MS1b Statistical Data Mining
Part 4: Supervised Learning
Ensemble Methods

Yee Whye Teh
Department of Statistics
Oxford

http://www.stats.ox.ac.uk/~teh/datamining.html

Outline

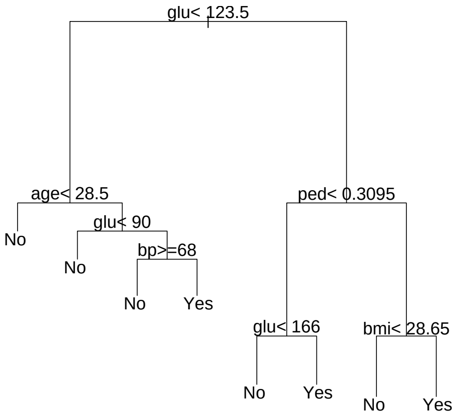
Outline

Bagging

An appeal of trees is their interpretability. Recall classification tree for Pima Indians example.

```
library(rpart)
library(MASS)
data(Pima.tr)                                         ## load data
Diabetes <- Pima.tr[,8]                               ## response
X <- Pima.tr[,-8]                                     ## predictor
tree <- rpart(Diabetes ~ ., data=X,
                   control=rpart.control(xval=10))) ## 10-fold CV
```

```
> plot(tree); text(tree)
```



Tree is very interpretable, selecting a subset of all predictor variables. Is the tree also ‘stable’ under small perturbations of the data or if we have slightly different training data? Can we do formal ‘significance testing’ as in linear models? How do we know we are not including irrelevant variables?

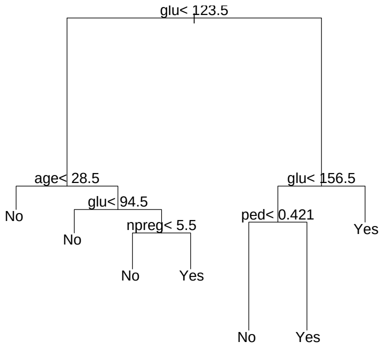
To fit the classification tree, we used all observations $i = 1, \dots, n$ with $n = 200$. What would the tree have looked like for a slightly different set of observations?
The Bootstrap (Efron, 79) is a natural way to assess the variance of estimators, fitting the tree repeatedly on so-called **bootstrap samples**. These are random sets of size n, where each element is drawn **with replacement** from the original n observations $\{1, \dots, n\}$.

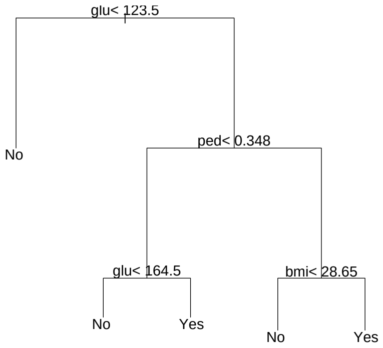
```
> n <- nrow(X)
> subsample <- sample(1:n, n , replace=TRUE)

> sort(subsample)
[1]  2 4 4 5 6 7 9 10 11 12 12 12 12 13 13 15 15 20 ...
```

Some of the original observations do not appear in the bootstrap sample (e.g. $i = 1$ or $i = 3$); some appear once (e.g. $i = 2$ or $i = 5$) and some twice or more often (e.g. $i = 4$).
Fit the tree on these resampled observations.

```
> tree_boot <- rpart(Diabetes ~ ., data=X,  subset=subsample,
                      control=rpart.control(xval=10)) ## 10-fold CV
```

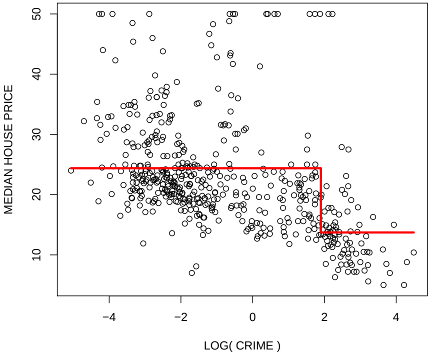
Doing this twice, we get the two following trees, each fitted on a different (random) subset of the data.

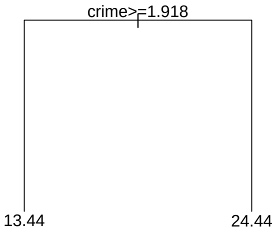




Classification trees are typically not very stable under subsampling of the data. This affects both interpretability and also prediction.
We might for example be suspicious of a particular classification (e.g. “No”) if a large fraction of resampled trees is classifying otherwise (classifying as “Yes”).

Can also look at regression trees.
Take the previous example of the Boston Housing data, trying to predict median house prices, based on the (univariate) predictor variable crime rate.





Fit a stump (the simplest tree – just a root node) to the data. This yields the fitted function $\hat{Y}(x)$, shown as a red solid line.

We fitted (tree) predictor $\hat{Y}(x)$ on the observations

$$(X_1, Y_1), \ldots, (X_n, Y_n), \qquad i = 1, \ldots, n.$$

Assess the variance of the fitted function $\hat{Y}(x)$ by taking $B = 20$ random subsamples of the original data. Fit bootstrap estimators (trees)

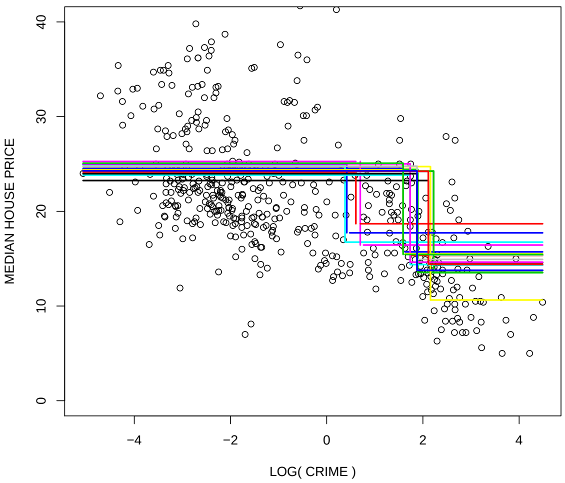
$$\hat{Y}^{*,b}(x), \qquad b = 1, \ldots, B$$

where each tree $\hat{Y}^{*,b}$ is fitted on the resampled data

$$(X_{j_1}, Y_{j_1}), \ldots, (X_{j_n}, Y_{j_n}), \qquad i = 1, \ldots, n,$$

each index j_k, $k = 1, \ldots, n$, being chosen at random from the set $\{1, \ldots, n\}$ with replacement.

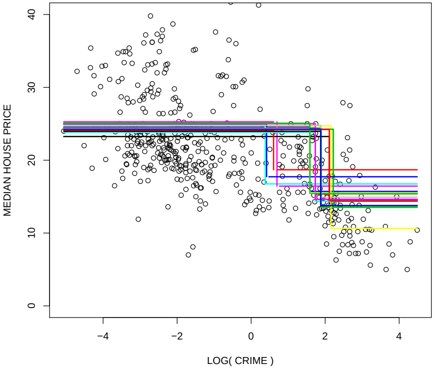
Trees $\hat{Y}^{*,1}, \ldots, \hat{Y}^{*,20}$ each fitted on a different (random) bootstrap sample of the original $n = 500$ observations.

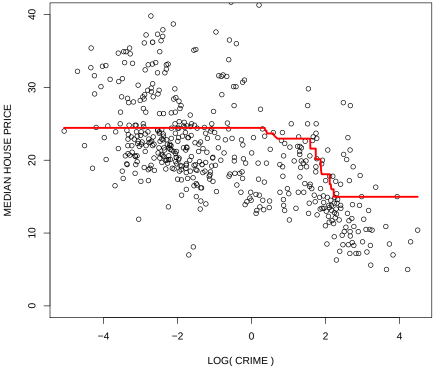


The variance of the fit $\hat{Y}^{*}$ is high in the region where the splitpoint is placed.

Idea of Bagging (**B**ootstrap **Agg**regation): average across all B trees fitted on different bootstrap samples,

$$\hat{Y}_{Bag} = \frac{1}{B} \sum_{i=1}^{B} \hat{Y}^{*,i}.$$





Empirically, Bagging seems to reduce the variance of $\hat{Y}$, e.g.

$$E\left((\hat{Y} - E(\hat{Y}))^2\right) \;\; \geq \;\; E\left((\hat{Y}_{Bag} - E(\hat{Y}_{Bag}))^2\right).$$

Bagged trees are an example of an **ensemble of trees**, as prediction is based on many individual predictors.
In summary, bagging trees has the following algorithm. Let $\hat{Y}$ be a tree (or other predictor), based on samples $(X_1, Y_1), \ldots, (X_n, Y_n)$.

1. Draw indices $(j_1, \ldots, j_n)$ from the set $\{1, \ldots, n\}$ with replacement. Fit the tree $\hat{Y}^*$ based on samples
$$(X_{j_1}, Y_{j_1}), \ldots, (X_{j_n}, Y_{j_n}).$$
2. Repeat first step B times to obtain
$$\hat{Y}^{*,1}, \ldots, \hat{Y}^{*,B}.$$
3. Bagged estimator is
$$\hat{Y}_{Bag} = \frac{1}{B} \sum_{b=1}^{B} \hat{Y}^{*,b}.$$

Variance reduction

Suppose, in an ideal world, we could instead base trees $\tilde{Y}^{*,b}$, $b = 1, \ldots, B$ on n samples drawn from the (unknown) joint distribution of (X, Y), instead of resampling from the original n observations.
The bagged estimator is then

$$\tilde{Y}_{Bag} \; = \; \frac{1}{B} \sum_{b=1}^{B} \tilde{Y}^{*,b}.$$

For $B \to \infty$ (many bootstrap samples),

$$\tilde{Y}_{Bag} \; \to \; E(\hat{Y}),$$

where the expectation is with respect to the random sample of n observations and $\hat{Y}$ is the standard estimator (tree) fitted on these n observations.

Now compare the squared error loss of $\tilde{Y}_{Bag}$ with the loss of the original tree estimator $\hat{Y}$,

$$E\big((Y-\hat{Y})^2\big),$$

where both $\hat{Y}=\hat{Y}(x)$ and $\hat{Y}_{Bag}=\hat{Y}_{Bag}(x)$ are evaluated at some $x\in\mathbb{R}^p$ and the expectation is with respect to a random new observation Y and a new training sample on which $\hat{Y}$ is fitted.
Using $\tilde{Y}_{Bag}\to E(\hat{Y})$ for $B\to\infty$,

$$\begin{aligned} E\big((Y-\hat{Y})^2\big) &= E\big((Y-\tilde{Y}_{Bag}+\tilde{Y}_{Bag}-\hat{Y})^2\big) \\ &= E\big((Y-\tilde{Y}_{Bag})^2\big)+E\big((\tilde{Y}_{Bag}-\hat{Y})^2\big) \\ &\geq E\big((Y-\tilde{Y}_{Bag})^2\big). \end{aligned}$$

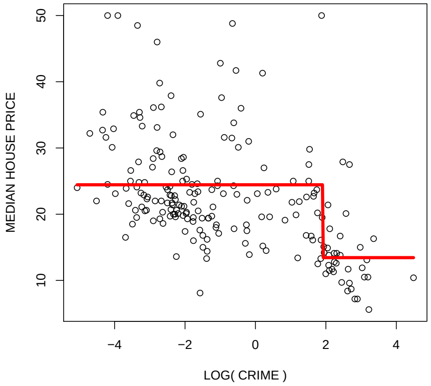
The (population) bagging estimator $\tilde{Y}_{Bag}$ thus reduced the squared error loss by eliminating the variance of $\hat{Y}$ around its mean $E(\hat{Y})$.

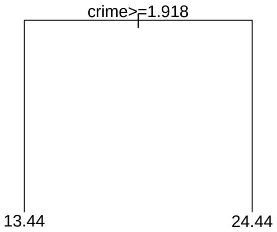
The variance reduction still applies if the idealized (population) estimate $\tilde{Y}_{Bag}$ is replaced by the actual bagging estimator $\hat{Y}$. This variance reduction is traded for a (small) increase in the bias in the procedure.
Bagging helps thus most for ‘flexible’ estimators $\hat{Y}$ which have a high variance

$$E\big((\hat{Y} - E(\hat{Y}))^2\big).$$

For trees, this means that bagging has a very beneficial effect on trees with a large size (number of leaf nodes), whereas the benefit of bagging on trees with small size is much less pronounced.

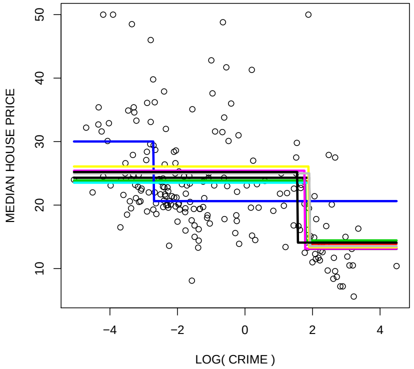
Look again at previous example of predicting house prices, using crime rate as the univariate predictor.



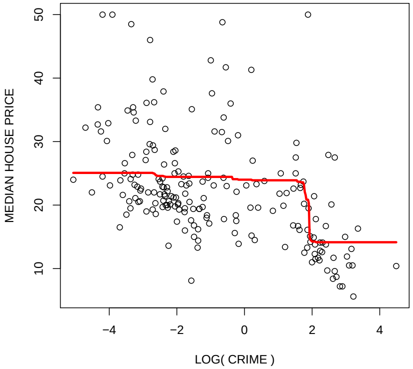


Fitting a single tree with depth $d = 1$ (a stump).

Bagged stumps $\hat{Y}^{*,b}$, $b = 1, \ldots, 10$.

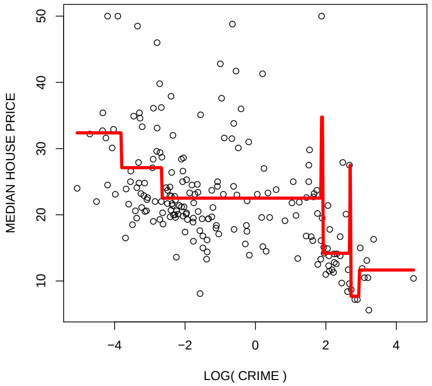


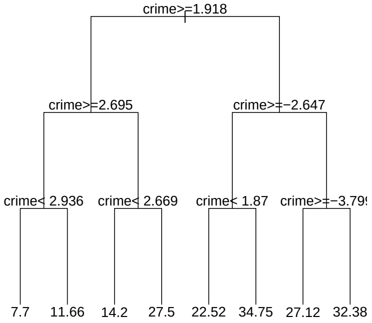
Averaged bagged estimator $\hat{Y}_{Bag}$.



A stump $\hat{Y}$ has (relative to larger trees) a low variance (and a high bias). Bagging leads to a small but not a dramatic improvement.

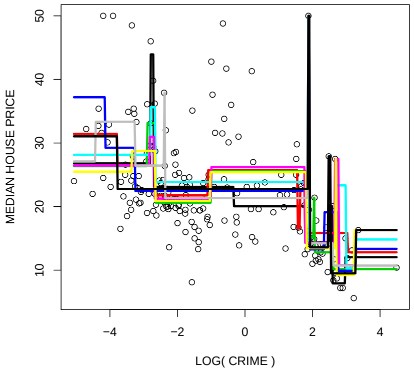
Now fit a tree with depth $d = 3$.



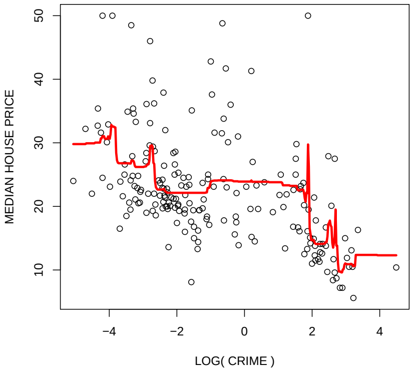


The fit of a single tree has a high variance and will have poor performance (when trying to predict new observations).

Bagged trees of depth $d = 3$, $\hat{Y}^{*,b}$, $b = 1, \ldots, 10$.



Averaged bagged estimator $\hat{Y}_{Bag}$.



As $\hat{Y}$ has a high variance (and a low bias), bagging leads to a large improvement.
Even though **improvement** through bagging is largest in general for trees with large depths, the optimal tree depth (yielding smallest prediction error when bagging) is not obvious a priori.

Out-of-bag test error estimation

To answer this question, we need again a good approximation to the test error (here for the squared error loss function L),

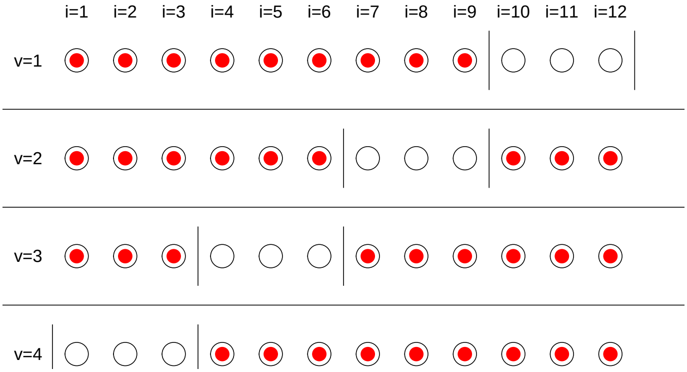
$$R_{test} := E(L(Y, \hat{Y}_{Bag})),$$

where the expectation is with respect to new random pairs (X, Y) and $\hat{Y}_{Bag} = \hat{Y}_{Bag}(X)$, to

- tune the parameters of the algorithm (e.g. select depth of the tree)
- or assess the true performance (and compare with other approaches).

Could compute generalization error $\widehat{R}_{test}$ by cross-validation (CV), as discussed previously.

Here schematic illustration of $V = 4$-fold CV for $n = 12$ samples.

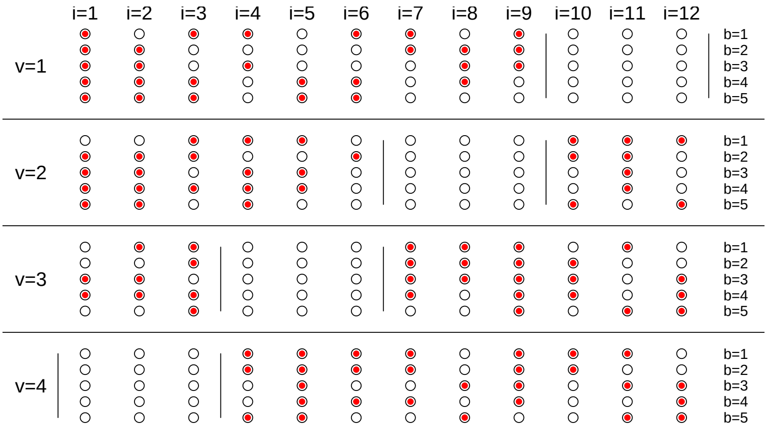


For each $v = 1, \ldots, V$,

- fit $\hat{Y}_{Bag}$ on the training samples, shown as red and filled dots.
- predict with this tree the left-out test observations, shown as white unfilled circles.

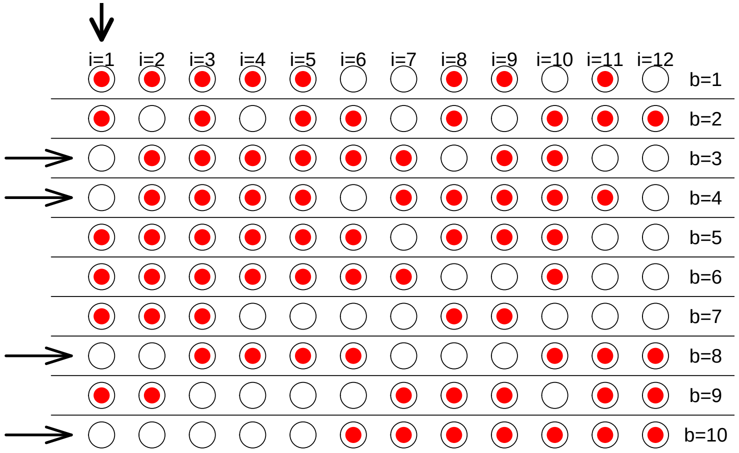
Compute the CV test error by averaging the loss across all test observations.

But to fit $\hat{Y}_{Bag}$ on the training samples for each $v = 1, \ldots, V$, need another set of B bootstrap samples on which the original tree is fitted (and whose average gives the $\hat{Y}_{Bag}$ for these training observations).



For each $v = 1, \ldots, V$, the tree needs to be fitted B times. In total, $V \times B$ fits are necessary. This can be very expensive computationally.
$\Rightarrow$ Out-of-bag estimation !

Idea: test on the “unused” data points in each bootstrap iteration to estimate the test error.



If fitting B bootstrap estimates $\hat{Y}^{*,b}$, to assess the prediction for $i = 1$, average only over such b, where observation $i = 1$ has not been used in fitting $\hat{Y}^{*,b}$.

Recall that, for B bootstrap samples $\hat{Y}^{*,b}$, the bagged estimator at observation i is given by $\hat{Y}_i := \hat{Y}_{Bag}(X_i)$,

$$\hat{Y}_i \;=\; \frac{1}{B} \sum_{b\in\{1,\ldots,B\}} \hat{Y}^{*,b}(X_i)$$

Instead, let now

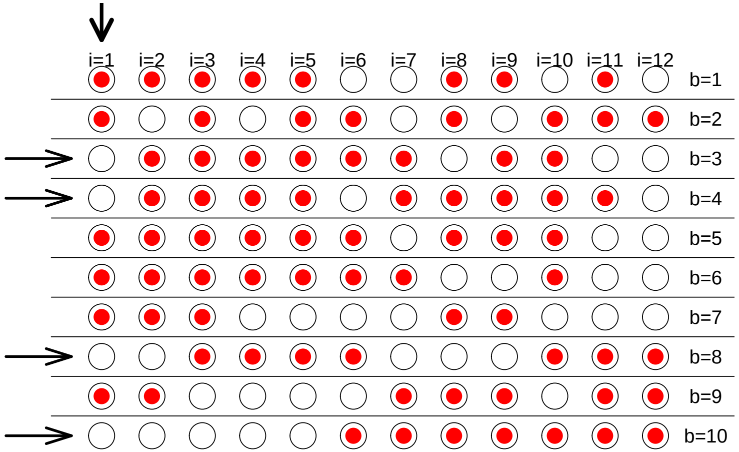
$$\hat{Y}_i^{oob} \;=\; \frac{1}{|\tilde{B}_i|} \sum_{b\in\tilde{B}_i} \hat{Y}^{*,b}(X_i),$$

where the sum is only taken over the set

$$\tilde{B}_i = \{b : X_i \text{ is not in training set}\} \subseteq \{1,\ldots,B\}.$$

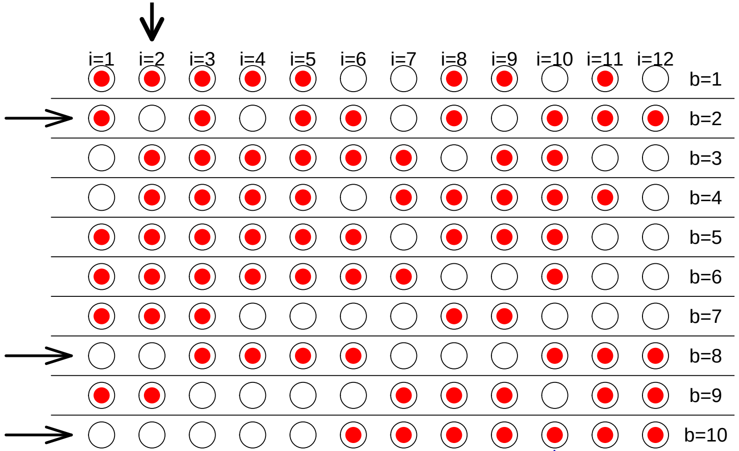
The estimate of the test error is then computed, as usual, by

$$\widehat{R}_{test} = n^{-1} \sum_{i=1}^{n} L(Y_i, \hat{Y}_i^{oob}).$$



In this example with $B = 10$ and $n = 12$, to get prediction for $i = 1$, average only over trees $\hat{Y}^{*,b}(X_1)$ with $b \in \{3, 4, 8, 10\}$, e.g.

$$\hat{Y}_1^{oob} = \frac{1}{4} \sum_{b \in \{3,4,8,10\}} \hat{Y}^{*,b}(X_1).$$



For predicting observation $i = 2$, average only over trees $\hat{Y}^{*,b}(X_2)$ with $b \in \{2, 8, 10\}$.

$$\hat{Y}_2^{oob} = \frac{1}{3} \sum_{b \in \{2,8,10\}} \hat{Y}^{*,b}(X_2).$$

We clearly need to average over many bootstrap samples in practice to get accurate results, e.g. $|\tilde{B}_i|$ needs to be reasonably large for all $i = 1, \ldots, n$.

What is the relation between $|\tilde{B}_i|$ and B?
The probability π^{oob} of an observation NOT being included in a bootstrap sample $(j_1, \dots, j_n)$ (and hence being 'out-of-bag') is, as all j_k for $k = 1, \dots, n$ are drawn with replacement from $\{1, \dots, n\}$,

$$\pi^{oob} = \prod_{i=1}^{n} (1 - \frac{1}{n}) \quad \overset{n \to \infty}{\to} \quad \exp(-1) \approx 0.367.$$

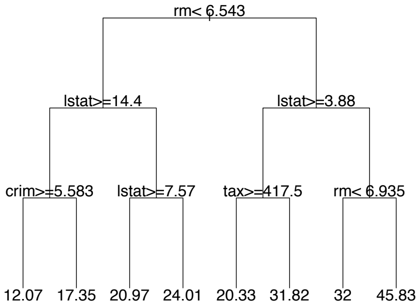
Hence $E(|\tilde{B}_i|) = \exp(-1) \cdot B \approx 0.367 \cdot B$ for all $i = 1, \dots, n$.
In practice, number of bootstrap samples B is typically between 200 and 1000, meaning that the number $|\tilde{B}_i|$ of out-of-bag samples will be approximately in the range $70 - 350$. The obtained test error estimate is asymptotically unbiased for large number B of bootstrap samples and large sample size n.

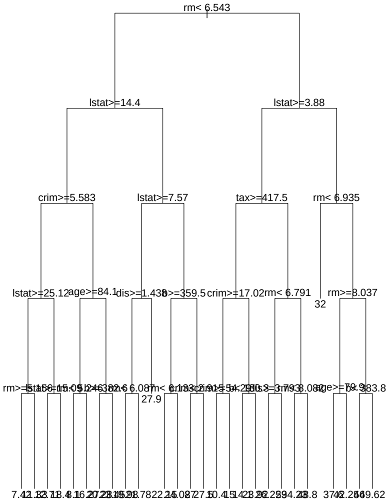
Apply out of bag estimation to select optimal tree depth and assess performance of bagged trees for Boston Housing data.
Use the entire dataset with $p = 13$ predictor variables. Fit first an ordinary tree of depth $d \in \{1, 2, 3, \ldots, 30\}$.

```
n <- nrow(BostonHousing)   ## n samples

X <- BostonHousing[,-14]
Y <- BostonHousing[,14]

maxdepth <- 3              ## fit here trees of depth 3
                           ## use function 'rpart' to fit tree
tree <- rpart(Y ~ ., data=X ,
       control=rpart.control(maxdepth=maxdepth,minsplit=2))
```

Plot trees of depth $d = 3$ and $d = 5$.

```
plot(tree,  margin=.1,  uniform=TRUE)
text(tree,  cex=1.3)
```





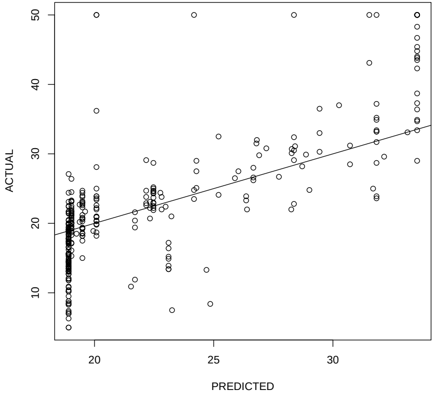
Bagging with $B = 100$ bootstrap samples, computing the out-of-bag (OOB) estimate of prediction error.

```
B <- 100
prediction_oob <- rep(0,length(Y))      ## vector with oob predictions
numbertrees_oob <- rep(0,length(Y))     ## how many oob trees
                                        ## for each sample ?

for (b in 1:B){                         ## loop over bootstrap samples
  subsample <- sample(1:n,n,replace=TRUE)       ## "in-bag" samples
  outofbag <- (1:n)[-subsample]                 ## "out-of-bag" samples

                                        ## fit tree on "in-bag" samples
  treeboot <- rpart(Y ~ ., data=X, subset=subsample,
        control=rpart.control(maxdepth=maxdepth,minsplit=2))

                                        ## predict on oob-samples
  prediction_oob[outofbag] <- prediction_oob[outofbag] +
                   predict(treeboot, newdata=X[outofbag,])
  numbertrees_oob[outofbag] <- numbertrees_oob[outofbag] +  1
}
## final oob-prediction is average across all "out-of-bag" trees
prediction_oob <- prediction_oob / numbertrees_oob
```

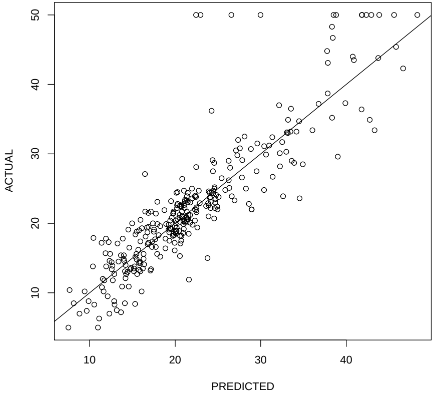
Plot out-of-bag predictions.

```
plot(prediction_oob, Y,  xlab="PREDICTED",  ylab="ACTUAL")
```

For depth $d = 1$.



For depth $d = 10$.



Out-of-bag estimates of test error

$$E\big((\hat{Y} - Y)^2\big)$$

as a function of tree depth d. Table shows CV-mean squared error loss (with out-of-bag prediction for the bagged estimator).

tree depth d	1	2	3	4	5	10	30
single tree $\hat{Y}$	60.7	44.8	32.8	31.2	27.7	26.5	27.3
bagged trees $\hat{Y}_{Bag}$	43.4	27.0	22.8	21.5	20.7	20.1	20.1

Without bagging, the optimal tree depth seems to be $d = 10$. With bagging, we could also take the depth up to $d = 30$.
Bagging strongly improves performance.
On the other hand, bagged trees cannot be displayed as nicely as single trees and some of the interpretability of trees is lost.

For classification, it is easily possible to construct (artifical) examples where bagging leads to a deterioration of performance.
Consider a two-class problem $Y \in \{0, 1\}$. Suppose all labels are truly $Y = 1$ and there is a random predictor $\hat{Y}$ which predicts

$$\hat{Y} = \begin{cases} 1 & \text{with probability } 0.3 \\ 0 & \text{with probability } 0.7 \end{cases} .$$

This classifier would have a misclassification error of 0.7.
Now bag this classifier by taking a mean $\hat{Y}_{Bag} = \sum_{b=1}^{B} \hat{Y}^{*,b}$ and classify by majority decision among all bagged trees, i.e. classify as $Y = 1$ if and only if $\hat{Y}_{Bag} > 0.5$.
The misclassification error of the bagged trees is now 1 and bagging made a bad predictor even worse.

Bagging trees typically improves prediction for real-life datasets. Consider the following datasets.

TABLE 1
Data set descriptions

Data set	**Training Sample size**	**Test Sample size**	**Variables**	**Classes**
Cancer	699	—	9	2
Ionosphere	351	—	34	2
Diabetes	768	—	8	2
Glass	214	—	9	6
Soybean	683	—	35	19
Letters	15,000	5000	16	26
Satellite	4,435	2000	36	6
Shuttle	43,500	14,500	9	7
DNA	2,000	1,186	60	3
Digit	7,291	2,007	256	10

Both trees and bagged trees (Forests) are fitted on these data.

The misclassification errors on the test sets for single trees and bagged trees ('Forests').

TABLE 2
Test set misclassification error (%)

Data set	**Forest**	**Single tree**
Breast cancer	2.9	5.9
Ionosphere	5.5	11.2
Diabetes	24.2	25.3
Glass	22.0	30.4
Soybean	5.7	8.6
Letters	3.4	12.4
Satellite	8.6	14.8
Shuttle $\times 10^3$	7.0	62.0
DNA	3.9	6.2
Digit	6.2	17.1

from Breiman: 'Statistical Modelling: the two cultures'.

Note that 'Forests' are not standard bagged trees, but so-called Random Forests, which employ additional randomization (more later).

Outline

Random Forests

The following misclassification errors compare “Random Forests” with single trees. RF are closely related to bagged trees.

TABLE 2
Test set misclassification error (%)

Data set	**Forest**	**Single tree**
Breast cancer	2.9	5.9
Ionosphere	5.5	11.2
Diabetes	24.2	25.3
Glass	22.0	30.4
Soybean	5.7	8.6
Letters	3.4	12.4
Satellite	8.6	14.8
Shuttle $\times 10^3$	7.0	62.0
DNA	3.9	6.2
Digit	6.2	17.1

from Breiman: “Statistical Modelling: the two cultures”.

Random Forests (Breiman, 2001) are widely believed to be the best "off-the-shelf" classifiers for high-dimensional data.

Similar to bagged decision trees with a few key differences:

- For each splitpoint, the search is not over all p variables but just over *mtry* variables (where e.g. *mtry* $= \lfloor p/3 \rfloor$)
- No pruning necessary. Trees can be grown until each node contains just very few observations (1 or 5).

Bagged decision trees can be seen as a special case of Random Forests (for *mtry*=p), if trees are not pruned, e.g. always grown to maximal depth.

Advantages of RF over bagged decision trees

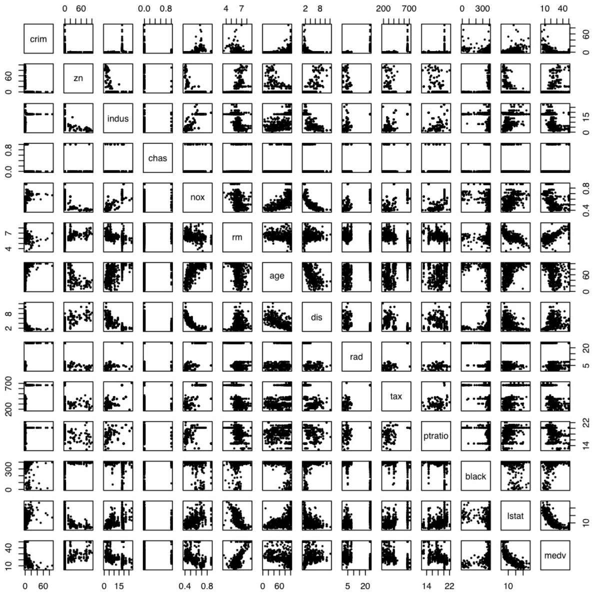
- better prediction (in general).
- almost no parameter tuning necessary with RF (although it still helps to vary the value of *mtry*). Tree depth needs to be chosen carefully with bagging, while we can always grow trees without pruning with RF.

Random Forests are implemented in package `randomForest`. Looking at the Boston Housing data again (and at the help page for `randomForest` first).

```
library(randomForest)
library(MASS)
data(Boston)

y <- Boston[,14]
x <- Boston[,1:13]

?randomForest
```

```
> randomForest          package:randomForest          R Documentation

Classification and Regression with Random Forest

Description:
     'randomForest' implements Breiman's random forest algorithm (based
     on Breiman and Cutler's original Fortran code) for classification
     and regression.  It can also be used in unsupervised mode for
     assessing proximities among data points.

Usage:
     ## S3 method for class 'formula':
     randomForest(formula, data=NULL, ..., subset, na.action=na.fail)
     ## Default S3 method:
     randomForest(x, y=NULL,  xtest=NULL, ytest=NULL, ntree=500,
                  mtry=if (!is.null(y) && !is.factor(y))
                  max(floor(ncol(x)/3), 1) else floor(sqrt(ncol(x))),
                  replace=TRUE, classwt=NULL, cutoff, strata,
                  sampsize = if (replace) nrow(x) else ceiling(.632*nrow(x
                  nodesize = if (!is.null(y) && !is.factor(y)) 5 else 1,
                  importance=FALSE, localImp=FALSE, nPerm=1,
                  proximity=FALSE, oob.prox=proximity,
                  norm.votes=TRUE, do.trace=FALSE,
                  keep.forest=!is.null(y) && is.null(xtest), corr.bias=FAL
                  keep.inbag=FALSE, ...)
```

Boston Housing data, again.



```
> rf <- randomForest(x,y)
> print(rf)
>
Call:
 randomForest(x = x, y = y)
              Type of random forest: regression
                    Number of trees: 500
No. of variables tried at each split: 4

          Mean of squared residuals: 10.26161
                    % Var explained: 87.84
```

Can plot the predicted values (out-of-bag estimation) vs. true values by

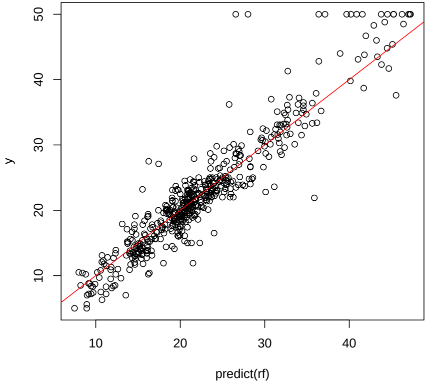
```
> plot( predict(rf), y)
> abline(c(0,1),col=2)
```

Same if treating the training data as new data

```
> plot( predict(rf,newdata=x), y)
```

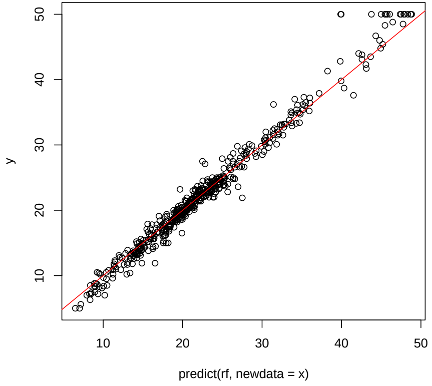
Out-of-bag error.

```
> plot( predict(rf), y)
> abline(c(0,1),col=2)
```



Training error.

```
> plot( predict(rf,newdata=x), y)
> abline(c(0,1),col=2)
```



Try `mtry` 2

```
> (rf <- randomForest(x,y,mtry=2))
Call:
 randomForest(x = x, y = y, mtry = 2)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 2

          Mean of squared residuals: 12.17176
                    % Var explained: 85.58
```

Try `mtry` 4

```
> (rf <- randomForest(x,y,mtry=4))
Call:
 randomForest(x = x, y = y, mtry = 4)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 4

          Mean of squared residuals: 10.01574
                    % Var explained: 88.14
```

And `mtry` 8 and 10.

```
> (rf <- randomForest(x,y,mtry=8))
Call:
 randomForest(x = x, y = y, mtry = 8)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 8

          Mean of squared residuals: 9.552806
                    % Var explained: 88.68

> > (rf <- randomForest(x,y,mtry=10))
Call:
 randomForest(x = x, y = y, mtry = 10)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 10

          Mean of squared residuals: 9.774435
                    % Var explained: 88.42
```

Choice of `mtry` makes little difference but is the only real tuning parameter.

Variable “importance”

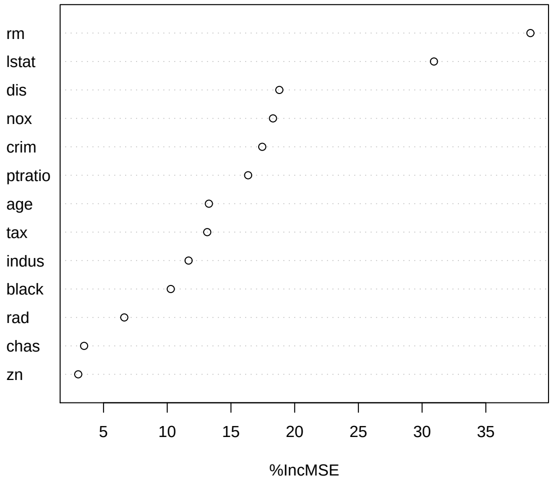
Despite the better predictive performance, single trees seem to have an edge over tree ensembles in terms of interpretability.
How do you interpret a forest of trees ?
Idea: denote by $\hat{e}$ the out-of bag estimate of the loss when using the original data samples.
For each variable $k \in \{1, \ldots, p\}$,

- permute randomly the k-th predictor variable to generate a new set of samples $(\tilde{X}_1, Y_1), \ldots, (\tilde{X}_n, Y_n)$.
- compute the out-of-bag estimate $\hat{e}_k$ of the prediction error with these new samples.

A measure of importance of variable k is then $\hat{e}_k - \hat{e}$, the increase in error rate due to random permutation of the k-th variable.

Example for Boston Housing data.

```
rf <- randomForest(x,y,importance=TRUE)
varImpPlot(rf)
```



Random Forests can be seen as an adaptive nearest neighbour technique. Let $P(x, x_i) \in [0, 1]$ be the proportion of trees for which an observation x falls into the same final leaf node as the original observation x_i. If every leaf node contains the same number of observations, the prediction of Random Forests (in regression mode) at predictor x is

$$\hat{Y}^{RF}(x) = \frac{\sum_{i=1}^{n} P(x, x_i) Y_i}{\sum_{i=1}^{n} P(x, x_i)},$$

which is a weighted (adaptive) nearest neighbour scheme and the weights are proportional to the proximities $P(x, x_i)$.
If the nodes contain different number of original observations, $P(x, x_i)$ is the weighted proportion of trees where x and x_i fall into the same leaf node, and weights are inversely proportional for each tree to the number of samples in the leaf node where x_i falls into.
For classification, the prediction will be the weighted majority vote, where again weights are proportional to the proximities $P(x, x_i)$.

Can visualize weights $P(x_i, x_j)$ for example by MDS.
Use Glass dataset as example.

```
> library(MASS)
> data(Glass)
> Glass[1:10,]
        RI    Na   Mg   Al    Si    K   Ca Ba   Fe Type
1  1.52101 13.64 4.49 1.10 71.78 0.06 8.75  0 0.00    1
2  1.51761 13.89 3.60 1.36 72.73 0.48 7.83  0 0.00    1
3  1.51618 13.53 3.55 1.54 72.99 0.39 7.78  0 0.00    1
4  1.51766 13.21 3.69 1.29 72.61 0.57 8.22  0 0.00    1
5  1.51742 13.27 3.62 1.24 73.08 0.55 8.07  0 0.00    1
6  1.51596 12.79 3.61 1.62 72.97 0.64 8.07  0 0.26    1
7  1.51743 13.30 3.60 1.14 73.09 0.58 8.17  0 0.00    1
8  1.51756 13.15 3.61 1.05 73.24 0.57 8.24  0 0.00    1
9  1.51918 14.04 3.58 1.37 72.08 0.56 8.30  0 0.00    1
10 1.51755 13.00 3.60 1.36 72.99 0.57 8.40  0 0.11    1
```

Try to predict glass type, based on chemical composition.

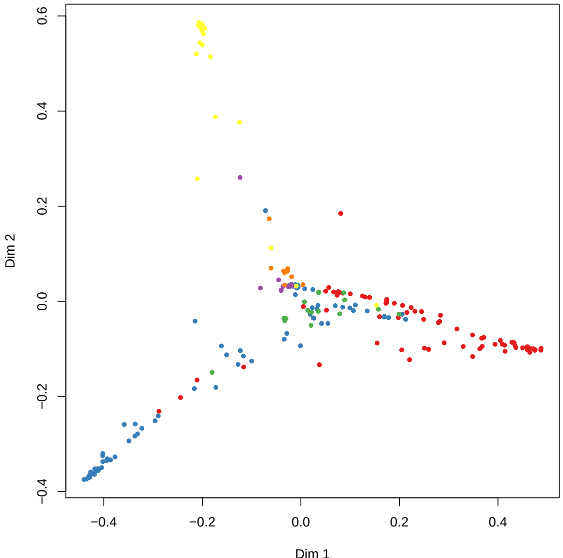
```
> X <- Glass[,-10]
> Y <- Glass[,10]
> rf <- randomForest(X,Y,ntree=500,proximity=TRUE,oob.prox=TRUE)
```

Calculate the proximities $P(x_i, x_j)$ based on out-of-bag observations.

```
> rf

Call:
 randomForest(x = X, y =  Y, proximity = TRUE, oob.prox = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 20.09%
Confusion matrix:
   1  2 3 4 5  6 class.error
1 63  6 1 0 0  0   0.1000000
2  9 60 2 2 2  1   0.2105263
3  7  3 7 0 0  0   0.5882353
4  0  3 0 9 0  1   0.3076923
5  0  2 0 0 7  0   0.2222222
6  1  3 0 0 0 25   0.1379310
```

Visualize proximities $P(x_i, x_j)$ for $i, j = 1, \ldots, n$ by MDS, using as distance matrix $D = 1 - P$.

```
> MDSplot(rf,Y)
```



Outline

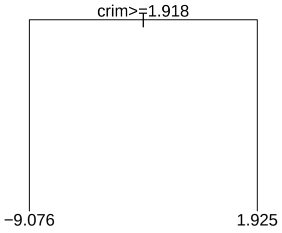
Boosting

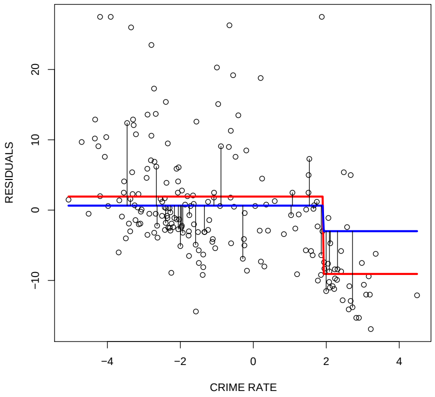
Boosting is a very different method to generate multiple predictions (function estimates) and combine them linearly. As with bagging, we have a base procedure yielding function estimates $\hat{g}(\cdot)$ (e.g. a tree algorithm).

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 $i = 1, \ldots, n$.

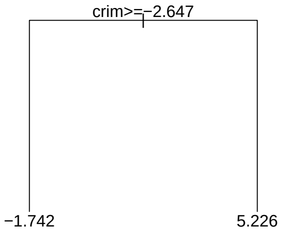
Example again Boston Housing data with single predictor variable crime rate.
First iteration: fit original observation with a stump.

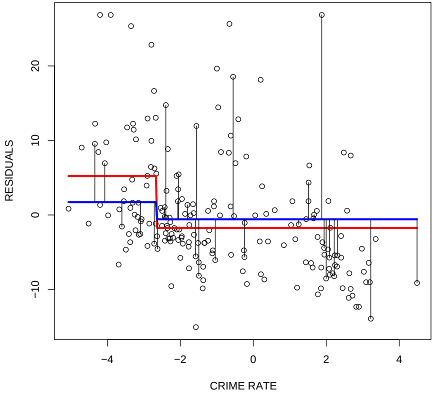




Fit of tree $\hat{g}_1(x)$ in red.
Shrunken fit $\nu\hat{g}_1(x)$ in blue.
Some residuals $U_i = Y_i - \nu\hat{g}_1(X_i)$ plotted with vertical bars. Fit these residuals in the next step.

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).



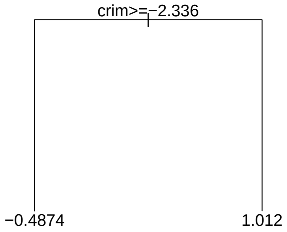


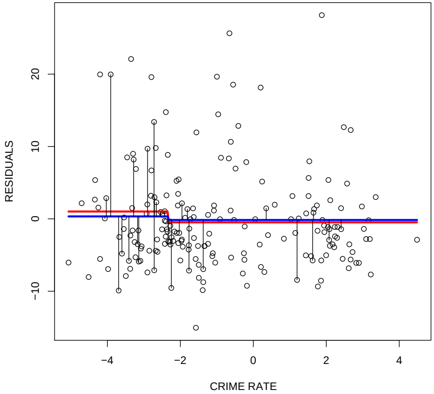
Fit of tree $\hat{g}_2(x)$ in red. Shrunken fit $\nu\hat{g}_2(x)$ in blue.
Some of the new residuals

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

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,

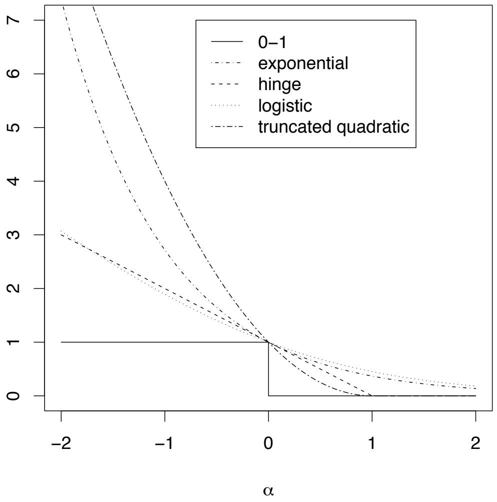
$$L(f) = n^{-1} \sum_{i=1}^{n} (Y_i - f(X_i))^2.$$

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)
```

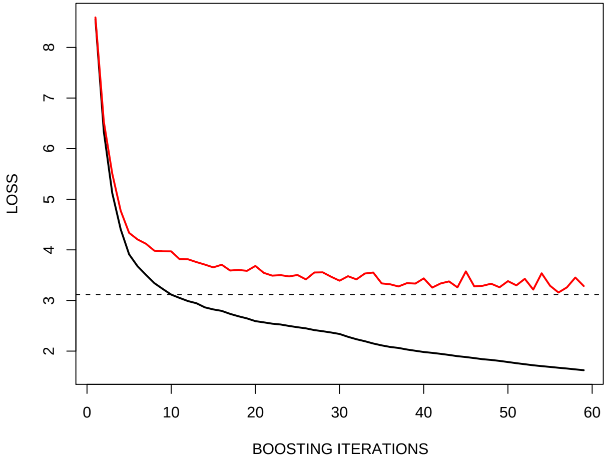
A simple cross-validation scheme.

```
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.