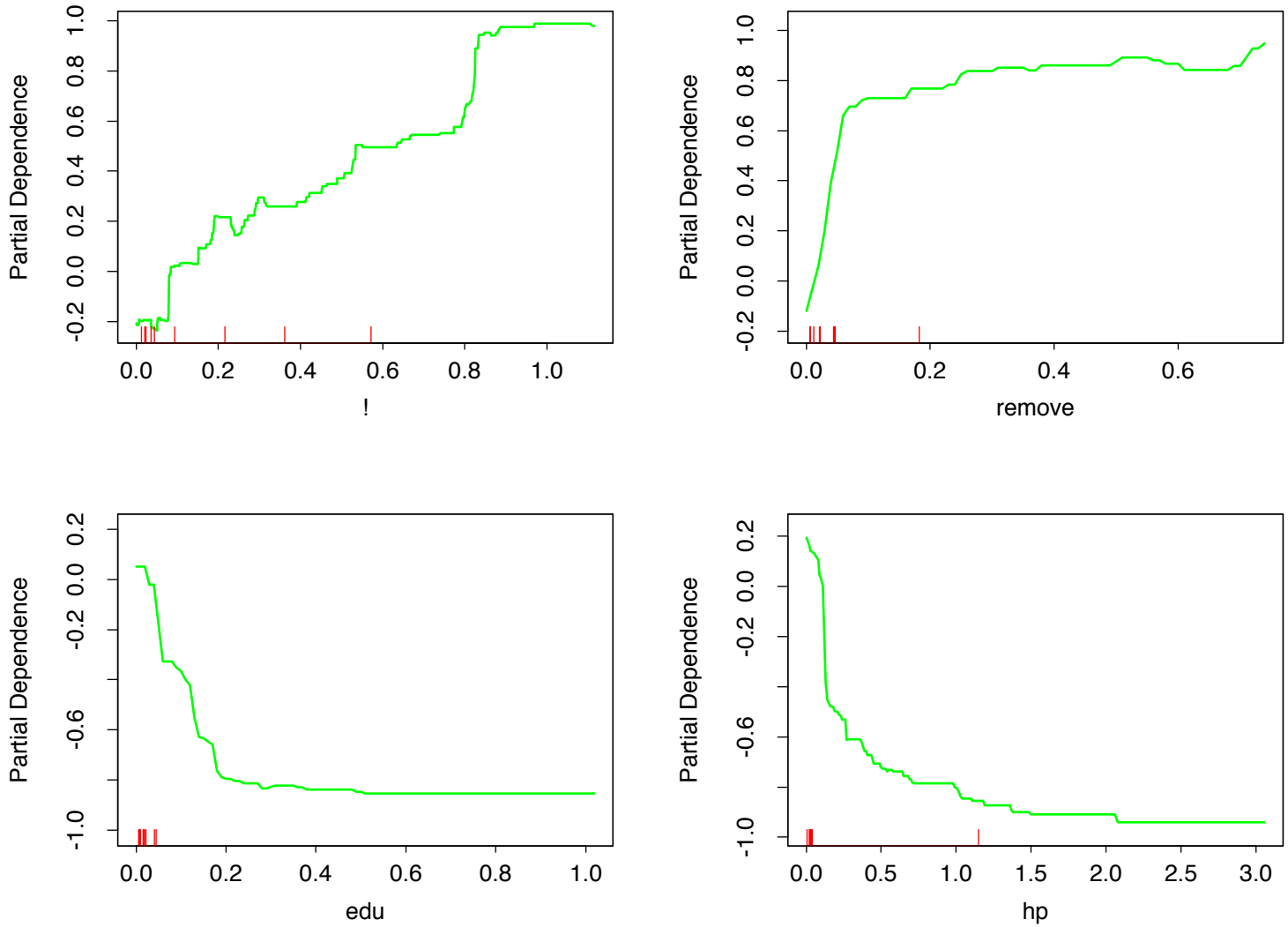


FIGURE 10.7. *Partial dependence of log-odds of* `spam` *on four important predictors. The red ticks at the base of the plots are deciles of the input variable.*

- You can get partial dependence plots by using the `partialPlot` command

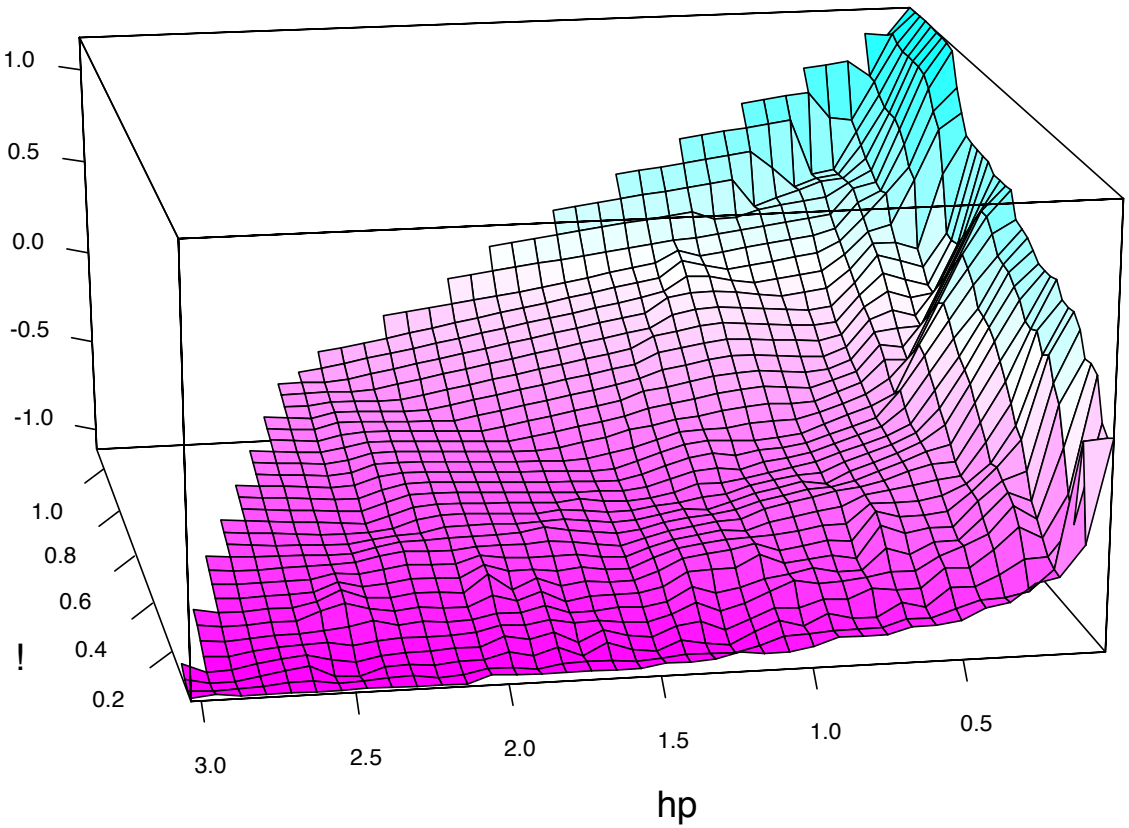


FIGURE 10.8. *Partial dependence of the log-odds of* `spam` *vs.* `email` *as a function of joint frequencies of* `hp` *and the character* `!`.

Bootstrap for SE estimation

- We introduced the Bootstrap for the purpose of bagging
- There's a more common use of bootstrapping: standard error estimation for complicated parameter estimates
- Commonly used to estimate the standard error of a coefficient, or to build confidence intervals
- Can be used to estimate uncertainty for very complex parameters, and in very complex sampling settings
  - We know how to do these things for Normal data, or when the Central limit theorem holds
  - Bootstrapping provides a way of estimating standard errors and building CI's even when the data generating distribution is non-Normal and the CLT cannot be expected to hold

A Toy Example: Asset Allocation

- Let X and Y denote the (log) returns of two financial assets
- We want to invest α of our money in asset X and $1 - \alpha$ of our money in asset Y
- We want to minimize the *risk* (variance) of our investment returns:

$$\mathrm{Var}(\alpha X + (1 - \alpha)Y)$$

- We're given 100 observations of daily returns $(x_1, y_1), \ldots, (x_{100}, y_{100})$
- In addition to estimating the best allocation (getting an estimate $\hat{\alpha}$), we also want to know the standard error of $\hat{\alpha}$
- If the SE of $\hat{\alpha}$ is large, this would mean that our investment strategy may be quite far from optimal

A Toy Example: Asset Allocation

- With some work, one can show that $\mathrm{Var}(\alpha X + (1-\alpha)Y)$ is minimized by

$$\alpha_{opt} = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}$$

  where $\sigma_X^2 = \mathrm{Var}(X)$, $\sigma_Y^2 = \mathrm{Var}(Y)$, $\sigma_{XY} = \mathrm{Cov}(X, Y)$

- We can use the data to calculate the *sample* variances of X and Y, along with a sample covariance.

- Thus we can estimate the optimal allocation strategy with

$$\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}}$$

- Now the tricky part: What is the standard error of $\hat{\alpha}$?

A Toy Example: Asset Allocation

Here's our estimate of the optimal asset allocation:

$$\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}}$$

- Suppose that we knew the data generating process for (X, Y) *exactly*
- We could then:
  1. Simulate a bunch of *new* data sets of 100 observations (say, do this 1000 times)
  2. Calculate *new* estimates $\hat{\alpha}_1, \hat{\alpha}_2, \ldots, \hat{\alpha}_{1000}$
  3. Estimate the standard error of $\hat{\alpha}$ by calculating the standard deviation of the estimates $\{\hat{\alpha}_r\}_{r=1}^{1000}$ from our simulated data:

$$\hat{\mathrm{SE}}(\hat{\alpha}) = \sqrt{\frac{1}{1000 - 1} \sum_{r=1}^{1000} (\hat{\alpha}_r - \bar{\alpha})}$$

where $\bar{\alpha} = \frac{1}{1000} \sum_{r=1}^{1000} \hat{\alpha}_r$

A Toy Example: Bootstrap Solution

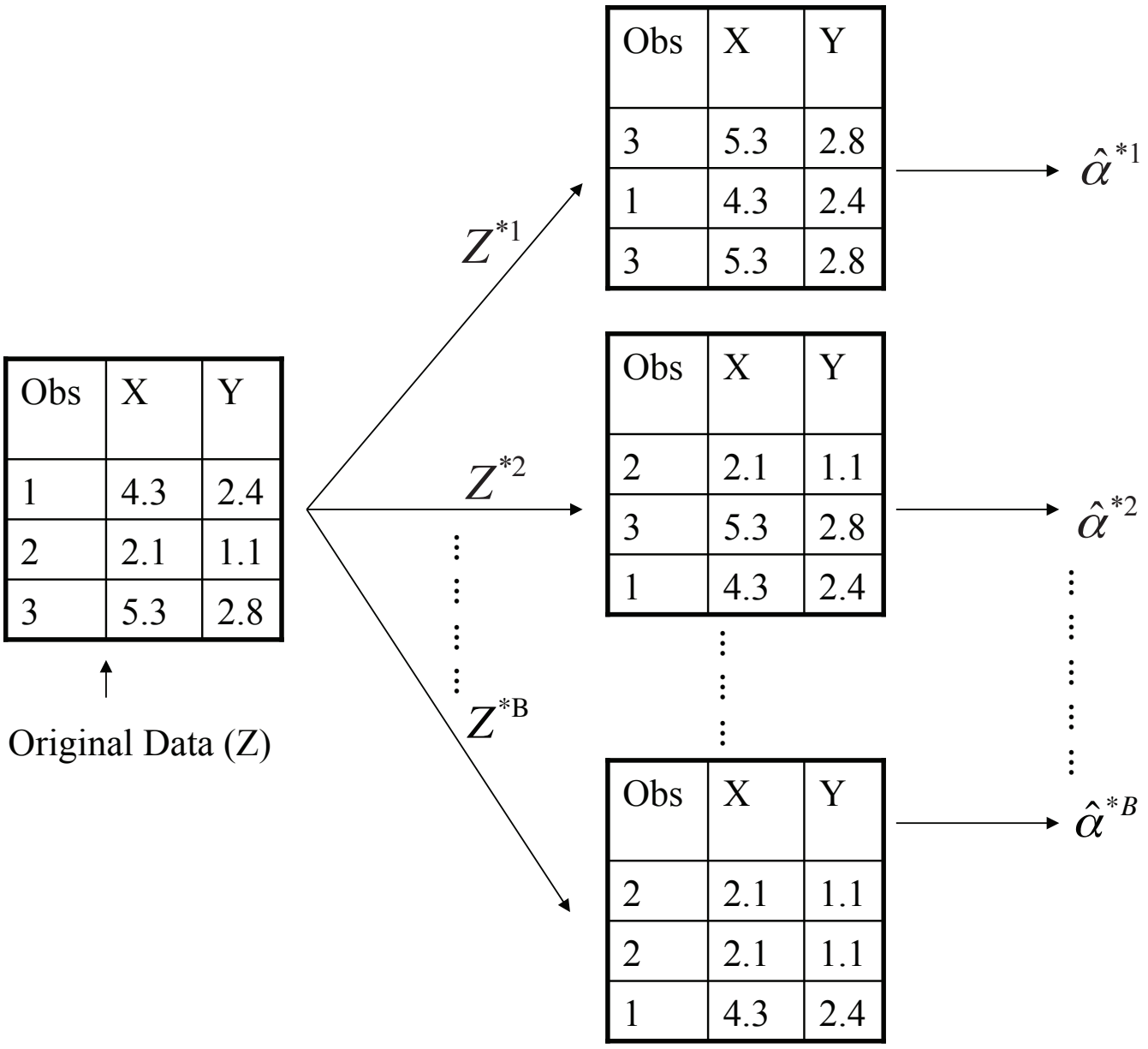
- Great! There's just one major problem…we do not know the distribution of X and Y exactly, so we can't simulate new batches of data

- Bootstrap approach: Let's try generating new data sets by resampling from the data itself…*Sounds crazy, right?*
  1. Get B new data sets $Z^{*1}, \ldots, Z^{*B}$, each by sampling 100 observations with replacement from our observed data (do this, say, $B = 1000$ times)
  2. Calculate new estimates $\hat{\alpha}^{*1}, \hat{\alpha}^{*2}, \ldots, \hat{\alpha}^{*B}$
  3. Estimate the standard error of $\hat{\alpha}$ by calculating the standard deviation of the estimates from our simulated data:

$$\hat{\mathrm{SE}}_B(\hat{\alpha}) = \sqrt{\frac{1}{B-1}\sum_{r=1}^{B}(\hat{\alpha}^{*r} - \bar{\alpha}^*)}$$

where $\bar{\alpha}^* = \frac{1}{B}\sum_{r=1}^{B}\hat{\alpha}^{*r}$

A Bootstrap Picture



How well did we do?

- When we know the data generating process (see p.188 of ISL), simulating 1000 data sets and calculating the standard errors of the corresponding $\hat{\alpha}$ estimates gives

$$\hat{\mathrm{SE}}(\hat{\alpha}) = 0.083$$

- Starting with a single data set of $n = 100$ observations and running the bootstrap procedure to resample $B = 1000$ data sets gives

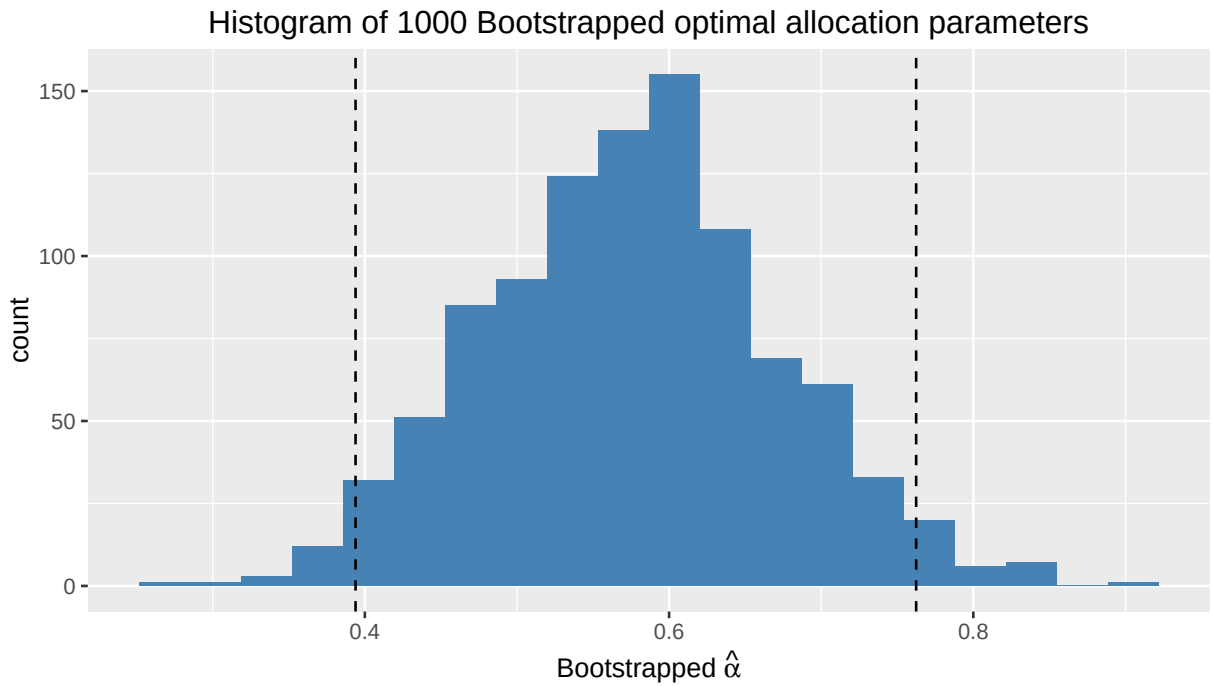
$$\hat{\mathrm{SE}}_B(\hat{\alpha}) = 0.087$$

- Amazing!

- Say we get $\hat{\alpha} = 0.576$. The estimated SE is non-negligible, so we know that there's still *a fair bit of uncertainty* in what allocation to choose. But the SE is small enough that choosing an allocation close to $\hat{\alpha} = 0.576$ seems like a reasonable thing to do.

Bootstrap Confidence Intervals

- The bootstrap procedure gives us B estimates $\hat{\alpha}^{*1}, \hat{\alpha}^{*2}, \ldots, \hat{\alpha}^{*B}$



- To form a $(1-\gamma)\cdot 100\%$ CI for α, we can use the $\gamma/2$ and $1-\gamma/2$ percentiles of the bootstrapped estimates
- In this simulation, we would get a 95% CI of $[0.39, 0.76]$

Acknowledgements

All of the lectures notes for this class feature content borrowed with or without modification from the following sources:

- 36-462/36-662 Lecture notes (Prof. Tibshirani, Prof. G'Sell, Prof. Shalizi)
- 95-791 Lecture notes (Prof. Dubrawski)
- *An Introduction to Statistical Learning, with applications in R* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani
- *Applied Predictive Modeling*, (Springer, 2013), Max Kuhn and Kjell Johnson