Notes 2: Introduction to Trees and Forests

1. Introduction

Introduction to Trees

Trees:

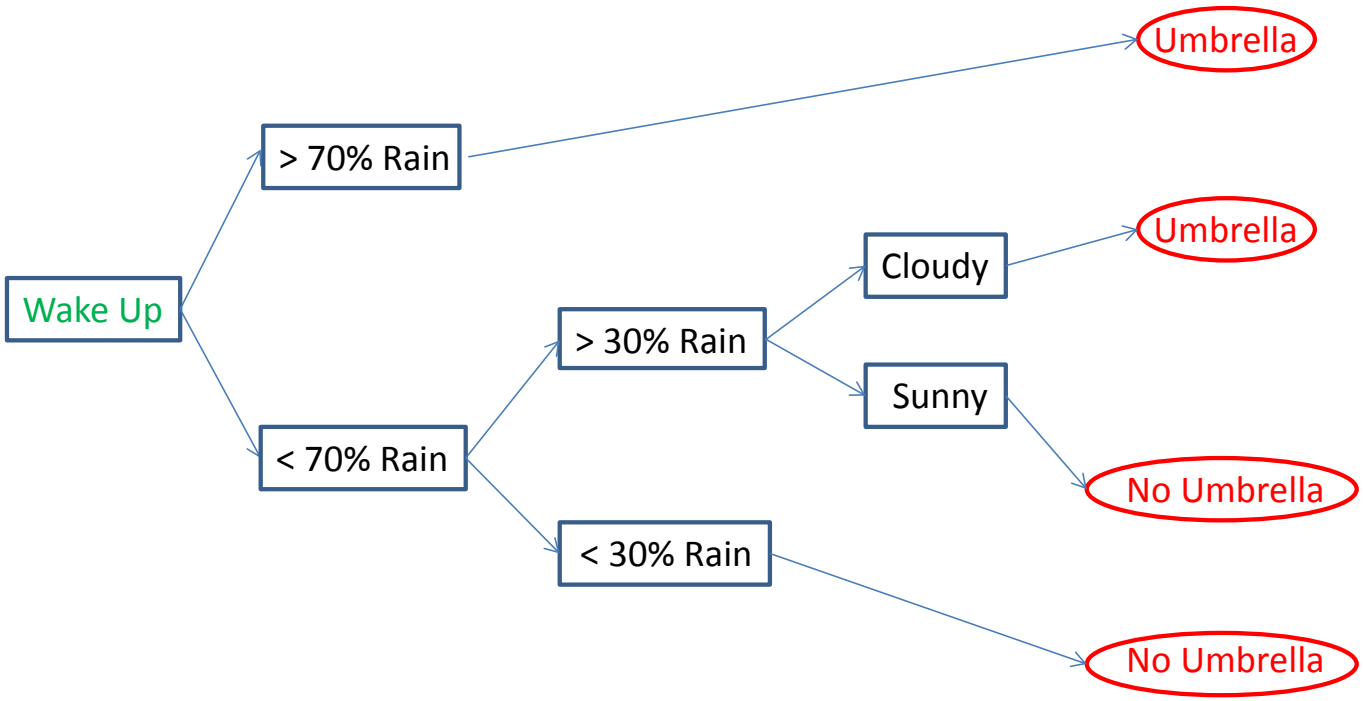
- tree-based methods widely used in data-mining applications
  - CART: Classification and Regression Trees
  - Random Forests
  - BART: Bayesian Adaptive Regression Trees
  - MARS: Multivariate Adaptive Regression Splines
  - Treed linear models, treed Gaussian processes, ...
- "Of all the well-known learning methods, decision trees come closest to meeting the requirements for serving as an off-the-shelf procedure for data mining." (Elements, p. 352)
- Gaining traction in econometrics - e.g.
  - Imbens, G. and S. Athey (2015), "Machine Learning Methods for Estimating Heterogeneous Causal Effects" (working paper)
  - Bajari, P., D. Nekipelov, S. Ryan, and M. Yang (2015), "Demand Estimation with Machine Learning and Model Combination" (working paper)
  - Asher, S., D. Nekipelov, P. Novosad, and S. Ryan (2016), "Classification Trees for Heterogeneous Moment-Based Models" (working paper)

Introduction to Trees

- Good features:
  - flexible (handles nonlinearity, interactions, etc. without prespecification)
  - invariant to monotone transformations of inputs
  - can deal with irrelevant inputs (i.e. variable selection)
  - relatively fast
  - scales to large problems (e.g. Taddy, M., C. Chen, J. Yu, and M. Wyle (2015), "Bayesian and empirical Bayesian forests," working paper)
  - relatively intuitive and interpretable
- Bad features:
  - Simple trees don't predict as well as some other methods and don't naturally lead to continuous models
  - Increase predictive performance with bagging and boosting but lose some interpretability

Tree Logic

A decision tree:



- Basic logic: Use a series of steps to reach a decision
- Each decision (blue box) is a node
- The final prediction (red oval) is a leaf node (or simply leaf)

2. Regression Trees

Building a Regression Tree: CART

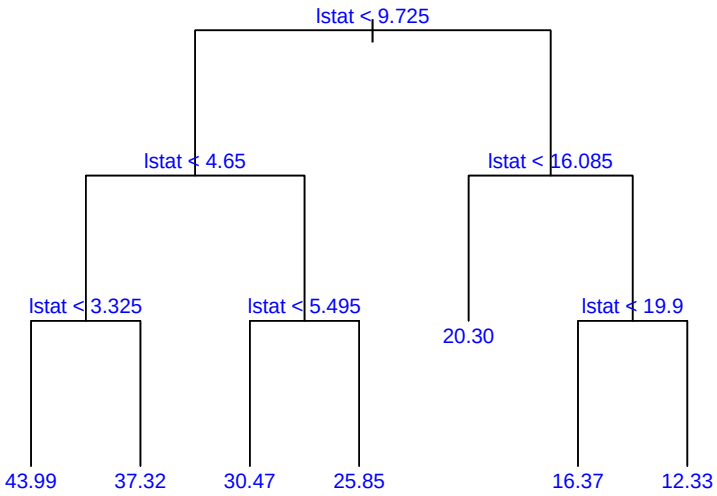
CART:

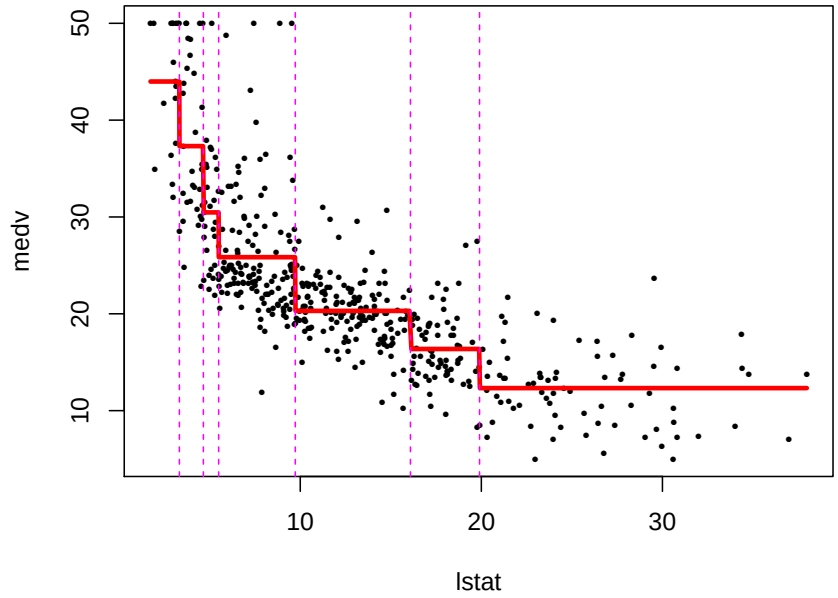
- Basic idea: Use the data to form a decision tree using recursive binary partitions
- Step 1:
  - Let $R_1(j,s) = \{X|X_j < s\}$ and $R_2(j,s) = \{X|X_j \geq s\}$
  - Choose j and s as
    $\arg\min \sum_{i:x_i \in R_1(j,s)} (y_i - \widehat{y}_{i,R_1})^2 + \sum_{i:x_i \in R_{i,2}(j,s)} (y_i - \widehat{y}_{i,R_2})^2$
    - $\widehat{y}_{i,R_k}$ is the forecast of y_i using data in region R_k (usually just the sample mean)
- Step 2:
  - Split the data into the two subregions from Step 1.
  - Repeat the splitting process on each subregion, taking the split that minimizes loss
- etc...
- Stop splitting when some stopping criterion reached (often a minimum node size - e.g. no less than 5 observations)

Simple Example: Boston Housing Data

Let's look at a tree in Boston housing data. $y = medv$ and $x = lstat$ as before. (Constrained to have only 7 terminal nodes)

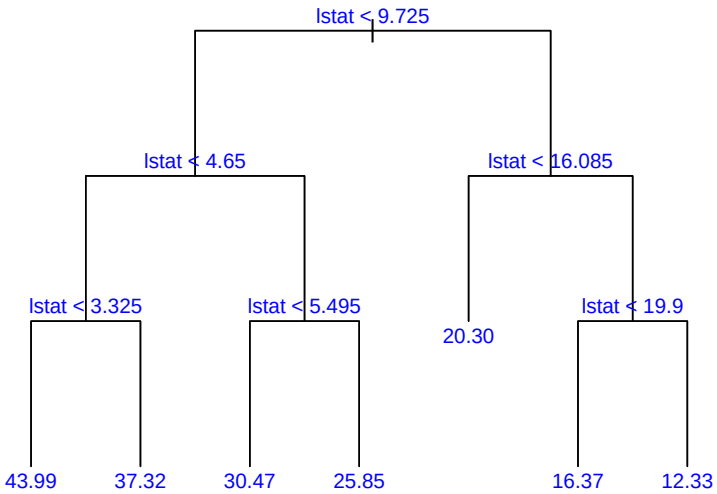
Code: SimpleTree_Example1.R

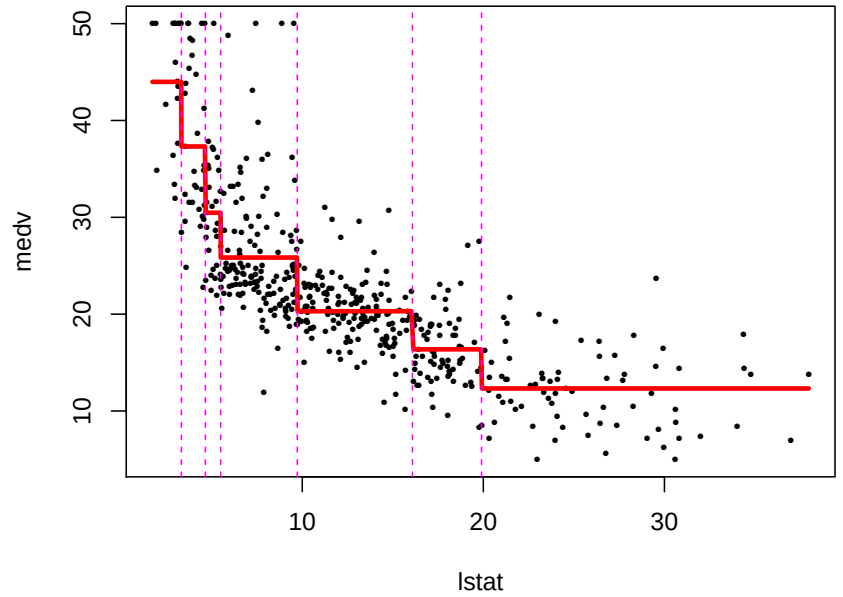




Left panel: Dendrogram
Right panel: Fitted model

Simple Example: Boston Housing Data

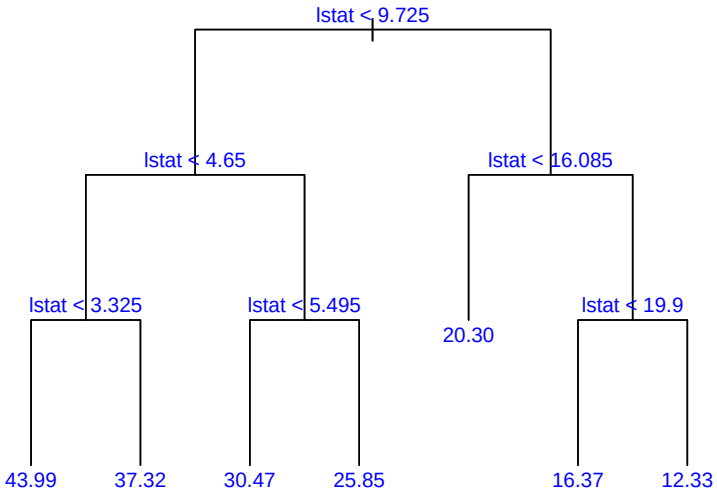


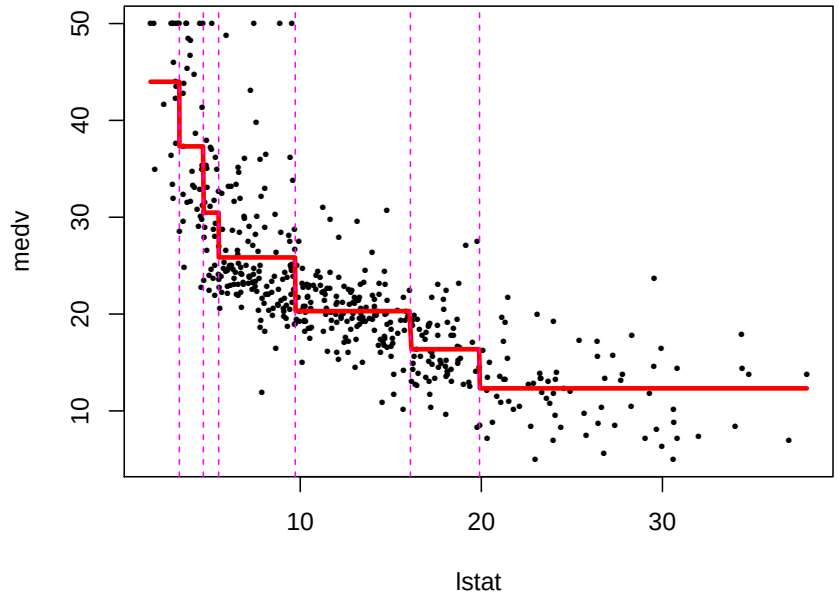


What's going on?

- At each interior node, decision rule $\{x < c\}$
  - Left if $x < c$, right if $x \geq c$
- Each observation sent down tree until it hits terminal node/leaf
- Set of leaves gives a partition of x-space into disjoint regions

Simple Example: Boston Housing Data





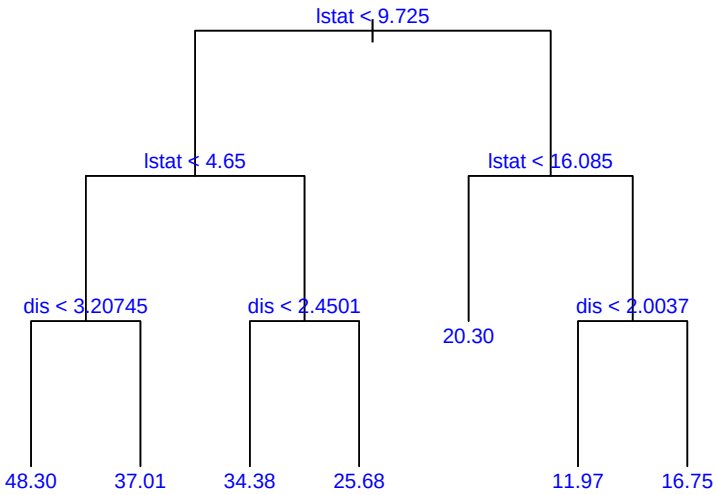
Estimated function just step function with steps given by tree partition

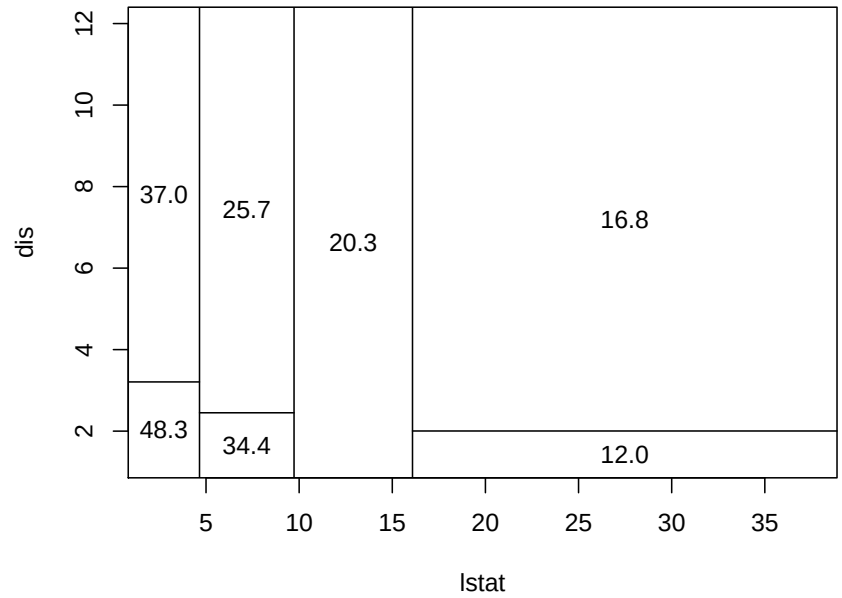
Forecasting easy, just find interval to which new observation belongs and take sample mean in that region

Simple Example: Boston Housing Data

Let's try two variables. $y = medv$, $x_1 = lstat$, $x_2 = dis$ (Constrained to have only 7 terminal nodes)

dis = weighted mean of distances to five Boston employment centers



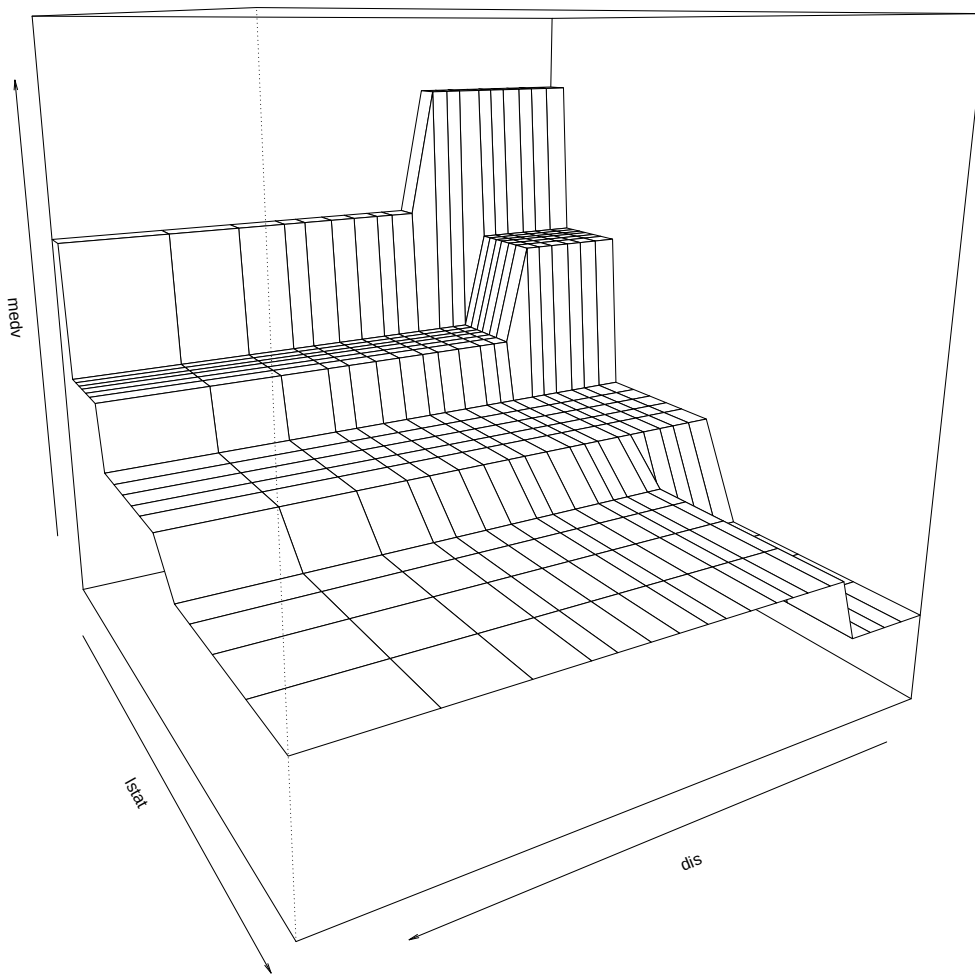


Right panel shows partition of X-space
Note presence of interaction (Automatic interaction detection - AID)

Simple Example: Boston Housing Data

Again estimated function just step function with steps given by tree partition

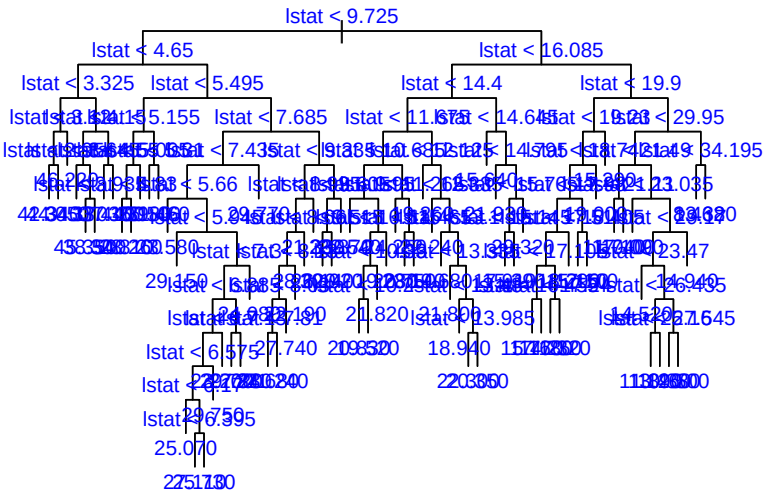
Can see the presence of interactions

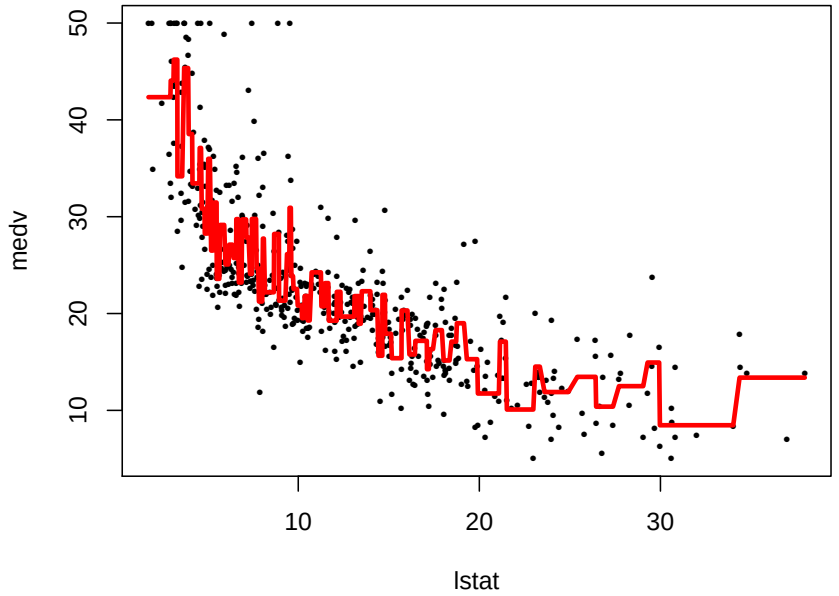


Simple Example: Boston Housing Data

Big tree (with just $x = lstat$):

- grown until minimum 5 observations per node
- splitting as long as split improves loss by at least .0001 (proportionally)





Which Tree?

A tree is just a flexible model for fitting data

Just as with other nonparametric/high-dimensional models, key question is how complex to make the tree

Usually measure complexity of a tree with number of leaves (terminal nodes)

Can fit any data set arbitrarily well with enough leaves (and a continuous input) - easy to overfit

For prediction, trade-off bias and variance as before

Minimization Problem

Choice of tree represented as minimization problem:

$$\widehat{T}_\alpha = \arg\min_T Q(T, y) = \arg\min_T L(T, y) + \alpha|T|$$

- $L(T, y)$ is loss of using tree T for forecasting data y
  - E.g. sum of squared residuals $\sum_{i=1}^{n}(y_i - T(x_i))^2$
- $|T|$ is the number of leaves - measures complexity

Minimization Problem

Choice of tree represented as minimization problem:

$$\widehat{T}_\alpha = \arg\min_T Q(T, y) = \arg\min_T L(T, y) + \alpha|T|$$

- α is the "cost" for complexity - regularization/penalization/"complexity cost penalty" parameter
  - With $\alpha = 0$, would choose tree that "perfectly fits" data - overfitting
  - Want to trade-off within-sample fit with complexity to produce generalizability - Regularization
  - α is key parameter that needs to be chosen (plays same role as λ in penalized estimation)
    - α is chosen by K-fold CV or a validation exercise

Minimization Algorithm

How do we actually do the minimization?

Idea of trees is simple, but trees are large and complex when viewed as variable in optimization problem

Key to tree modeling is success of simple, heuristic algorithm to tree fitting (cost-complexity pruning).

Minimization Algorithm

1. Grow Big

Use greedy, recursive binary split forward search to build a big tree

Want to grow big initially because seemingly worthless split high in the tree might really help lower down

i. Start with the tree that is a single node.

ii. At each bottom node, search over all possible decision rules to find the one that gives the biggest decrease in loss (increase in fit).

iii. Grow a big tree, stopping (for example) when each bottom node has 5 observations in it.

Minimization Algorithm

2. Prune Back

i. Recursively, prune back the big tree from step 1.

ii. Given a current pruned tree, examine every pair of bottom nodes (having the same parent node) and consider eliminating the pair.

Prune the pair the gives the smallest increase in loss.

This gives a sequence of subtrees of our initial big tree that must contain $\widehat{T}_\alpha$ (if initial tree big enough).

iii. For a given α, choose the subtree of the big tree that has the smallest $Q(T, y)$.

Choice of α

Now all we have to do is choose α.

Usual approach is by performance on a validation sample or K-fold CV

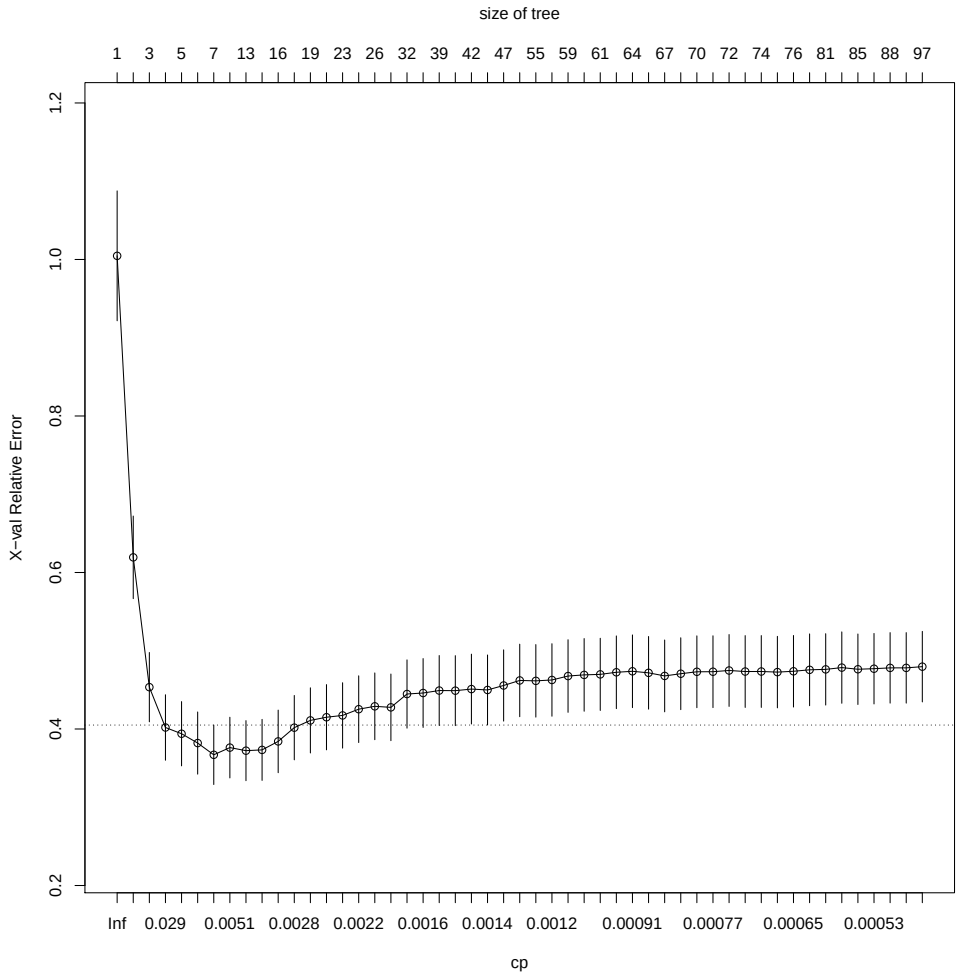
Simple Example: Boston Housing Data

Housing data again with $y = medv$ and $x = lstat$.

Code:

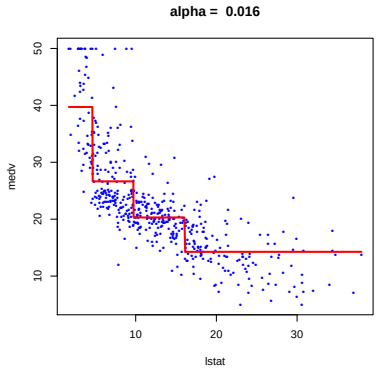
SimpleTree_Example3.R

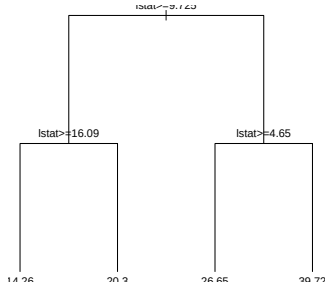
10-Fold CV (relative to sample mean) as function of (transformed) α:



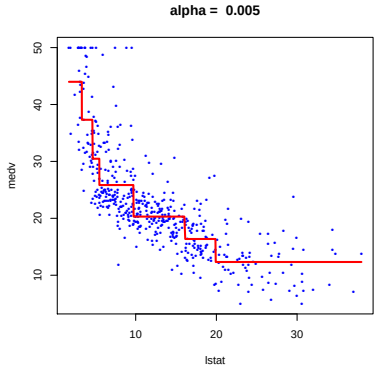
Simple Example: Boston Housing Data

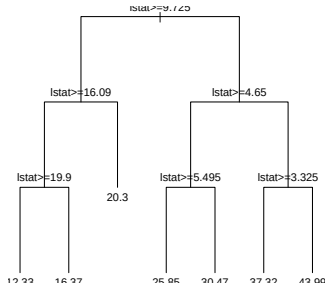
Three different tree fits for three different values of α



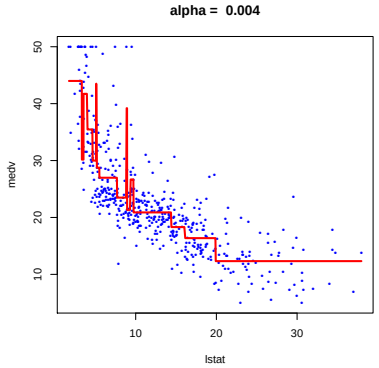


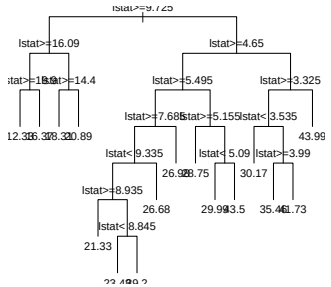
The smaller α, the smaller cost for complexity and the bigger the tree.
Top tree subset of middle tree which is a subset of bottom tree ($\alpha_{top} > \alpha_{middle} > \alpha_{bottom}$)





Middle tree corresponds to CV-min α





Increase p Simulation - Tree

Code: SimpleTreeIncreaseP_Example4.R:

	$p = 1$		$p = 5$		$p = 20$		$p = 50$	
	CV	MSE	CV	MSE	CV	MSE	CV	MSE
OLS	0.056	0.064	0.058	0.069	0.076	0.074	0.134	0.119
LASSO	0.056	0.064	0.057	0.066	0.058	0.066	0.059	0.070
TREE	0.067	0.093	0.074	0.107	0.089	0.111	0.080	0.114

10-Fold CV for Tree (not LOOCV)

Increase p Simulation - Tree

	$p = 90$		$p = 100$		$p = 200$	
	CV	MSE	CV	MSE	CV	MSE
OLS	0.760	0.716	1.337	3.011	0.798	0.554
LASSO	0.060	0.070	0.059	0.071	0.059	0.071
TREE	0.103	0.117	0.121	0.108	0.106	0.108

Simple trees a little worse LASSO

People don't usually use simple trees (boosting, bagging, and random forests)

Design tailor-made for LASSO to do very well

3. Bagging, Boosting, and Random Forests

Bagging

Bootstrap aggregation (bagging):

- Breiman, L. (1996), "Bagging Predictors," *Machine Learning* 26: 123-140.
  - Many extensions available
- Example of "ensemble learning": using multiple methods to get better results than any individual
- Idea:
  - fit lots of noisy, low bias models
  - average them together to reduce variability

Bagging Algorithm

Bagging straightforward - proceeds by bootstrap.

- For $b = 1$ to B
  - Draw a bootstrap sample $\{y_i^b, x_i^b\}_{i=1}^n$
  - Estimate model using bootstrap sample $\rightarrow \widehat{f}_b(\cdot)$
- Bagging estimator: $\widehat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \widehat{f}_b(x)$

Comments:

- B is user chosen number of bootstrap reps
- Choose B big enough that procedure has "settled down"
- Typically choose tuning parameter "too small" but less sensitive to choice in estimating model
- Only interesting when $\widehat{f}(x)$ is nonlinear or adaptive in the data
  - E.g. Would have $\frac{1}{B}\sum_{b=1}^{B} x_i'\widehat{\beta}_b \rightarrow x_i'\widehat{\beta}_{OLS}$ in linear model with OLS estimate of parameter

Out-of-Bag Error Estimation

Useful feature of bagging:

With careful accounting, get an estimate of out-of-sample performance for free!

Out-of-Bag (OOB) Error Estimation:

- Sampling with replacement $\Rightarrow$ Not all observations used at each replication b
- Can form an out-of-sample prediction for each observation not used at replication b (called "out-of-bag" observations) along with estimating $\widehat{f}_b$
- Average predicted OOB responses to get out-of-sample prediction for i^{th} observation
- Use these predictions to form OOB loss (e.g. OOB MSE)
- OOB MSE $\rightarrow$ leave-one-out CV as $B \rightarrow \infty$

Bagging Trees

Bagged Trees:

- Use bootstrap to generate new data
- Fit a large/flexible model in each bootstrap sample
  - A tree with lots of leaves - fast
- Aggregate over bootstrap replications to get "signal" and kill excess variability
  - Take average prediction for "continuous" y
  - Take average $\widehat{p}$ for discrete y (not so common)
  - Take majority vote for discrete y (most common in software anyway)
- Use out-of-bag (OOB) error to validate model

Need B big enough for things to stabilize

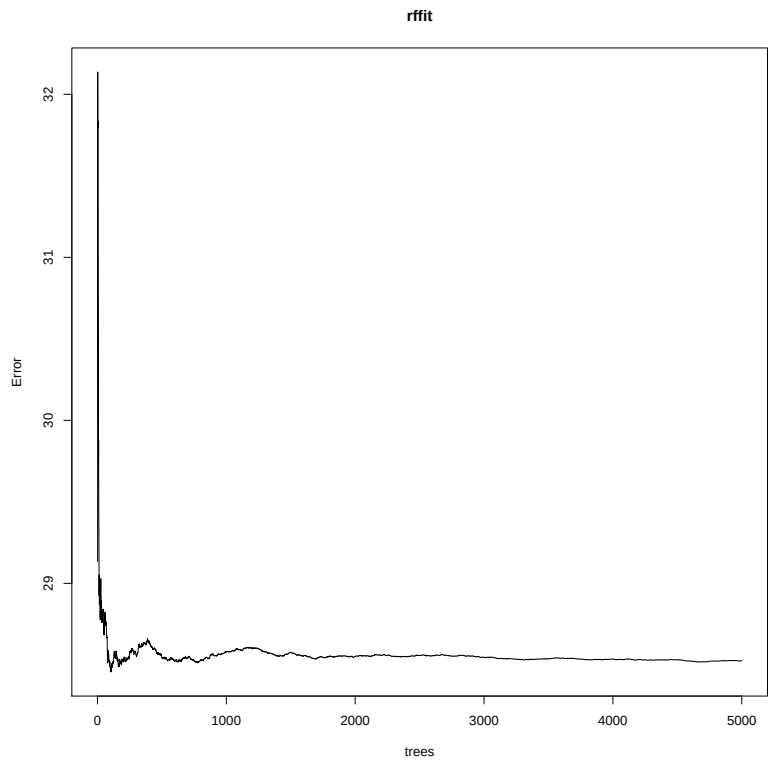
- Cost of big B is computation
- Err on side of big B

Simple Example: Boston Housing Data

Housing data again with $y = medv$ and $x = lstat$ - Bagging trees with 10 leaves.

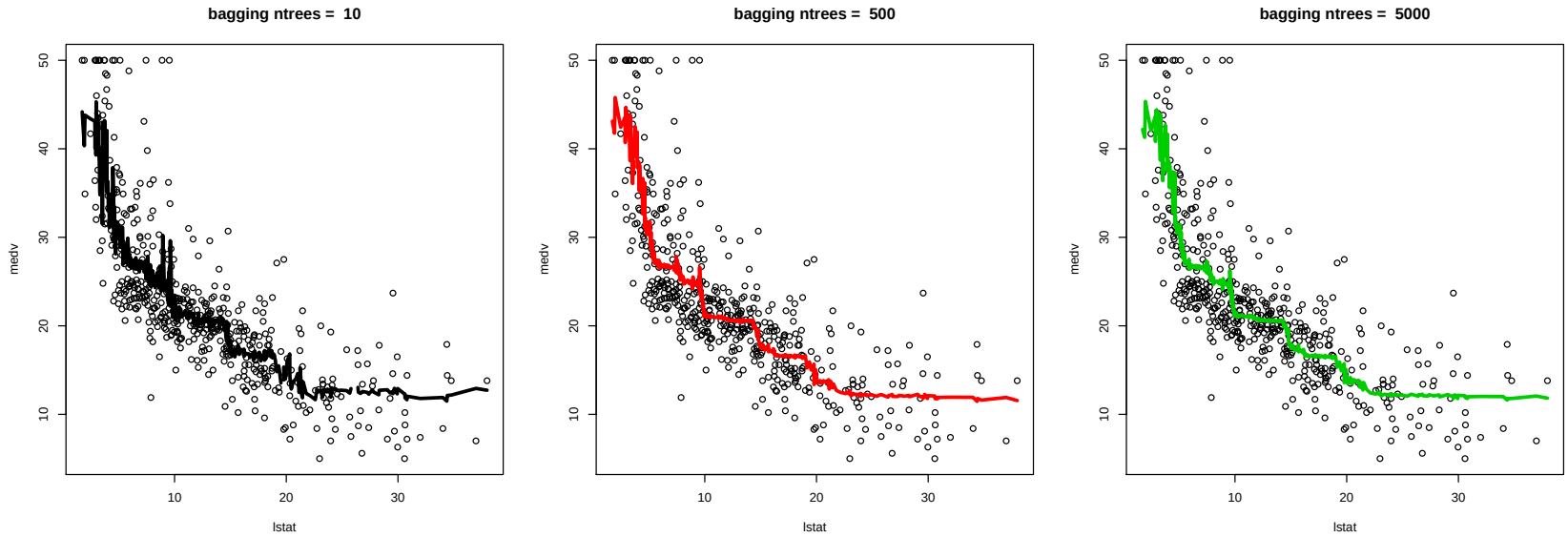
Code: SimpleTree_Example6.R

OOB Error estimates (pretty stable with a couple thousand trees):



Simple Example: Boston Housing Data

Fitted models for different forest sizes:



Note that although base model is a tree, no longer getting a step function

Random Forests

To get to "full" random forests, add one more layer of randomization

- Don't consider all p available predictors for each tree
- Select a subset of regressors to use in fitting model in each bootstrap draw
  - Use only $m < p$ regressors; common recommendation is $m = \sqrt{p}$ for classification and $m = p/3$ for regression
- Forces tree to move around more when there are lots of regressors
- Explore a richer set of models in tree space, allow moderate predictors to come through
- The "key" variables should come out consistently though
- Bagging is random forests with $m = p$
- Only bother with this if p really big

Boosting

Boosting:

- Schapire, R. (1990), "The strength of weak learnability," *Machine Learning*, 5, 197-227; Friedman, J., T. Hastie, and R. Tibshirani (2000), "Additive logistic regression: a statistical view of boosting (with discussion)," *Annals of Statistics*, 28, 337-407.
  - Many extensions - popular to combine boosting and bagging
- Another example of "ensemble learning"
- Idea:
  - fit lots of small, low variance "weak learners" (e.g. models that don't forecast very well)
  - aggregate them slowly to improve forecasting
  - differs from other approaches in forcing "slow" learning

Boosting Regression Trees

L_2 boosting for regression trees:

1. Initialize model - $f_0(x) = 0$
2. For $m = 1$ to M
   - Compute residuals - $r_{im} = y_i - f_{m-1}(x_i)$ for $i = 1, ..., n$
   - Fit tree for predicting "residuals" with $d + 1$ terminal nodes $\rightarrow h_m(x)$
   - Update your model to $f_m(x) = f_{m-1}(x) + \lambda\, h_m(x)$ for shrinkage parameter $\lambda \leq 1$
3. Prediction model is $f_M(x)$

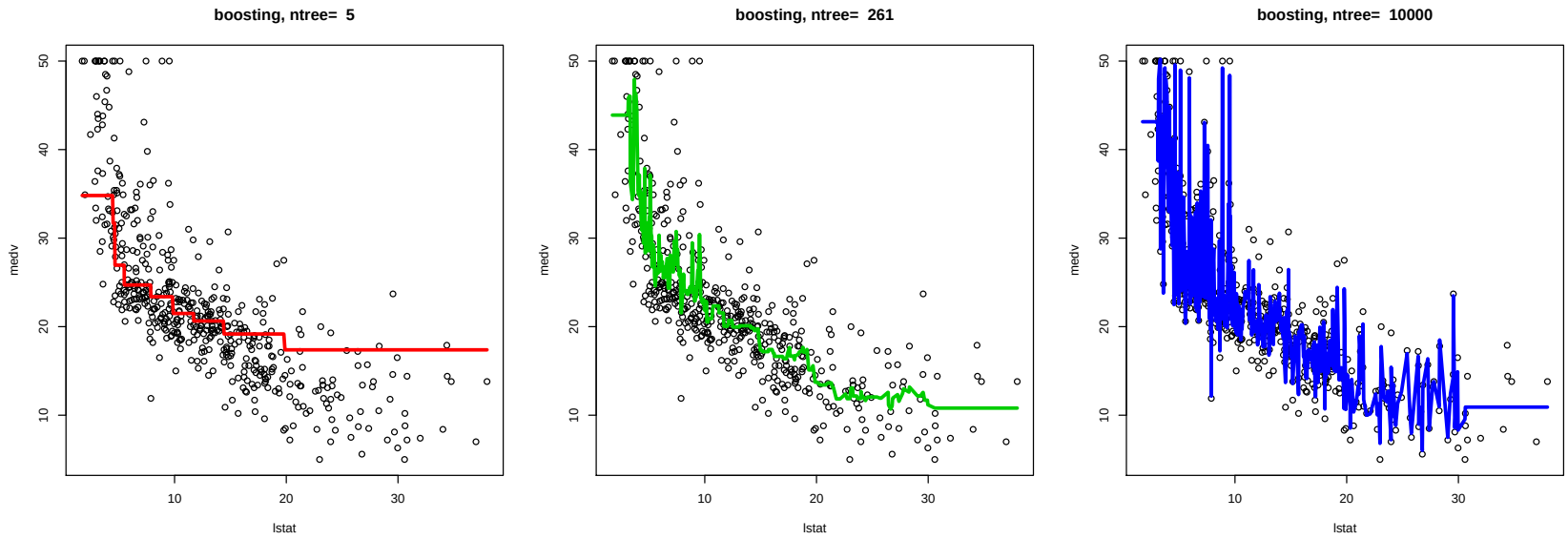
Tuning parameters: λ, d, M

- λ small (.01 or smaller)
- d small (1-5) - 1 produces additive model
- M pretty big - monitored with CV

Simple Example: Boston Housing Data

Housing data again with $y = medv$ and $x = lstat$ - Boosting with $d = 2$, $\lambda = .01$ (and bagging).

Code: SimpleTree_Example8.R



Middle one minimizes OOB error

Simulation: Nonlinear Model Design

Simulation Design:

$$Y = \exp(-X_1/2) + .75X_1 1(X_1 > 0) + .1(\sum_{j=2}^{5} X_j) + .1(X_2 - X_3 + X_4 - X_5)^2 + .1\varepsilon$$

- $(X_1, ...X_p, \varepsilon)' \sim N(0, I_{p+1})$
- $n \in \{100, 200\}$, $p \in \{1, 5, 20, 50, 100, 200\}$

Look at OLS, K-NN, LASSO, Regression Tree, Random Forest, Boosted Regression Tree

Increase p simulation - Nonlinear Model Methods

- OLS, 5-fold CV reported
- K-NN, K selected by 5-fold CV
- LASSO, penalty parameter (λ) selected by 5-fold CV
- Regression Tree, Cost-complexity parameter (α) selected by 5-fold CV
- Random Forest, $m = \sqrt{p}$, 15 leaves, $B = 5000$, OOB Error estimate reported (NOT CV)
- Boosted Regression Tree, $d = 2$, shrinkage (λ) of .01, number of trees selected by 5-fold CV
- In principle, could/should further tune some of these smoothing parameter choices

Increase p Simulation - Nonlinear Model (N = 100)

Code: IncreasePNonlinearN100_Example10.R,

	$p = 1$		$p = 5$		$p = 20$	
	CV	MSE	CV	MSE	CV	MSE
OLS	0.439	0.552	0.518	0.531	0.657	0.601
KNN	0.281	0.431	0.238	0.331	0.383	0.525
LASSO	0.420	0.577	0.416	0.577	0.424	0.577
TREE	0.331	0.460	0.415	0.577	0.355	0.465
FOREST	0.324	0.421	0.244	0.336	0.336	0.451
BOOST.TREE	0.323	0.443	0.313	0.323	0.380	0.429

Tree based methods competitive or dominant across the board

Increase p Simulation - Nonlinear Model (N = 100)

Code: IncreasePNonlinearN100_Example10.R,

	$p = 50$		$p = 100$		$p = 200$	
	CV	MSE	CV	MSE	CV	MSE
OLS	0.895	0.955	3.395	11.809	2.781	2.712
KNN	0.400	0.572	0.415	0.578	0.410	0.580
LASSO	0.379	0.591	0.407	0.578	0.408	0.578
TREE	0.339	0.477	0.415	0.577	0.418	0.477
FOREST	0.367	0.501	0.382	0.528	0.394	0.546
BOOST.TREE	0.379	0.496	0.384	0.481	0.379	0.501

Tree based methods competitive or dominant across the board

Increase p Simulation - Nonlinear Model (N = 200)

Code: IncreasePNonlinearN200_Example10.R

	$p = 1$		$p = 5$		$p = 20$	
	CV	MSE	CV	MSE	CV	MSE
OLS	0.270	0.534	0.330	0.503	0.341	0.541
KNN	0.236	0.411	0.196	0.280	0.327	0.536
LASSO	0.286	0.535	0.321	0.503	0.307	0.526
TREE	0.363	0.412	0.236	0.369	0.293	0.457
FOREST	0.252	0.401	0.189	0.323	0.246	0.413
BOOST.TREE	0.371	0.412	0.154	0.190	0.219	0.364

Tree based methods competitive or dominant across the board

Increase p Simulation - Nonlinear Model (N = 200)

Code: IncreasePNonlinearN200_Example10.R

	$p = 50$		$p = 100$		$p = 200$	
	CV	MSE	CV	MSE	CV	MSE
OLS	0.445	0.607	0.709	0.755	1.667	11.924
KNN	0.352	0.577	0.373	0.569	0.376	0.575
LASSO	0.331	0.524	0.355	0.526	0.356	0.534
TREE	0.282	0.422	0.307	0.457	0.306	0.435
FOREST	0.274	0.452	0.298	0.479	0.319	0.504
BOOST.TREE	0.242	0.384	0.244	0.388	0.255	0.396

Tree based methods competitive or dominant across the board

Boosted regression tree comes out near the top uniformly in this design

4. Interpretation of Additive Tree Models

Linear Combinations of Trees

Tree model gives nice 2-D graphic with relatively easy interpretation

Linear combinations of trees (i.e. boosted or bagged trees, random forests) lose this feature

Two common aids to interpretation of results:

- Relative importance - A measure of the relevance of each predictor variable
- Partial dependence plots - Average marginal/partial effects

Relative Importance Measures

In many data-mining applications, goal is to learn identity of important predictors

Breiman (1984) measure of relevance of predictor X_l for a single tree T:

$$\mathcal{I}_l^2(T) = \sum_{t=1}^{J-1} \hat{\imath}_t^2 1(v(t) = l)$$

- Sum over $J-1$ internal nodes of the tree
- Variable $X_{v(t)}$ used to partion node t
- Chosen variable gives maximum estimated improvement in squared error fit, $\hat{\imath}_t^2$
- Importance measure is just sum of squared improvements over all internal nodes where X_l was the splitting variable

Relative Importance Measures

$\mathcal{I}_l^2(T)$ easily generalized to additive tree expansions:

$$\mathcal{I}_l^2 = \frac{1}{M}\sum_{m=1}^{M}\mathcal{I}_l^2(T_m)$$

Comments:

- $\mathcal{I}_l^2$ and $\mathcal{I}_l^2(T)$ are "squared relevance" - customary to report square root
- Customary to normalize: 1) to sum to 100 OR 2) largest equal to 100
- For K class classification, K probability models are fit.
  - Relevance measure average over K models: $\mathcal{I}_l^2 = \frac{1}{K}\sum_{k=1}^{K}\mathcal{I}_{lk}^2$ where $\mathcal{I}_{lk}^2 = \frac{1}{M}\sum_{m=1}^{M}\mathcal{I}_l^2(T_{km})$

Example: 401(k) Eligibility

Data: 1991 SIPP, $n = 9915$

Leave 1915 observations out for validation exercise - Use 8000 for training

Goal: Predict 401(k) eligibility

Predictors:

- income
- age
- schooling
- family size
- married
- male
- two-earner household

E.g. estimate propensity score for understanding effect of 401(k)'s on wealth as in Poterba, Venti, and Wise (1994, 1995, 1996, 2001); Abadie (2003); Benjamin (2003); Chernozhukov and Hansen (2004); Belloni, Chernozhukov, Fernández-Val, and Hansen (2014)

401(k) Eligibility

Code: Tree401k_Example11.R

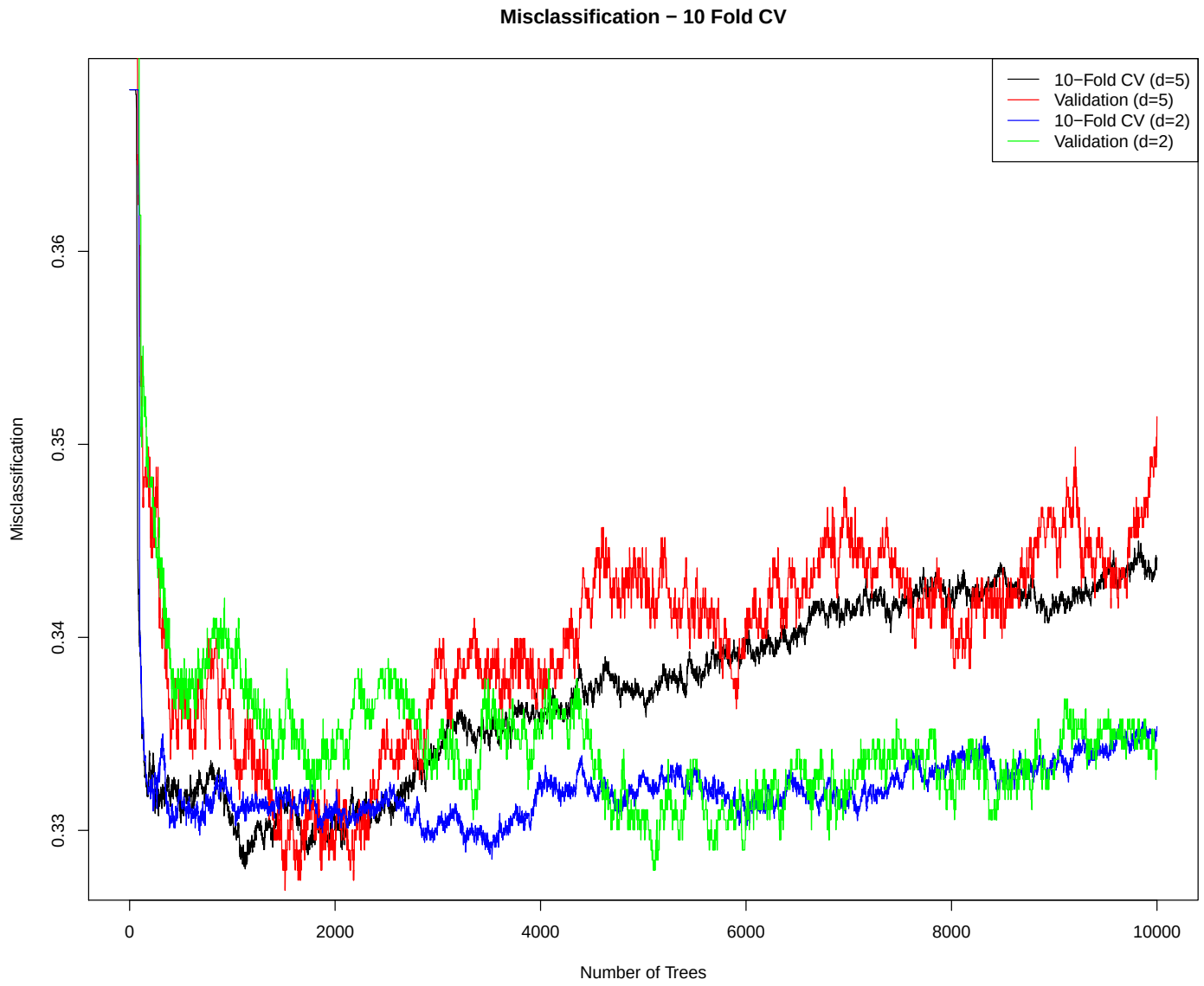
Let's look at boosted trees for 401(k) eligibility model

Set $d = 2$ or $d = 5$, $\lambda = .01$

Consider up to 10000 trees - choose final number from 10-fold CV based on missclassification rate

Also look at performance in validation data for each number of trees

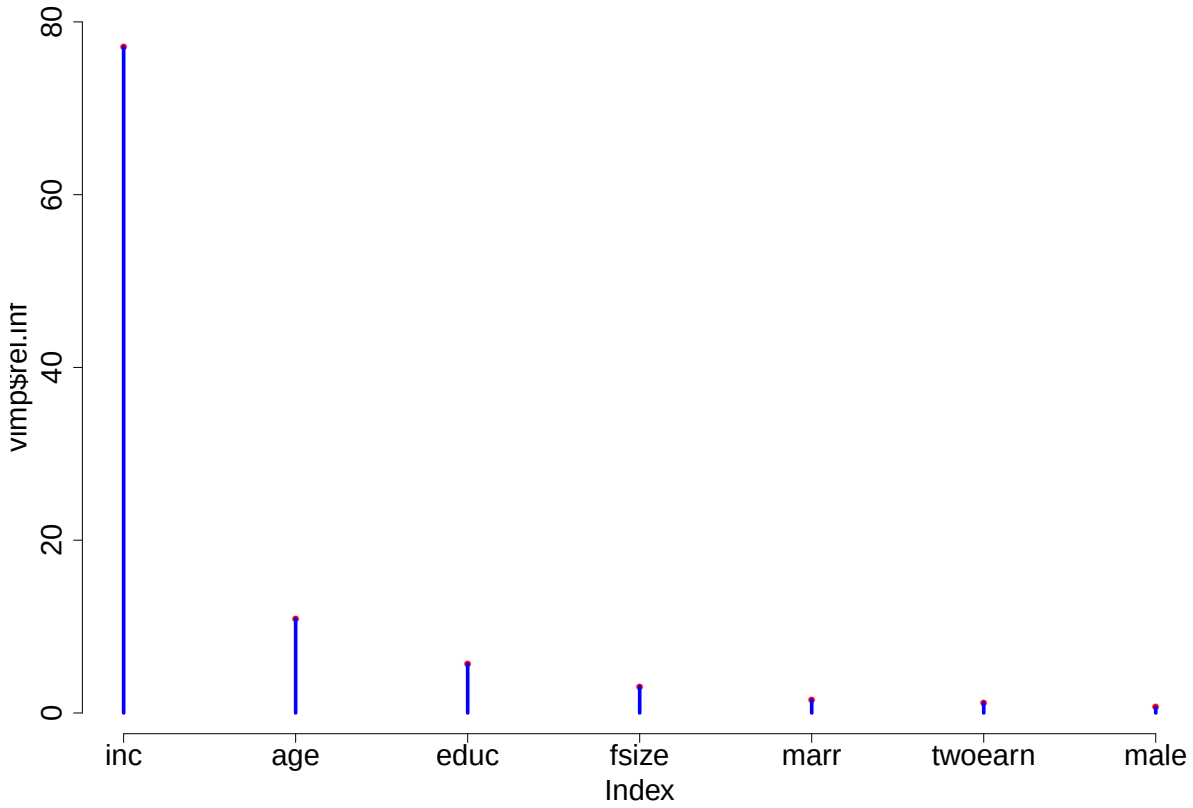
CV and Validation data performance:



401(k) Eligibility: Relative Importance

Code: Tree401k_Example12.R

Look at relative importance from boosted tree with $d = 5$, $\lambda = .01$, and 10-Fold CV M (but use all data)



Partial Dependence

Other common interpretation device is to plot estimated function

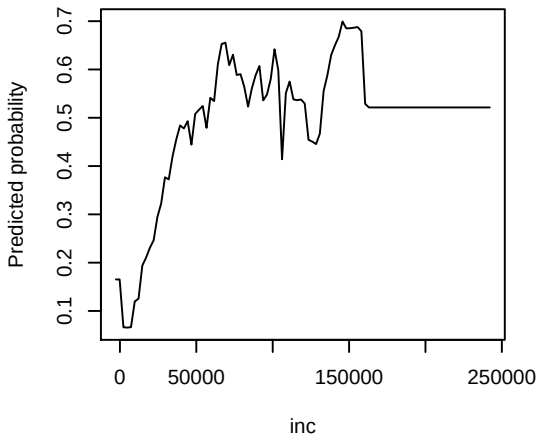
Obviously hard when dimension of X is bigger than two

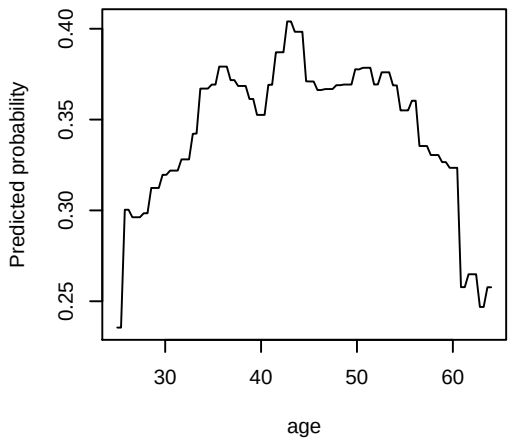
Instead, report "partial dependence" of function on different X variables

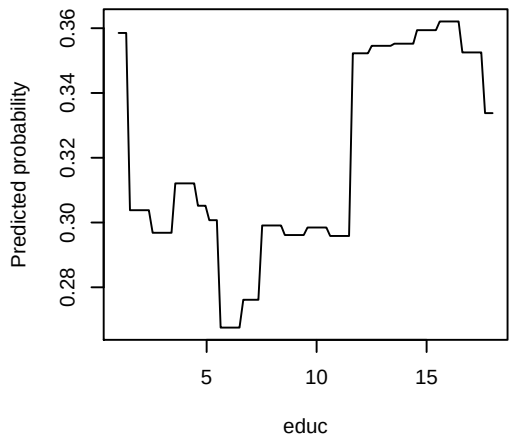
- Let X_S be a subvector of the X variables of interest ($S \subset \{1, ..., p\}$)
- Let X_C be the complement of X_S ($S \cup C = \{1, ..., p\}$)
- Function of interest $f(X) = f(X_S, X_C)$
- Partial dependence is just $\mathrm{E}_{X_C}[f(X_S, X_C)]$ - average out dimensions not interested in
- Natural estimator: $\bar{f}_S(X_S) = \frac{1}{n} \sum_{i=1}^{n} \widehat{f}(X_S, x_{ic})$

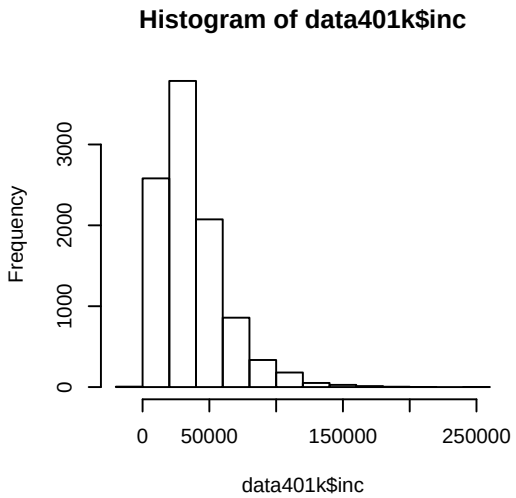
401(k) Eligibility: Partial Dependence Plots

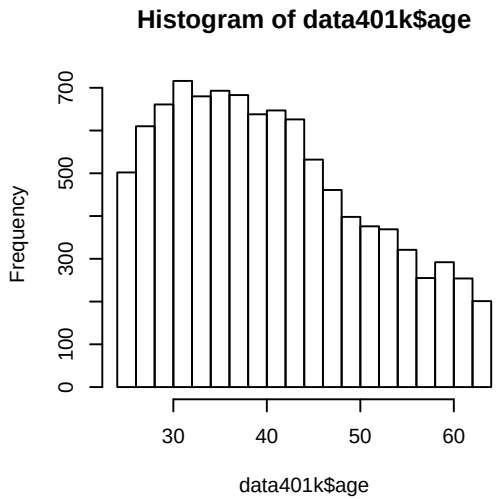
Code: Tree401k_Example12.R

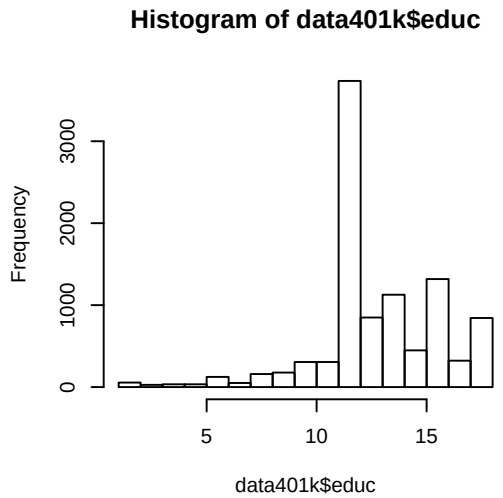




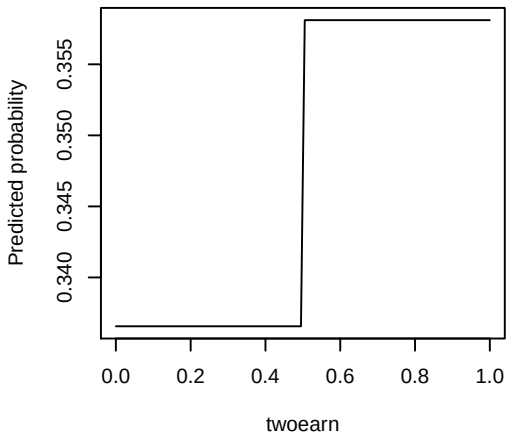


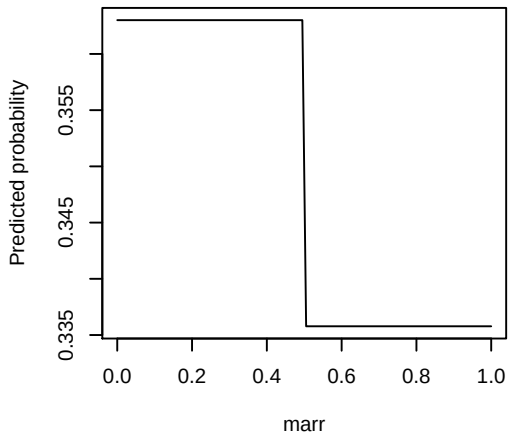


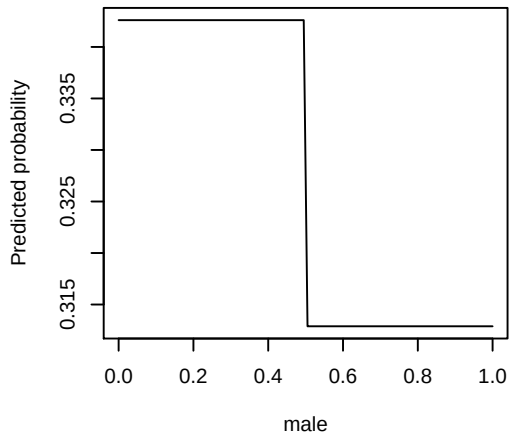


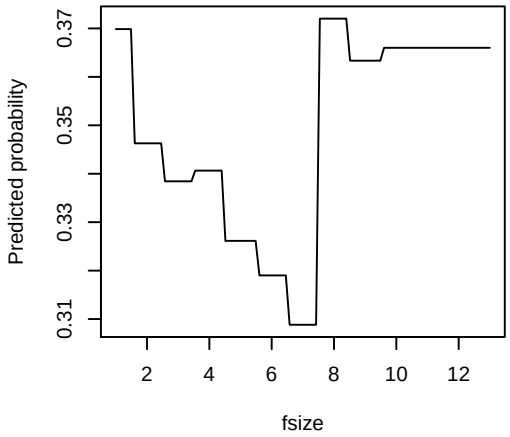


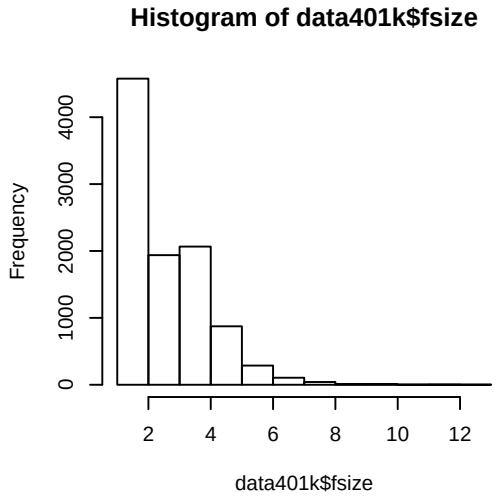
401(k) Eligibility: Partial Dependence Plots











5. Example: Spam Filter

Spam Filter Example

Code: Spam_Example13.R

Data: 4601 observations (digitized email) [Split into 3000 training and 1601 validation]

Variables:

- Outcome: 0-1, 1 = spam
- Predictors: 57 attributes
  - 48 word frequencies - [0,100], (# times WORD appears in email)/(total number of words)
  - 6 character frequencies - [0,100], (# times CHAR appears in email)/(total number of characters)
  - 3 measures of length of sequences of capital letters

Goal: Classify email as spam/not-spam. [Note: In practice, blocking not-spam on accident is more costly than accidentally letting spam through. In principle want to tune filter with asymmetric costs in mind.]

Performance on Validation Data

Missclassification Rates:

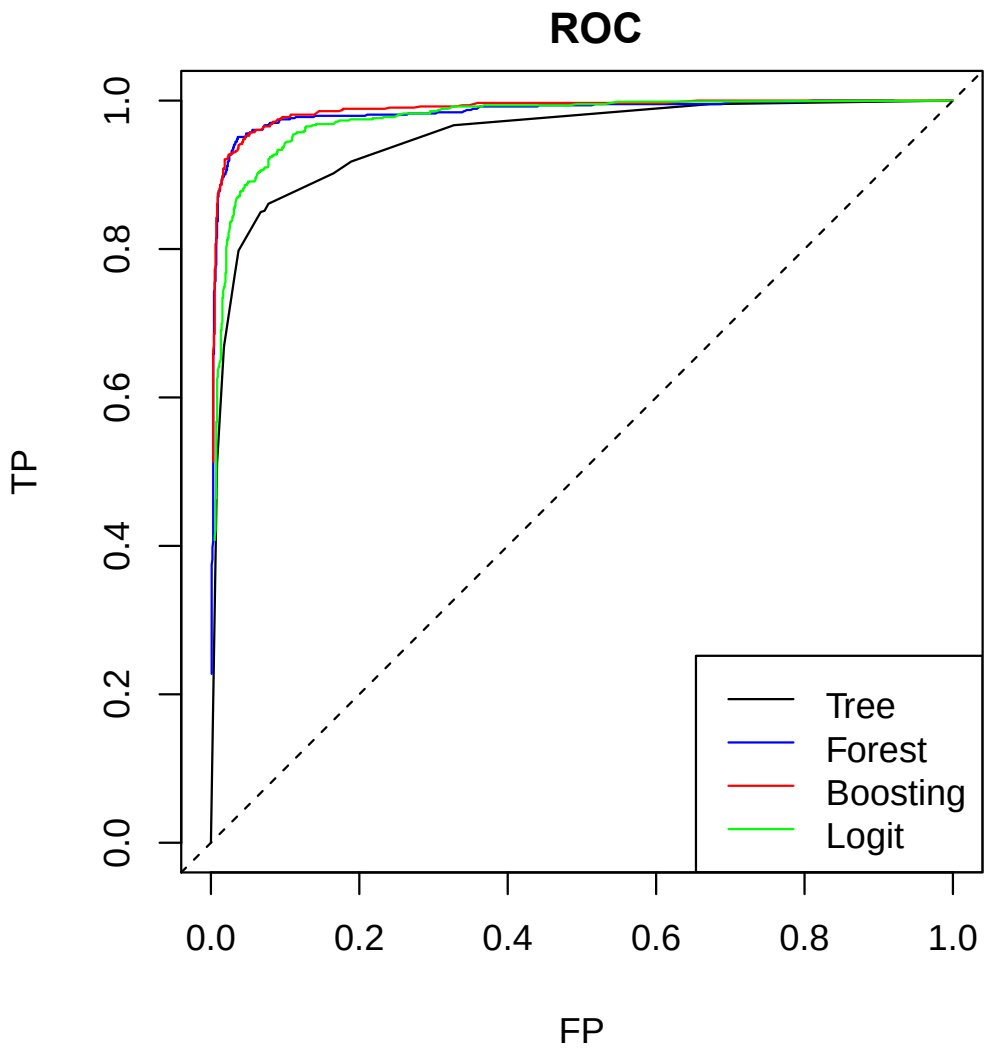
- Baseline (sample mean - all not spam): .395
- Logit with all 57 predictors: .074
- Tree with α chosen by 10-fold CV: .100
- Random Forest with 5000 trees, $m = \sqrt{p}$, maximum depth default in R (as large as possible): .052
- Boosted Forest with # trees and depth (2 or 5) chosen by 10-fold CV (missclassification), $\lambda = .01$: .043

Boosted trees again the best followed by random forests

Computation time inversely related to performance (boosted trees take by far the longest)

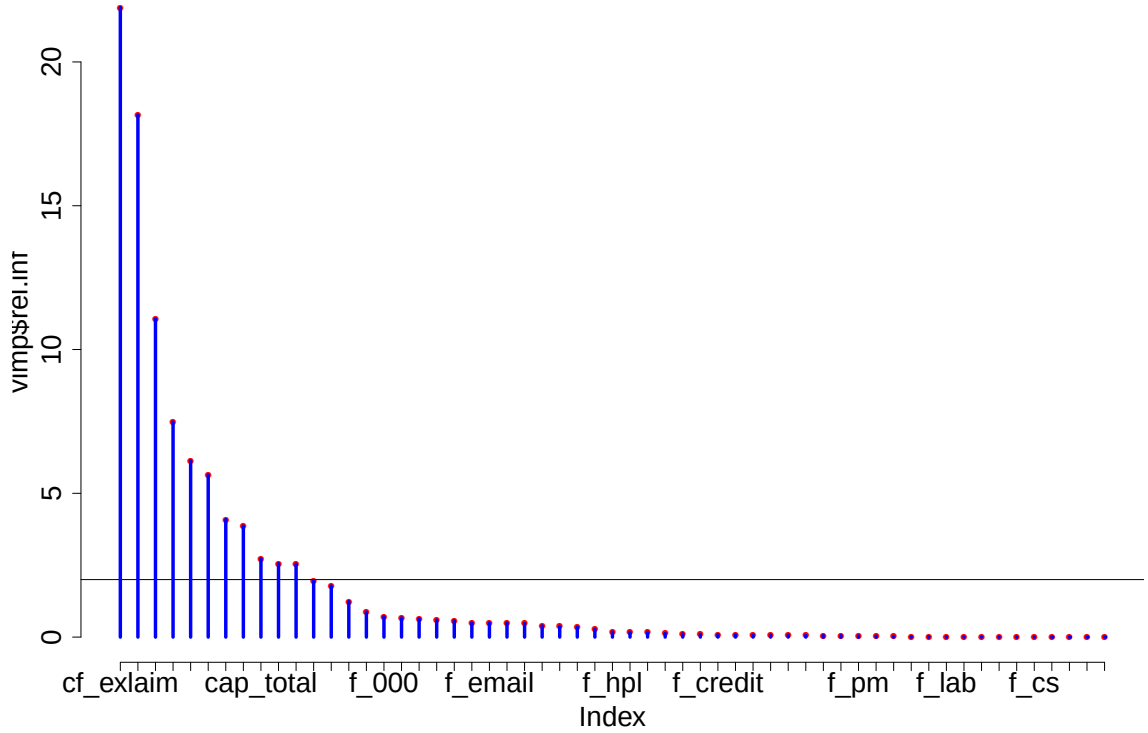
Spam ROC

ROC (Receiver Operating Characteristics) is interesting in this example:



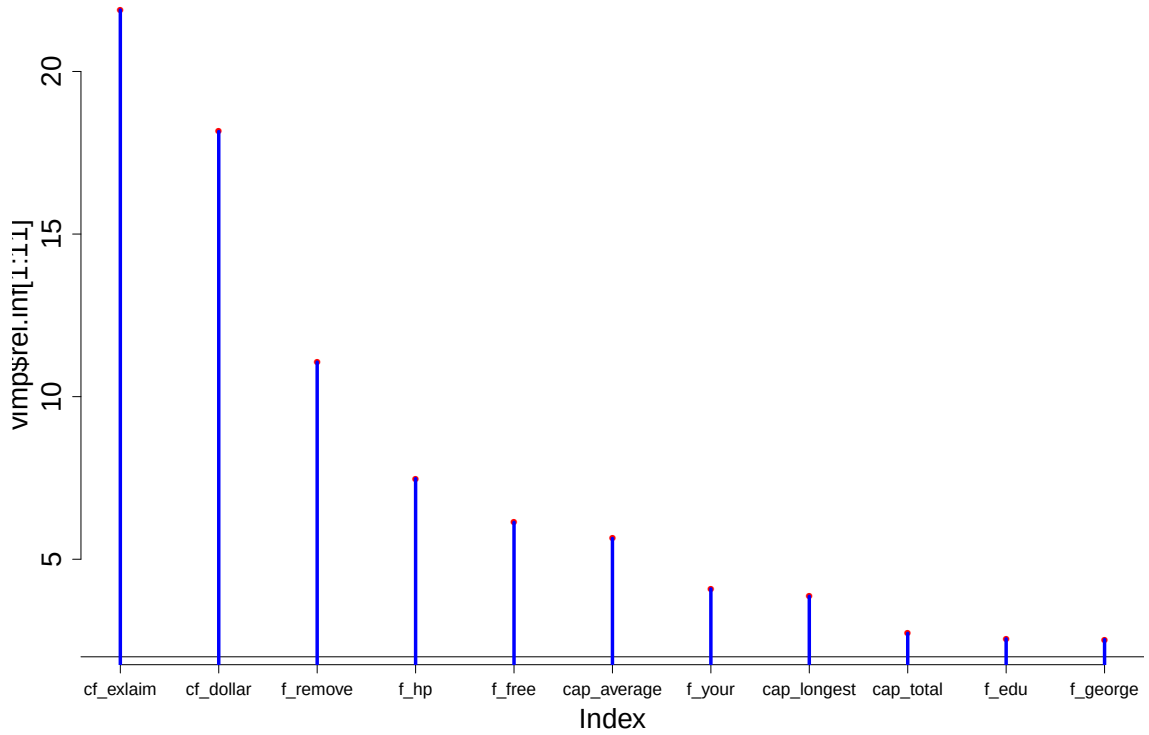
Variable Importance using Boosted Model

Variable importance for all variables:



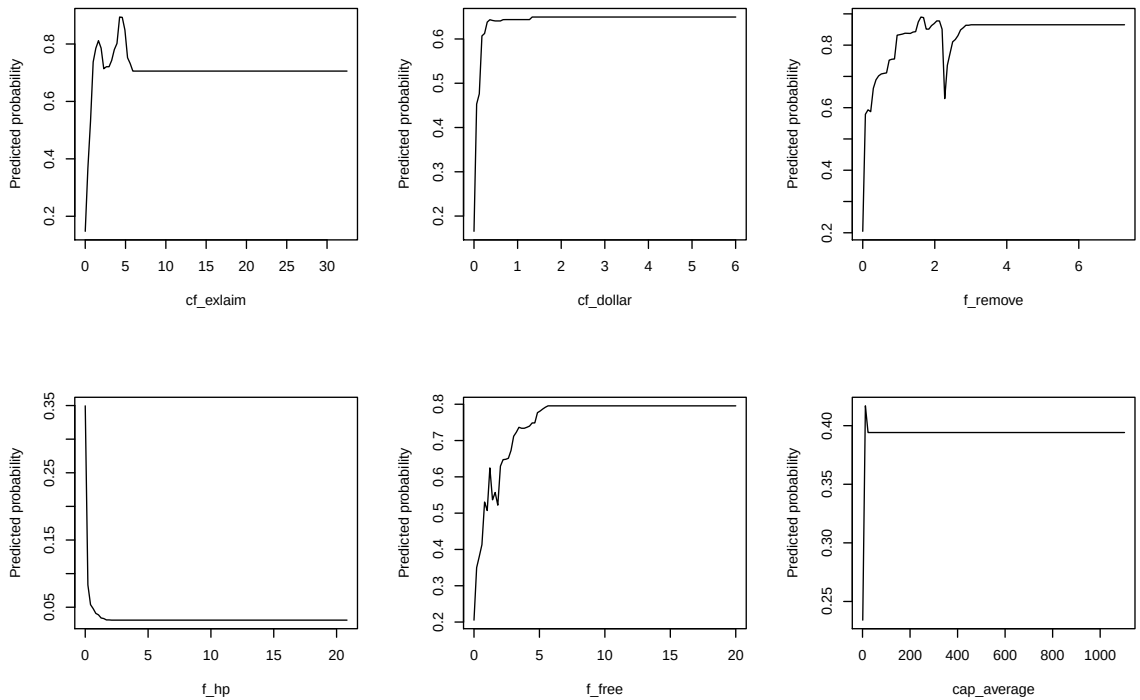
Variable Importance using Boosted Model

Looking at all 57 is a little hard to read - Just pull out top variables



Partial Dependence using Boosted Model

Just look at top 6 variables:



6. Comments on Tree-Based Methods

Parting Thoughts

Some remarks on tree-based methods:

- Tree-based methods provide "default" in many modern big-data/data-mining settings
- Tree-based methods attractive for a variety of reasons
  - Almost automatic - require relatively little choice on the part of a user
    - do variable selection
    - detect and model nonlinearities - including interactions - without requiring prespecification
  - Relatively fast computationally and scale readily to very large data sets - easily parallelized/distributed
  - Many extensions, related methods (e.g. can put more elaborate models at each leaf)
- Have some unappealing feature
  - May be outperformed in forecasting by methods that are more readily able to build in sensible structure
  - Become hard to interpret - especially when ensembles are formed
  - Theory relatively poorly developed relative to penalized estimation methods
    - Hard to verify tuning parameter choices do not lead to overfitting