Boosting

Ryan Tibshirani
Data Mining: 36-462/36-662

April 25 2013

Optional reading: ISL 8.2, ESL 10.1–10.4, 10.7, 10.13

Reminder: classification trees

Suppose that we are given training data (x_i, y_i), $i = 1, \ldots n$, with $y_i \in \{1, \ldots K\}$ the class label and $x_i \in \mathbb{R}^p$ the associated features

Recall that the CART algorithm to fit a classification tree starts by considering splitting on variable j and split point s, and defines the regions

$$R_1 = \{X \in \mathbb{R}^p : X_j \leq s\}, \quad R_2 = \{X \in \mathbb{R}^p : X_j > s\}$$

Within each region the proportion of points of class k is

$$\hat{p}_k(R_m) = \frac{1}{n_m} \sum_{x_i \in R_m} 1\{y_i = k\}$$

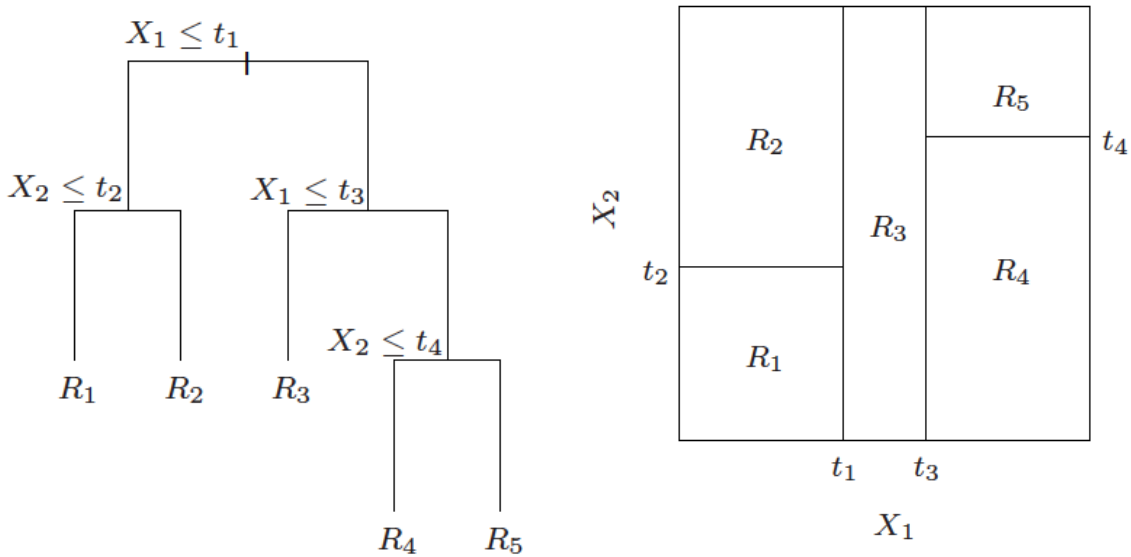
with n_m the number of points in R_m. The most common class is

$$c_m = \underset{k=1,\ldots K}{\text{argmax}} \ \hat{p}_k(R_m)$$

The feature and split point j, s are chosen greedily by minimizing the misclassification error

$$\underset{j,s}{\operatorname{argmin}} \left(\left[1 - \hat{p}_{c_1}(R_1) \right] + \left[1 - \hat{p}_{c_2}(R_2) \right] \right)$$

This strategy is repeated within each of the newly formed regions



Classification trees with observation weights

Now suppose that we are are additionally given observation weights w_i, $i = 1, \ldots n$. Each $w_i \geq 0$, and a higher value means that we place a higher importance in correctly classifying this observation

The CART algorithm can be easily adapted to use the weights. All that changes is that our sums become weighted sums. I.e., we now use the weighted proportion of points of class k in region R_m:

$$\hat{p}_k(R_m) = \frac{\sum_{x_i \in R_m} w_i 1\{y_i = k\}}{\sum_{x_i \in R_m} w_i}$$

As before, we let

$$c_m = \underset{k=1,\ldots K}{\operatorname{argmax}} \ \hat{p}_k(R_m)$$

and hence $1 - \hat{p}_{c_m}(R_m)$ is the weighted misclassification error

Boosting

Boosting[1] is similar to bagging in that we combine the results of several classification trees. However, boosting does something fundamentally different, and can work a lot better

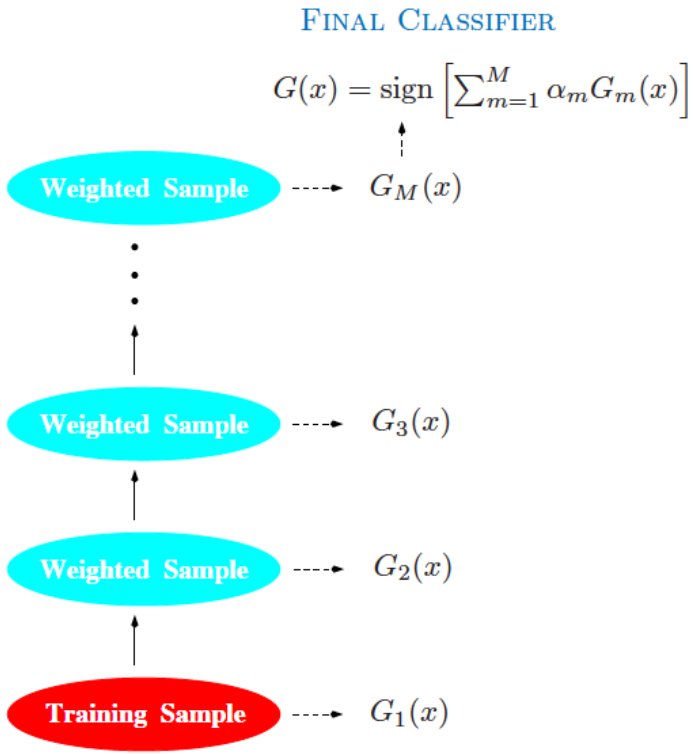
As usual, we start with training data (x_i, y_i), $i = 1, \ldots n$. We'll assume that $y_i \in \{-1, 1\}$ and $x_i \in \mathbb{R}^p$. Hence we suppose that a classification tree (fit, e.g., to the training data) will return a prediction of the form $\hat{f}^{\text{tree}}(x) \in \{-1, 1\}$ for an input $x \in \mathbb{R}^p$

In boosting we combine a weighted sum of B different tree classifiers,

$$\hat{f}^{\text{boost}}(x) = \text{sign}\left(\sum_{b=1}^{B} \alpha_b \, \hat{f}^{\text{tree},b}(x) \right)$$

[1] Freund and Schapire (1995), "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". Similar ideas were around earlier

One of the key differences between boosting and bagging is how the individual classifiers $\hat{f}^{\text{tree},b}$ are fit. Unlike in bagging, in boosting we fit the tree to the entire training set, but adaptively weight the observations to encourage better predictions for points that were previously misclassified



(From ESL page 338)

The basic boosting algorithm (AdaBoost)

Given training data (x_i, y_i), $i = 1, \ldots n$, the basic boosting method is called AdaBoost, and can be described as:

- Initialize the weights by $w_i = 1/n$ for each i
- For $b = 1, \ldots B$:
  1. Fit a classification tree $\hat{f}^{\text{tree},b}$ to the training data with weights $w_1, \ldots w_n$
  2. Compute the weighted misclassification error

$$e_b = \frac{\sum_{i=1}^n w_i 1\{y_i \neq \hat{f}^{\text{tree},b}(x_i)\}}{\sum_{i=1}^n w_i}$$

  3. Let $\alpha_b = \log\{(1 - e_b)/e_b\}$
  4. Update the weights as

$$w_i \leftarrow w_i \cdot \exp(\alpha_b 1\{y_i \neq \hat{f}^{\text{tree},b}(x_i)\})$$

     for each i
- Return $\hat{f}^{\text{boost}}(x) = \text{sign}\Big(\sum_{b=1}^B \alpha_b \hat{f}^{\text{tree},b}(x)\Big)$

Example: boosting stumps

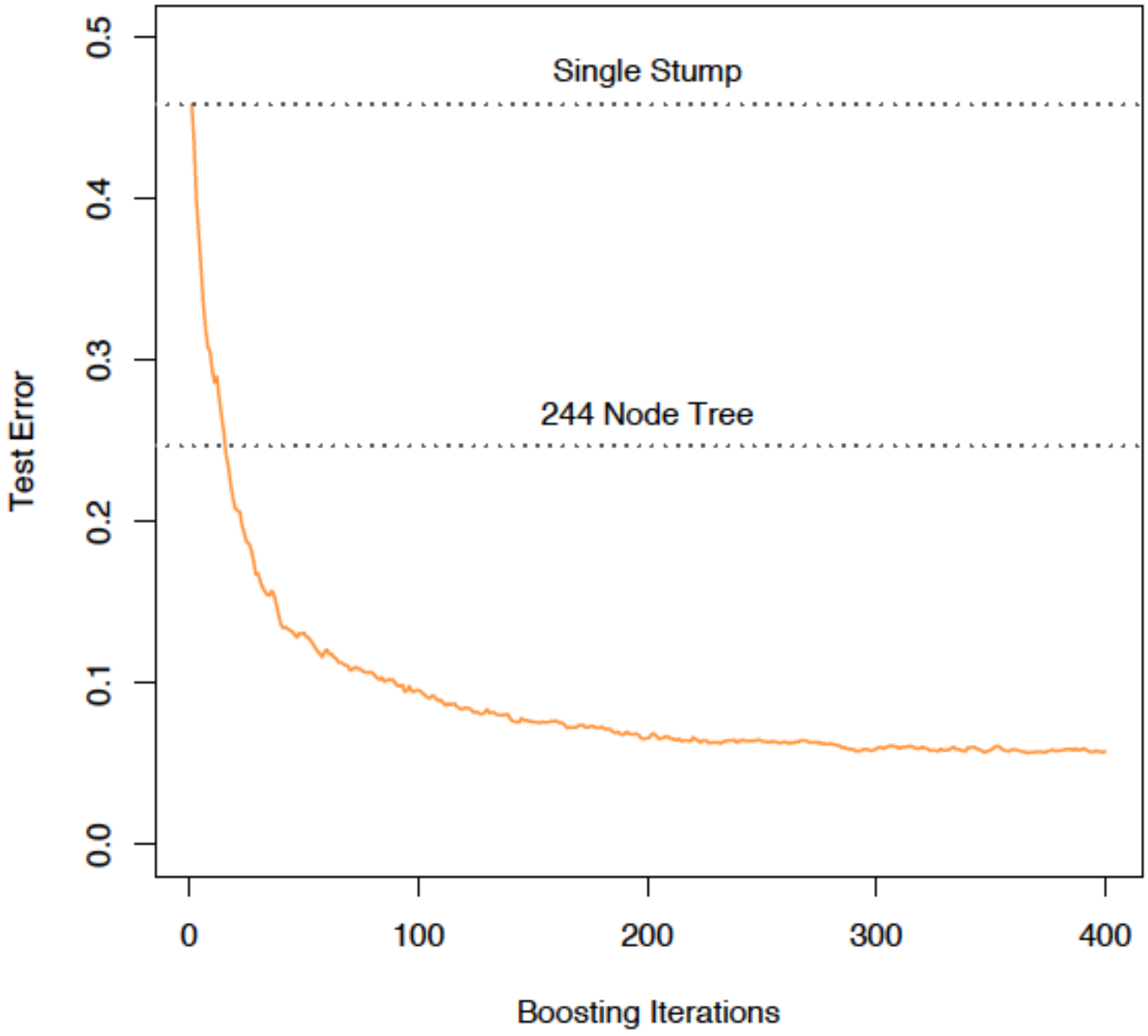
Example (from ESL page 339): here $n = 1000$ points were drawn from the model

$$Y_i = \begin{cases} 1 & \text{if } \sum_{j=1}^{10} X_{ij}^2 > \chi_{10}^2(0.5) \\ -1 & \text{otherwise} \end{cases}, \quad \text{for } i = 1, \ldots n$$

where each $X_{ij} \sim N(0, 1)$ independently, $j = 1, \ldots 10$

A stump classifier was computed: this is just a classification tree with one split (two leaves). This has a bad misclassification rate on independent test data, of about 45.8%. (Why is this not surprising?)

On the other hand, boosting stumps achieves an excellent error rate of about 5.8% after $B = 400$ iterations. This is much better than, e.g., a single large tree (error rate 24.7%)


Single Stump
244 Node Tree
Test Error
0.0
0.1
0.2
0.3
0.4
0.5
0
100
200
300
400
Boosting Iterations

Why does boosting work?

The intution beyond boosting is simple: we weight misclassified observations in such a way that they get properly classified in future iterations. But there are many ways to do this—why does the particular weighting seem to work so well?

One nice discovery was the connection between boosting and forward stepwise modeling.[2] To understand this connection, it helps to recall forward stepwise linear regression. Given a continuous response $y \in \mathbb{R}^n$ and predictors $X_1, \ldots X_p \in \mathbb{R}^n$, we:

- Choose the predictor X_j giving the smallest squared error loss $\sum_{i=1}^n (y_i - \hat{\beta}_j X_{ij})^2$ ($\hat{\beta}_j$ is obtained by regressing y on X_j)
- Choose the predictor X_k giving the smallest additional loss $\sum_{i=1}^n (r_i - \hat{\beta}_k X_{ik})^2$ ($\hat{\beta}_k$ is obtained by regressing r on X_k), where r is the residual $r = y - \hat{\beta}_j X_j$
- Repeat the last step

[2] Friedman et al. (2000), "Additive Logistic Regression: A Statistical View of Boosting"

Boosting fits an additive model

In the classification setting, y_i is not continuous but discrete, taking values in $\{-1, 1\}$. Therefore squared error loss is not appropriate. Instead we use exponential loss:

$$L(y_i, f(x_i)) = \exp(-y_i f(x_i))$$

Furthermore, boosting doesn't model $f(x_i)$ as a linear function of the variables $x_{i1}, \ldots x_{ip}$, but rather as a linear function of trees $T_1(x_i), \ldots T_M(x_i)$ of a certain size. (Note that the number M of trees of a given size is finite.) Hence the analogous forward stepwise modeling is as follows:

- Choose the tree T_j and coefficient β_j giving the smallest exponential loss $\sum_{i=1}^{n} \exp(-y_i \beta_j T_j(x_i))$
- Choose the tree T_k and coefficient β_k giving the smallest additional loss $\sum_{i=1}^{n} \exp(-y_i\{\beta_j T_j(x_i) + \beta_k T_k(x_i)\}) = \sum_{i=1}^{n} \exp(-y_i \beta_j T_j(x_i)) \cdot \exp(-y_i \beta_k T_k(x_i))$
- Repeat the last step

This forward stepwise procedure can be tied concretely to the AdaBoost algorithm that we just learned (see ESL 10.4 for details)

This connection brought boosting from an unfamiliar algorithm to a more-or-less familiar statistical concept. Consequently, there were many suggested ways to extend boosting:

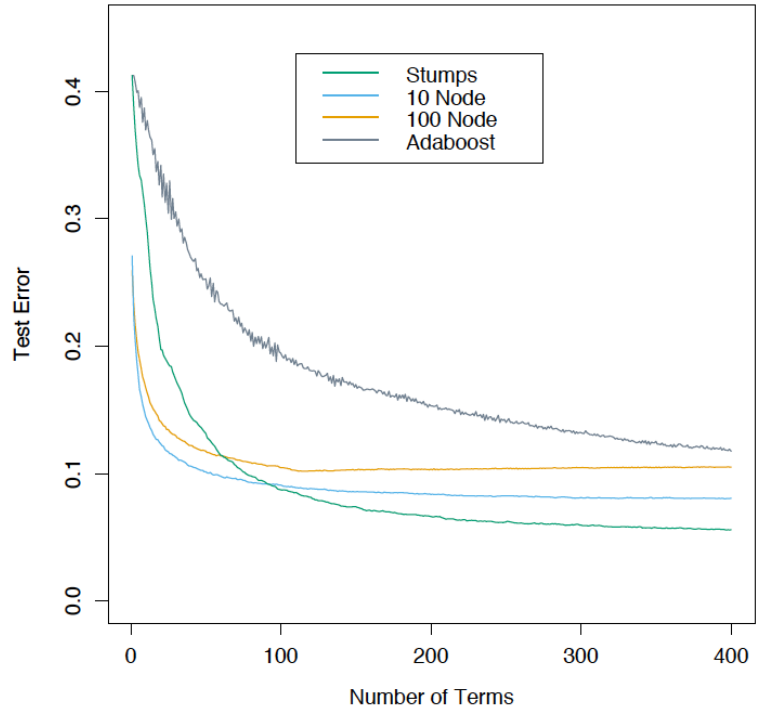
- Other losses: exponential loss is not really used anywhere else. It leads to a computationally efficient boosting algorithm (AdaBoost), but binomial deviance and SVM loss are two alternatives that can work better. (Algorithms for boosting are now called gradient boosting)
- Shrinkage: instead of adding one tree at a time (thinking in the forward stepwise perspective), why don't we add a small multiple of a tree? This is called shrinkage a form of regularization for the model building process. This can also be very helpful in terms of prediction accuracy

Choosing the size of the trees

How big should the trees be used in the boosting procedure?

Remember that in bagging we grew large trees, and then either stopped (no pruning) or pruned back using the original training data as a validation set. In boosting, actually, the best methods if to grow small trees with no pruning

In ESL 10.11, it is argued that right size actually depends on the level of the iteractions between the predictor variables. Generally trees with 2–8 leaves work well; rarely more than 10 leaves are needed



Disadvantages

As with bagging, a major downside is that we lose the simple interpretability of classification trees. The final classifier is a weighted sum of trees, which cannot necessarily be represented by a single tree.

Computation is also somewhat more difficult. However, with AdaBoost, the computation is straightforward (moreso than gradient boosting). Further, because we are growing small trees, each step can be done relatively quickly (compared to say, bagging)

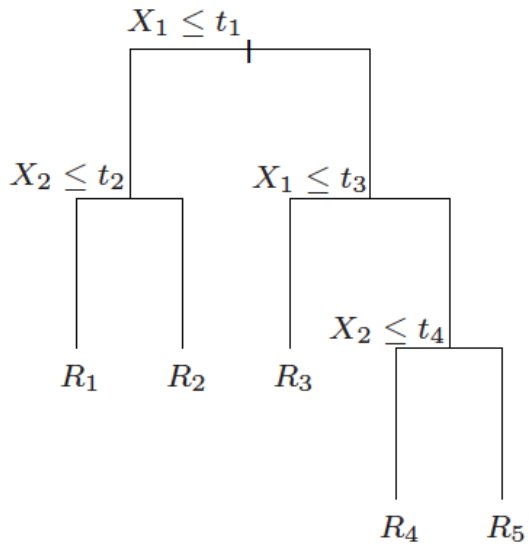
To deal with the first downside, a measure of variable importance has been developed for boosting (and it can be applied to bagging, also)

Variable importance for trees

For a single decision tree $\hat{f}^{\text{tree}}$, we define the squared importance for variable j as

$$\text{Imp}_j^2(\hat{f}^{\text{tree}}) = \sum_{k=1}^{m} \hat{d}_k \cdot 1\{\text{split at node } k \text{ is on variable } j\}$$

where m is the number of internal nodes (non-leaves), and $\hat{d}_k$ is the improvement in training misclassification error from making the kth split



Variable importance for boosting

For boosting, we define the squared importance for a variable j by simplying averaging the squared importance over all of the fitted trees:

$$\text{Imp}_j^2(\hat{f}^{\text{boost}}) = \frac{1}{B}\sum_{b=1}^{B}\text{Imp}_j^2(\hat{f}^{\text{tree},b})$$

(To get the importance, we just take the squared root.) We also usually set the largest importance to 100, and scale all of the other variable importances accordingly, which are then called relative importances

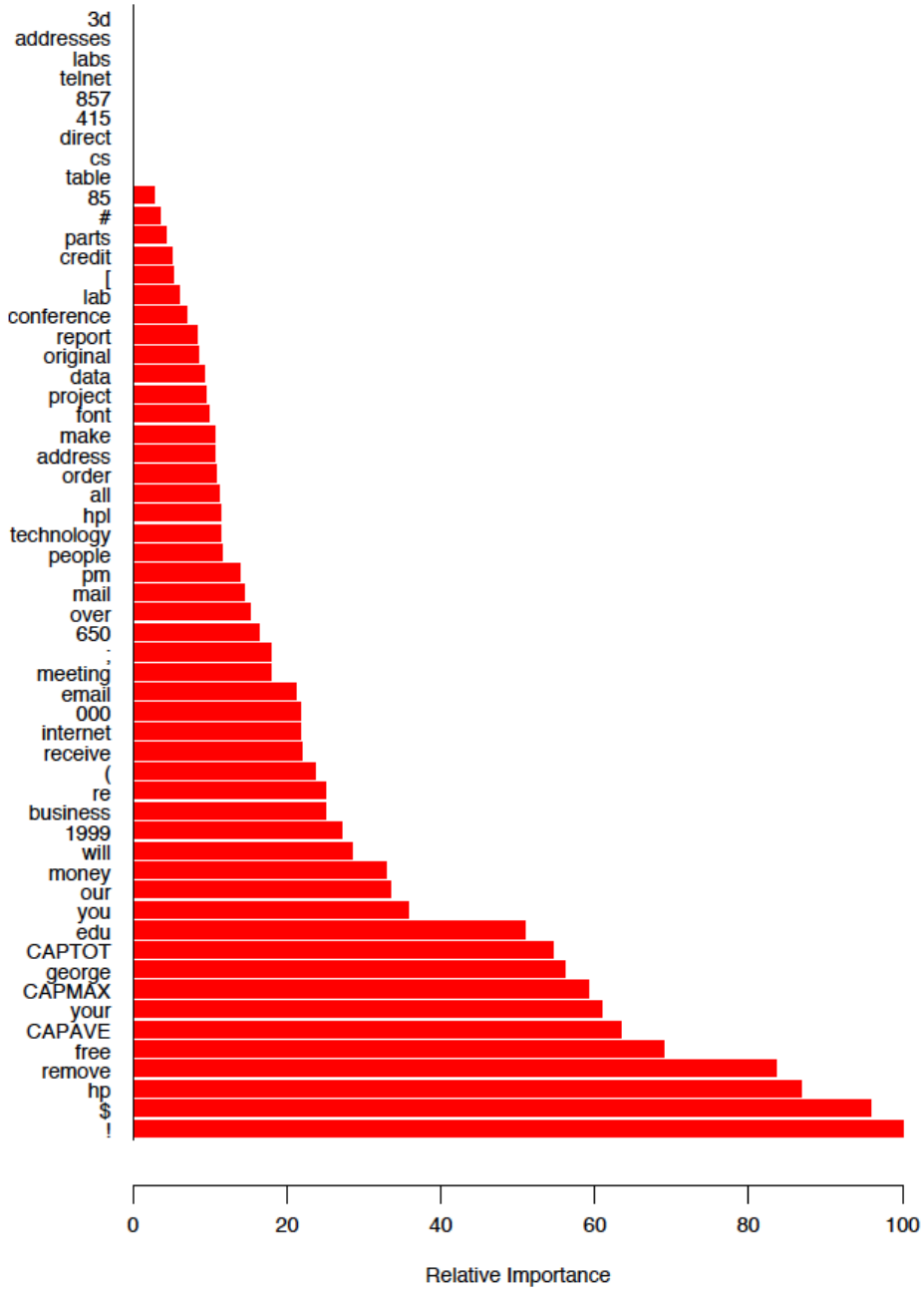
This averaging stabilizes the variable importances, which means that they tend to be much more accurate for boosting than they do for any single tree

Example: spam data

Example from ESL 10.8: recall the spam data set, with $n = 4601$ emails (1813 spam emails), and given $p = 58$ attributes on each email

A classification tree, grown by CART and pruned to 15 leaves, had a classification error of 8.7% on an independent test set. Gradient boosting with stumps (only two leaves per tree) achieved an error rate of 4.7% on an independent test set. This is nearly an improvement by a factor of 2! (It also outperforms additive logistic regression and MARS)

So what does the boosting classifier look like? This question is hard to answer, because it's not a tree, but we can look at the relative variable importances


3d
addresses
labs
telnet
857
415
direct
cs
table
85
#
parts
credit
[
lab
conference
report
original
data
project
font
make
address
order
all
hpl
technology
people
pm
mail
over
650
;
meeting
email
000
internet
receive
(
re
business
1999
will
money
our
you
edu
CAPTOT
george
CAPMAX
your
CAPAVE
free
remove
hp
$
!
0
20
40
60
80
100
Relative Importance

Bagging and boosting in R

The package `adabag` implements both bagging and boosting (AdaBoost) for trees, via the functions `bagging` and `boosting`

The package `ipred` also performs bagging with the `bagging` function.

The package `gbm` implements both AdaBoost and gradient boosting; see the function `gbm` and its `distribution` option

The package `mboost` is an alternative for boosting, that is quite flexible; see the function `blackboost` for boosting trees, and the function `mboost` for boosting arbitrary base learners

Recap: boosting

In this lecture we learned boosting in the context of classification via decision trees. The basic concept is to repeatedly fit classification trees to weighted versions of the training data. In each iteration we update the weights in order to better classify previously misclassified observations. The final classifier is a weighted sum of the decisions made by the trees

Boosting shares a close connection to forward stepwise model building. In a precise way, it can be viewed as forward stepwise model building where the base learners are trees, and the loss is exponential. Using other losses can improve performance, as can employing shrinkage. Boosting is a very general, powerful tool

Although the final classifier is not as simple as a tree itself, relative variable importances can be computed by looking at how many times a given variable contributes to a split, across all trees

Next time: final projects!

The final project has you perform crime mining (data mining on a crime data set)