Outline

Supervised Learning: Ensemble Methods

- Bagging
- Random Forests
- Boosting

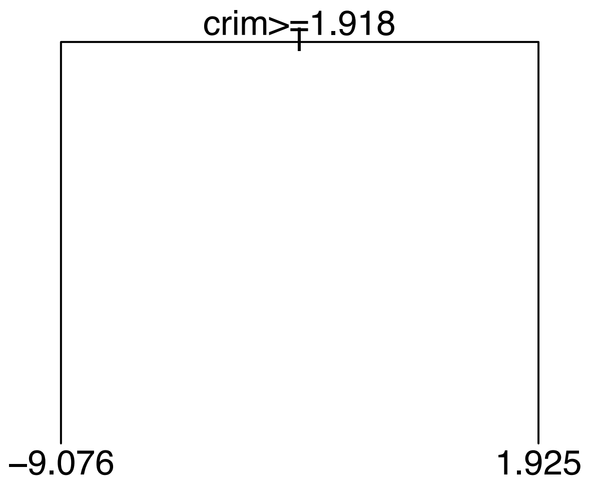
Boosting

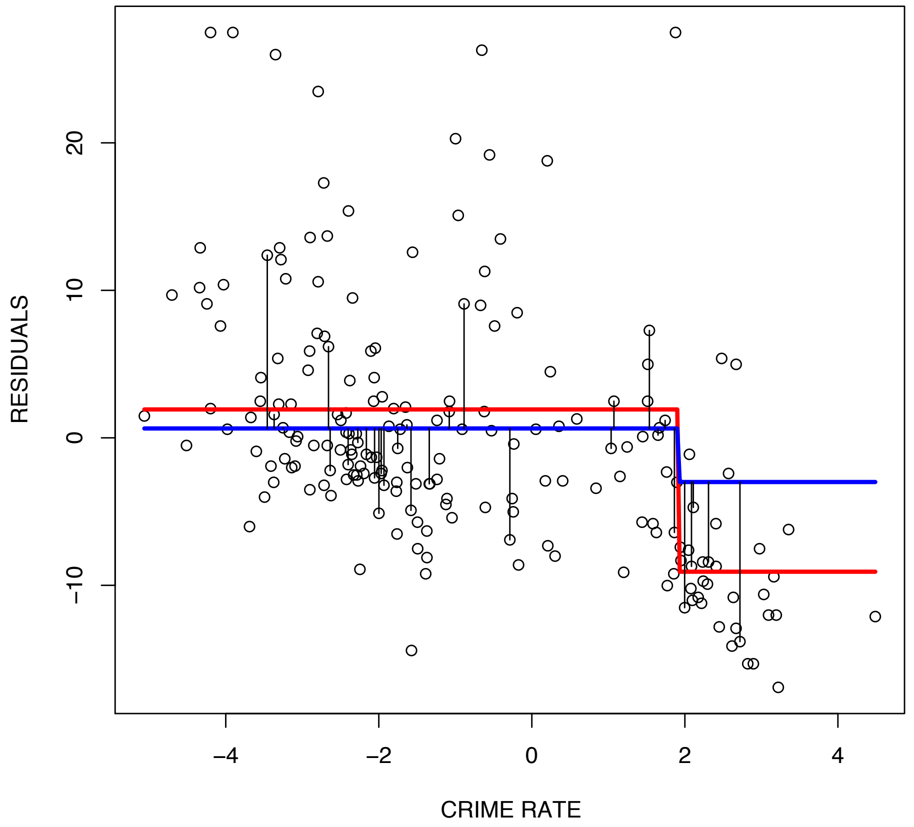
Boosting is a very different method to generate multiple predictions (function estimates) and combine them linearly. As with bagging, we have a base procedure yielding function estimates $\hat{g}(\cdot)$ (e.g. a tree algorithm).

The so-called L_2Boosting method (for regression) works as follows.

1. Fit a first function estimate from the data $\{(X_i, Y_i);\ i = 1, \ldots, n\}$ yielding a first function estimate $\hat{g}_1(\cdot)$.
   Compute residuals
   $$U_i = Y_i - \nu \hat{g}_1(X_i)\ (i = 1, \ldots, n).$$
   Denote by $\hat{f}_1(\cdot) = \nu \hat{g}_1(\cdot)$ (with shrinkage $0 < \nu \leq 1$).
2. For $m = 2, 3, \ldots, M$ do:
   Fit the residuals $(X_i, U_i) \to \hat{g}_m(\cdot)$ and set
   $$\hat{f}_m(\cdot) = \hat{f}_{m-1}(\cdot) + \nu \hat{g}_m(\cdot).$$
   Compute the current residuals $U_i = Y_i - \hat{f}_m(X_i)$ for $i = 1, \ldots, n$.

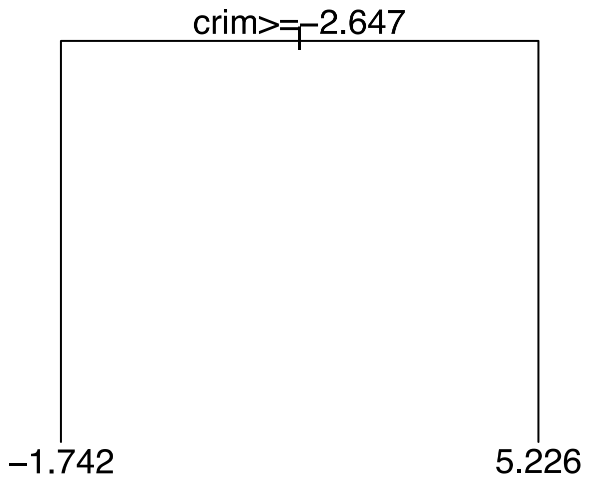
Example again Boston Housing data with single predictor variable crime rate.
First iteration: fit original observation with a stump.

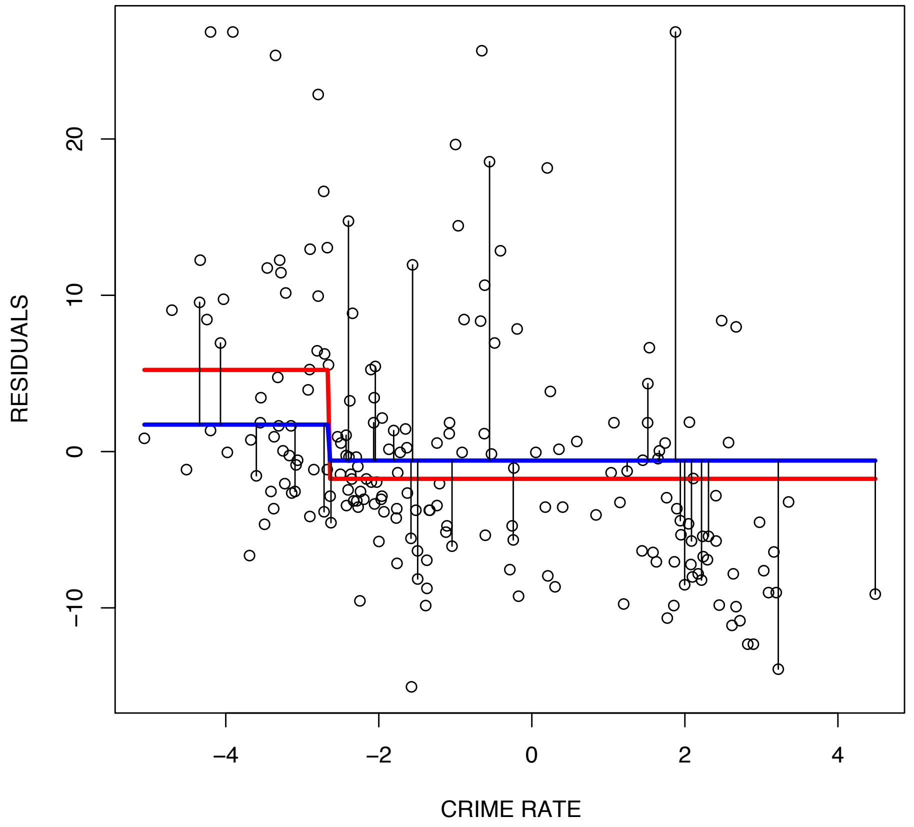




Fit of tree $\hat{g}_1(x)$ in red.
Shrunken fit $\nu\hat{g}_1(x)$ in blue.
Some residuals $U_i = Y_i - \nu\hat{g}_1(X_i)$ plotted with vertical bars. Fit these residuals in the next step.

second iteration: fit residuals $U_i = Y_i - \nu\hat{g}_1(X_i)$ from first iteration with a stump (after setting mean of Y to 0).



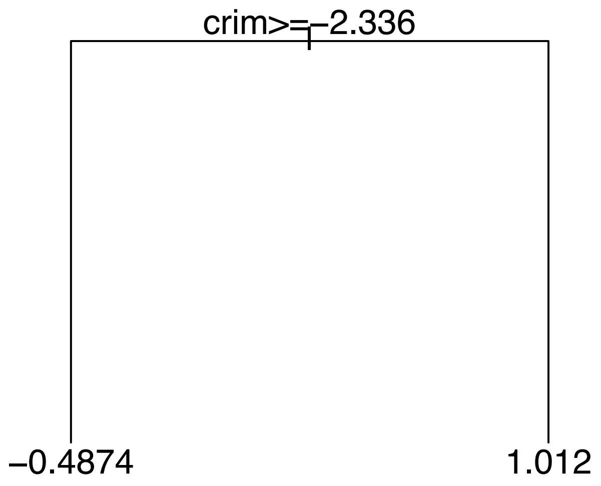


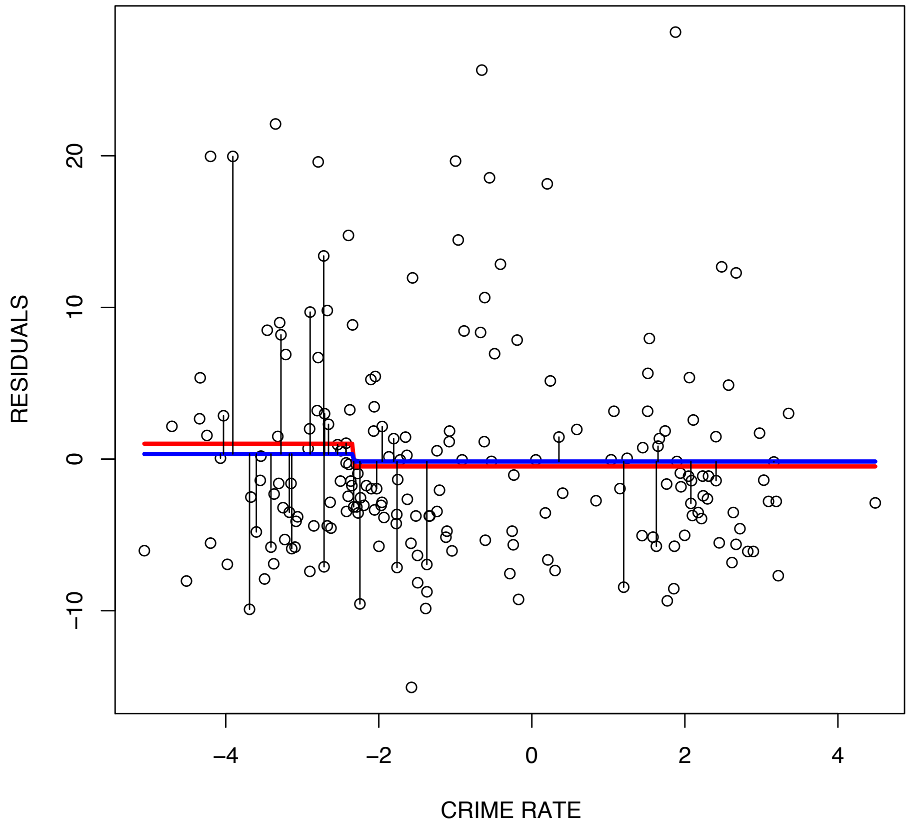
Fit of tree $\hat{g}_2(x)$ in red. Shrunken fit $\nu\hat{g}_2(x)$ in blue.
Some of the new residuals

$$U_i - \nu\hat{g}_2(X_i) = Y_i - \nu\hat{g}_1(X_i) - \nu\hat{g}_2(X_i)$$

plotted with vertical bars. Fit these residuals in the next step.

after 10 iterations:





Fit of tree $\hat{g_1 0}(x)$ in red. Shrunken fit $\nu\hat{g}_1 0(x)$ in blue.
Note that there is not a lot of signal left in the data to be fitted by $\hat{Y}(x)$. The changes in the fit are very small after many iterations.

Some notes on Boosting:

- The shrinkage parameter ν can and should be chosen to be small, e.g. $\nu = 0.1$.
- The stopping parameter M is a tuning parameter of boosting. For ν small we typically can choose M large.

Boosting is a bias reduction technique, in contrast to bagging. Boosting typically improves the performance of a single (simple) tree model.

- We often cannot construct trees which are sufficiently large due to thinning out of observations in the terminal nodes.
- Boosting is then a device to come up with a more complex solution by taking linear combination of trees.
- In presence of high-dimensional predictors, boosting is also very useful as a regularization technique for additive or interaction modeling.

Boosting can be viewed as function gradient descent.
Let $L(f)$ be a differentiable loss function defined on the empirical data sample, e.g. for squared error loss,

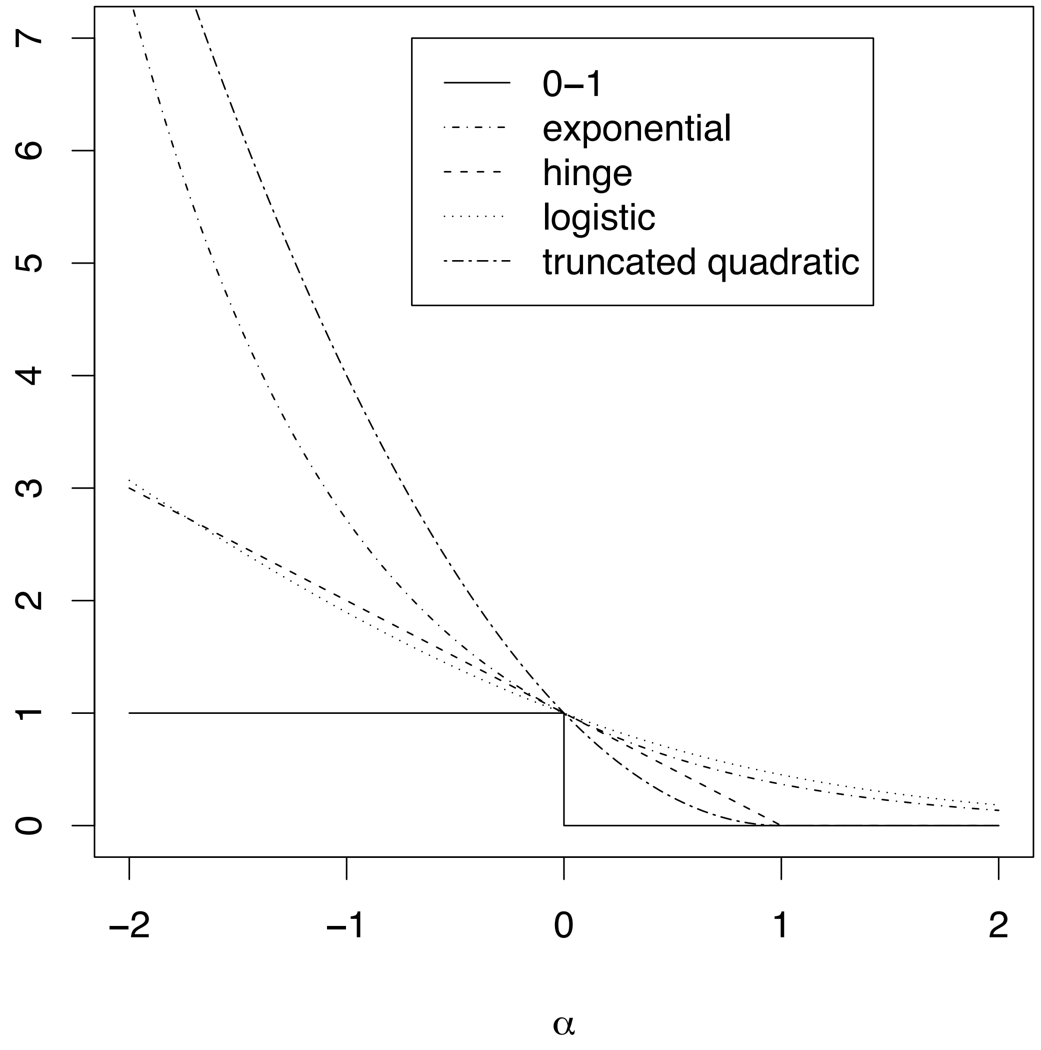
$$L(f) = n^{-1} \sum_{i=1}^{n} (Y_i - f(X_i))^2.$$

The Boosting algorithm can be viewed as functional gradient descent.

1. Fit a first function estimate from $\{(X_i, -\nabla L(f \equiv 0));\ i = 1, \ldots, n\}$ yielding $\hat{g}_1(\cdot)$. Denote by
$$\hat{f}_1(\cdot) = \nu \hat{g}_1(\cdot).$$
2. For $m = 2, 3, \ldots, M$ do:
Fit the gradient $(X_i, (-\nabla L)(\hat{f}_{m-1})) \to \hat{g}_m(\cdot)$ and set
$$\hat{f}_m(\cdot) = \hat{f}_{m-1}(\cdot) + \nu \hat{g}_m(\cdot).$$

For classification with $Y_i \in \{-1, 1\}$,

$$L(f) = n^{-1} \sum_{i=1}^{n} \phi(Y_i f(X_i)).$$



- Obtain L_2Boosting when using the quadratic loss function

$$L(f) = n^{-1} \sum_{i=1}^{n} (Y_i - f(X_i))^2.$$

- Obtain AdaBoost (the original boosting algorithm by Freund and Shapire) when using the exponential loss (for $Y \in \{-1, 1\}$)

$$L(f) = n^{-1} \sum_{i=1}^{n} \exp(-Yf(X_i)).$$

- Obtain LogitBoost when using the logistic loss function (again $Y \in \{-1, 1\}$),

$$L(f) = n^{-1} \sum_{i=1}^{n} \log(1 + \exp(-Yf(X_i))).$$

Boosting is implemented in package `mboost`.

```
> library(mboost)
> library(help=mboost)
> ?blackboost
blackboost              package:mboost              R Documentation
Gradient Boosting with Regression Trees

Description:
     Gradient boosting for optimizing arbitrary loss functions where
     regression trees are utilized as base learners.
Usage:
     ## S3 method for class 'formula':
     blackboost(formula, data = list(), weights = NULL, ...)
     ## S3 method for class 'matrix':
     blackboost(x, y, weights = NULL, ...)
     blackboost_fit(object, tree_controls =
         ctree_control(teststat = "max",
                       testtype = "Teststatistic",
                       mincriterion = 0,
                       maxdepth = 2),
         fitmem = ctree_memory(object, TRUE), family = GaussReg(),
         control = boost_control(), weights = NULL)
```

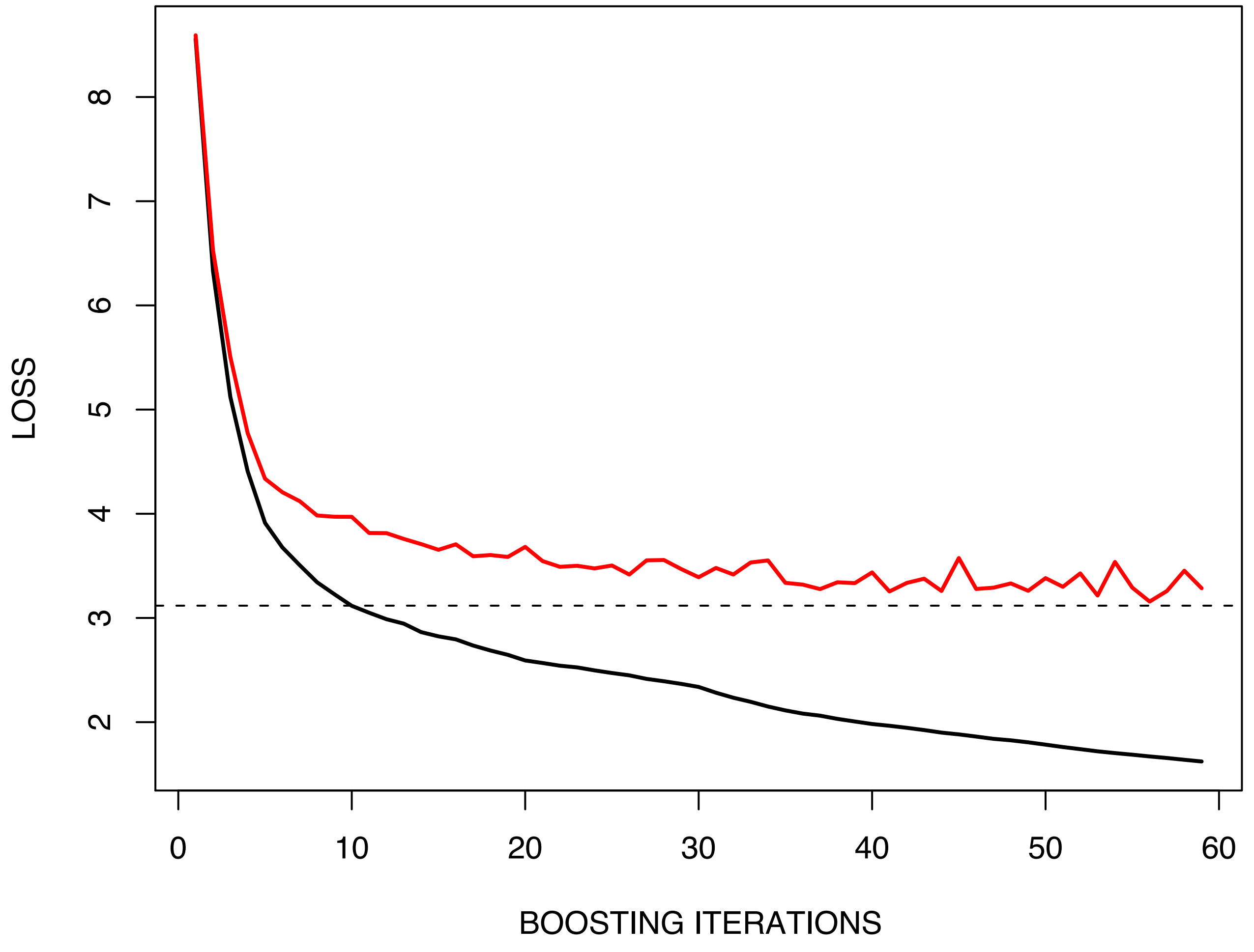
A simple cross-validation scheme.

```
library(mboost)
?blackboost                ## help function for tree boosting
n <- length(y)             ## number of observations

Mvec <- 1:500              ## Mvec is vector with various stopping times
nM <- length(Mvec)         ## number of possible stopping times
loss <- numeric(nM)        ## loss contains the training error
losscv <- numeric(nM)      ## losscv contains the cross-validated
                           ## test error

...
```

```
...
for (mc in 1:nM){          ## loop over stopping times (not efficient)
  yhat <- numeric(n)       ## yhat are the fitted values
  yhatcv <- numeric(n)     ## yhatcv the cross-validated fitted values

  M <- Mvec[mc]            ## use M iterations

  V <- 10                  ## 10-fold cross validation
                           ## indCV contains the ‘block’ in 1,...,10
                           ## each observation falls into
  indCV <- sample( rep(1:V,each=ceiling(n/V)), n)

  for (cv in 1:V){         ## loop over all blocks
    bb <- blackboost(y[indCV!=cv] ~ .,data=x[indCV!=cv,],
                control=boost_control(mstop=M))
                           ## predict the unused observations
    yhatcv[indCV==cv] <- predict(bb,x[indCV==cv,])
  }
  losscv[mc] <- sqrt(mean( (y-yhatcv)^2 ))   ## CV test error

  bb <- blackboost(y ~ .,data=x,control=boost_control(mstop=M))
  yhat <- predict(bb,x)
  loss[mc] <- sqrt(mean( (y-yhat)^2 ))       ## training error
}
```

Plot CV-test error in red as a function of the boosting iterations and training error in black.

```
matplot( cbind(loss,losscv), type="p",lwd=2,col=c(1,2),lty=1)
abline(h= sqrt(mean(( predict(rf)-y)^2)),lwd=1,lty=2)
```



Comparison with RF

Both RF and Boosting are tree ensembles.

- As RF, Boosting does not seem to overfit (the CV curve stays flat). This is not quite true, though: what is

  $$\lim_{m\to\infty} \hat{f}_m(X_i) \ ?$$

  Need to stop early (after having done M iterations)!
- The stopping parameter M needs to be adjusted by either
  - cross-validation, which is computationally expensive or
  - model selection, which does not work very well for trees as base learners (what are the degrees of freedom of a tree?)
- Predictive performance is very similar.
- Properties of Boosting (and why it is successful) are rather well understood (e.g. by bias reduction), but remain more of a mystery for RF.