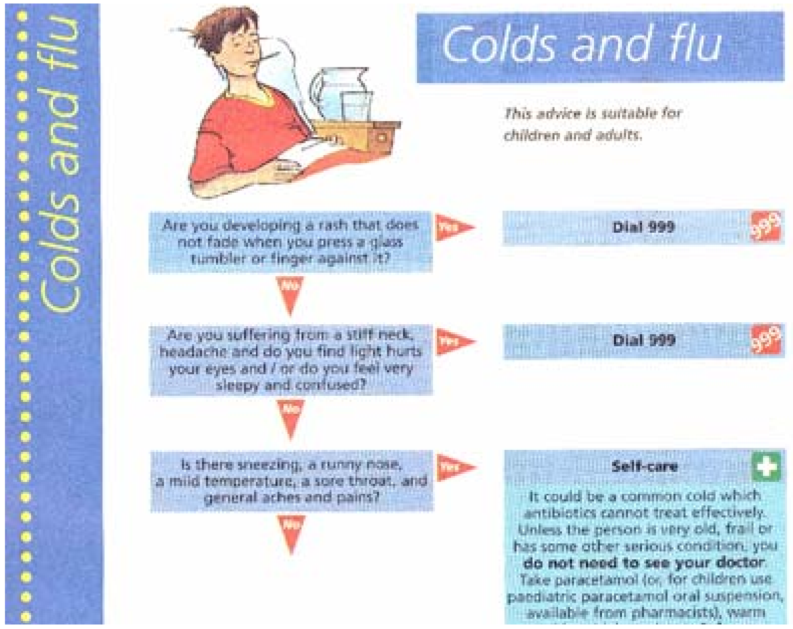
Example: NHS Direct Self-help Guide



Colds and flu

Colds and flu

This advice is suitable for children and adults.

Are you developing a rash that does not fade when you press a glass tumbler or finger against it? Yes → Dial 999

No

Are you suffering from a stiff neck, headache and do you find light hurts your eyes and / or do you feel very sleepy and confused? Yes → Dial 999

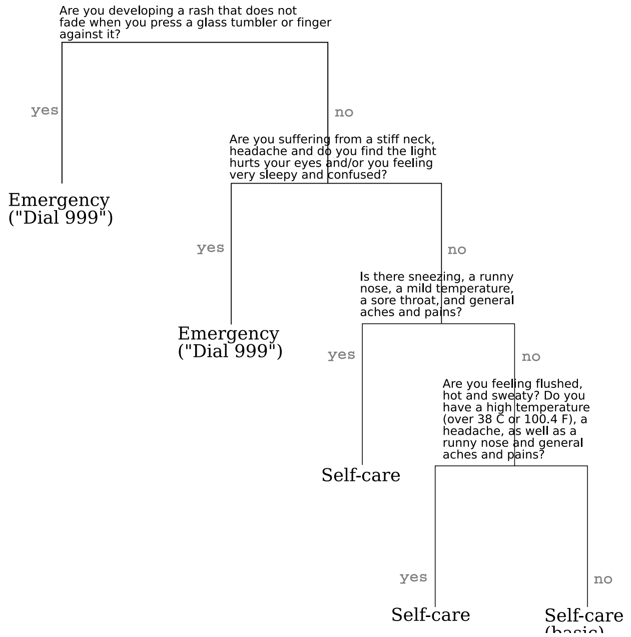
No

Is there sneezing, a runny nose, a mild temperature, a sore throat, and general aches and pains? Yes → Self-care

It could be a common cold which antibiotics cannot treat effectively. Unless the person is very old, frail or has some other serious condition, you **do not need to see your doctor**. Take paracetamol (or, for children use paediatric paracetamol oral suspension, available from pharmacists), warm

No

Example: NHS Direct Self-help Guide

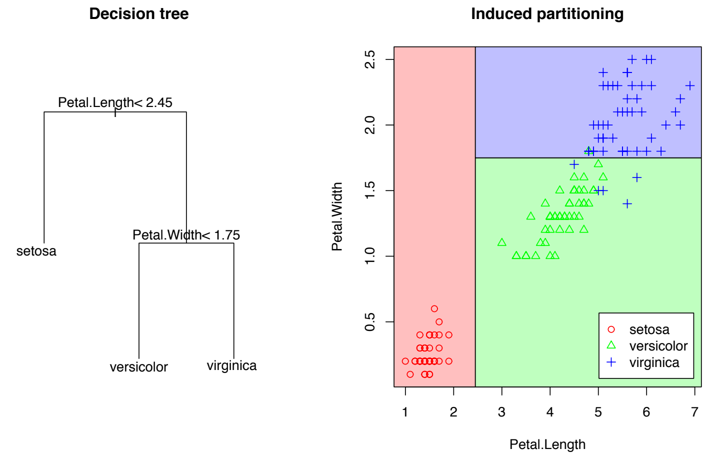

Are you developing a rash that does not fade when you press a glass tumbler or finger against it?
yes
no
Emergency ("Dial 999")
Are you suffering from a stiff neck, headache and do you find the light hurts your eyes and/or you feeling very sleepy and confused?
yes
no
Emergency ("Dial 999")
Is there sneezing, a runny nose, a mild temperature, a sore throat, and general aches and pains?
yes
no
Self-care
Are you feeling flushed, hot and sweaty? Do you have a high temperature (over 38 C or 100.4 F), a headache, as well as a runny nose and general aches and pains?
yes
no
Self-care
Self-care (basic)

Example: Iris Data

```
Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
         4.4         3.2          1.3         0.2     setosa
         5.9         3.0          5.1         1.8  virginica
         6.3         3.3          6.0         2.5  virginica
         5.3         3.7          1.5         0.2     setosa
         5.5         2.5          4.0         1.3 versicolor
         6.1         2.9          4.7         1.4 versicolor
         6.1         3.0          4.9         1.8  virginica
         5.7         2.8          4.5         1.3 versicolor
         5.4         3.0          4.5         1.5 versicolor
         4.8         3.4          1.6         0.2     setosa
         4.6         3.1          1.5         0.2     setosa
         4.9         3.1          1.5         0.2     setosa
         6.4         2.9          4.3         1.3 versicolor
         .......
```

Previously seen Iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

Example: Iris Data



Decision Trees

- A decision tree is a hierarchically organized structure, with each node splitting the data space into two halves based on value of a feature.
- Equivalent to a partition of $\mathbb{R}^p$ into R disjoint sets $(\mathcal{R}_1, \ldots, \mathcal{R}_R)$, where each $\mathcal{R}_j \subset \mathbb{R}^p$.
- On each region $\mathcal{R}_j$ the same decision/prediction is made $\hat{f}(x) = \beta_j$ for all $x \in \mathcal{R}_j$.
- Some terminology:
  - **Parent** of a node c is the node which have an arrow pointing into c.
  - **Children** of a node c are those nodes which have node c as a parent.
  - **Root node** is the top node of the tree; the only node without parents.
  - **Leaf nodes** are nodes which do not have children.
  - **Stumps** are trees with just the root node and two leaf nodes.
  - A K**-ary tree** is a tree where each node (except for leaf nodes) has K children. Usually working with binary trees ($K = 2$).
  - The **depth** of a tree is the maximal length of a path from the root node to a leaf node.

Classification and Regression Trees

- For regression problems, the parameterized function is

$$\hat{f}(x) = \sum_{j=1}^{R} \beta_j \mathbb{1}_{[x \in \mathcal{R}_j]},$$

  Using squared loss, optimal parameters are:

$$\hat{\beta}_j = \frac{\sum_i y_i \mathbb{1}_{[x_i \in \mathcal{R}_j]}}{\sum_i \mathbb{1}_{[x_i \in \mathcal{R}_j]}}$$

- For classification problems, the estimated probability of each class k in region $\mathcal{R}_j$ is simply:

$$\hat{\beta}_{jk} = \frac{\sum_i \mathbb{1}(y_i = k) \mathbb{1}_{[x_i \in \mathcal{R}_j]}}{\sum_i \mathbb{1}_{[x_i \in \mathcal{R}_j]}}$$

- These estimates can be regularized as well.

Partition Estimation

- Ideally, would like to find partition that achieves minimal risk: lowest mean-squared error for prediction or misclassification rate for classification.
- Number of potential partitions is too large to search exhaustively.
- 'Greedy' search heuristics for a good partition:
  - Start at root.
  - Determine best feature and value to split.
  - Recurse on children of node.
  - Stop at some point.

Growth Heuristic for Regression Trees

1. Start with $\mathcal{R}_1 = \mathbb{R}^p$.
2. For each feature $j = 1, \ldots, p$, for each value $v \in \mathbb{R}$ that we can split on:
   2.1 Split data set:

$$I_< = \{i : x_{ij} < v\} \qquad I_> = \{i : x_{ij} \geq v\}$$

   2.2 Estimate parameters:

$$\beta_< = \frac{\sum_{i \in I_<} y_i}{|I_<|} \qquad \beta_> = \frac{\sum_{i \in I_>} y_i}{|I_>|}$$

   2.3 Quality of split is the squared loss:

$$\sum_{i \in I_<} (y_i - \beta_<)^2 + \sum_{i \in I_>} (y_i - \beta_>)^2$$

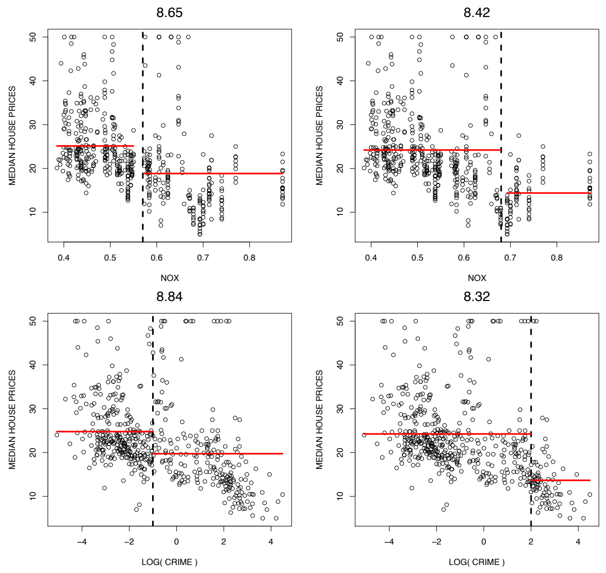
3. Choose split with minimal loss.
4. Recurse on both children, with datasets $(x_i, y_i)_{i \in I_<}$ and $(x_i, y_i)_{i \in I_>}$.

Boston Housing Data

crim	per capita crime rate by town
zn	proportion of residential land zoned for lots over 25,000
indus	proportion of non-retail business acres per town
chas	Charles River dummy variable (= 1 if tract bounds river; (
nox	nitric oxides concentration (parts per 10 million)
rm	average number of rooms per dwelling
age	proportion of owner-occupied units built prior to 1940
dis	weighted distances to five Boston employment centres
rad	index of accessibility to radial highways
tax	full-value property-tax rate per USD 10,000
ptratio	pupil-teacher ratio by town
b	1000(B - 0.63)^2 where B is the proportion of blacks by tc
lstat	percentage of lower status of the population
medv	median value of owner-occupied homes in USD 1000's

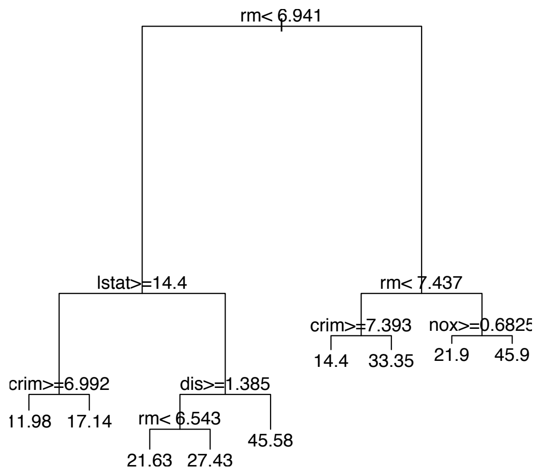
- Predict median house value.

Boston Housing Data



Boston Housing Data

- Overall, the best first split is on variable `rm`, average number of rooms per dwelling.
- Final tree contains predictions in leaf nodes.



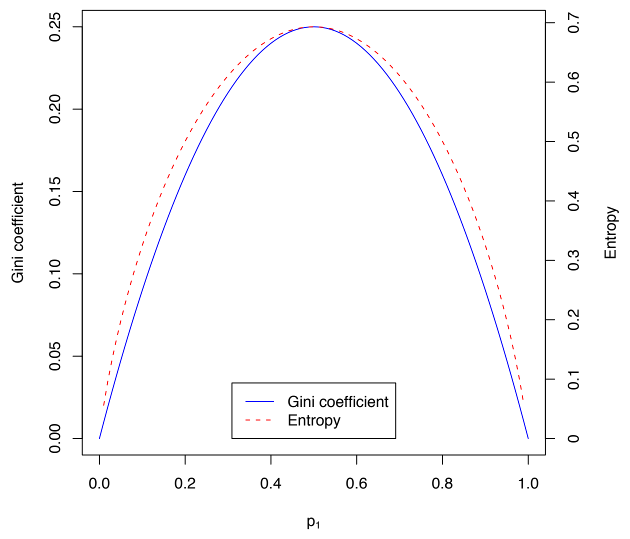
Growth Heuristics for Classification Trees

- For binary classification,

$$\hat{\beta}_{j1} = \frac{\sum_i \mathbb{1}(y_i = 1)\mathbb{1}_{[x_i \in \mathcal{R}_j]}}{\sum_i \mathbb{1}_{[x_i \in \mathcal{R}_j]}}$$

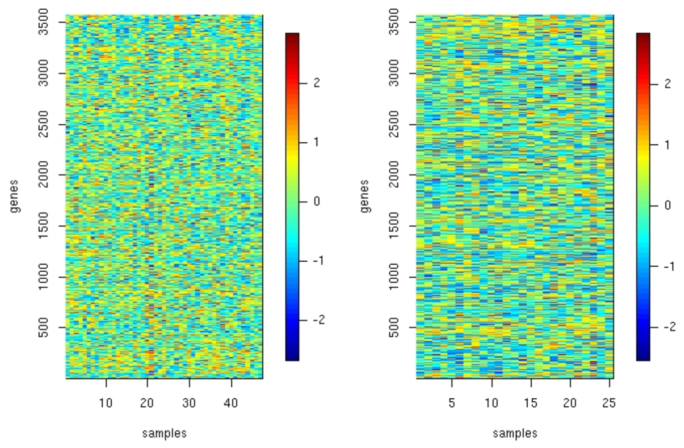
- A split is good if both sides are more "pure", i.e. $\hat{\beta}_{j1}$ is closer to 0 or 1.
- Different measures of node impurity:
  - Misclassification error: $1 - \max\{\hat{\beta}_{j1}, 1 - \hat{\beta}_{j1}\}$.
  - Gini Index: $2\hat{\beta}_{j1}(1 - \hat{\beta}_{j1})$.
  - Entropy: $-\hat{\beta}_{j1} \log \hat{\beta}_{j1} - (1 - \hat{\beta}_{j1}) \log(1 - \hat{\beta}_{j1})$.
- Typically prefer Gini and entropy: differentiable and produces purer nodes.
- Extension to multi-class:
  - Misclassification error: $1 - \max_k \hat{\beta}_{jk}$.
  - Gini Index: $\sum_{k=1}^K \hat{\beta}_{jk}(1 - \hat{\beta}_{jk})$.
  - Entropy: $-\sum_{k=1}^K \hat{\beta}_{jk} \log \hat{\beta}_{jk}$.
- Stops once a node has insufficient number of items, or is pure.

Growth Heuristics for Classification Trees



Misclassification error?

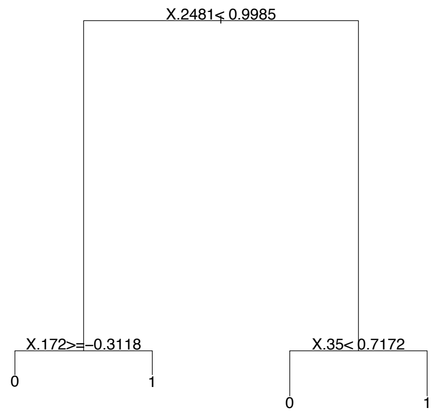
Example: Leukemia Prediction



Leukemia Dataset: Expression values of 3541 genes for 47 patients with Leukemia ALL subtype (left) and 25 patients with AML (right).

Example: Leukemia Prediction

- Tree found is of depth 2.
- Very interpretable as it selects 3 out of 4088 genes and bases prediction only on these.



Example: Pima Indians Diabetes Dataset

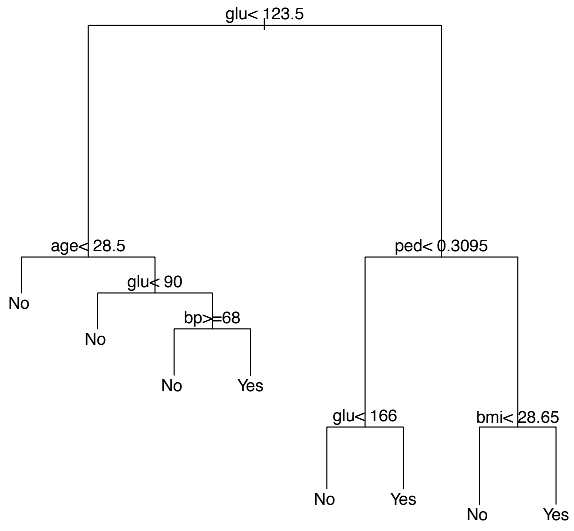
- The subjects: women who were at least 21 years old, of Pima Indian heritage living near Phoenix, Arizona.
- Tested for diabetes according to World Health Organisation criteria.
- Features:
  - number of pregnancies (npreg)
  - plasma glucose concentration (glu)
  - diastolic blood pressure (bp)
  - tricep skin fold thickness (skin)
  - body mass index(bbi)
  - diabetes pedigree function (ped)
  - age (age)

Example: Pima Indians Diabetes Dataset

```
> library(rpart)
> library(MASS)
> data(Pima.tr)
> rp <- rpart(Pima.tr[,8] ~ ., data=Pima.tr[,-8])
> rp
n= 200

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 200 68 No (0.66000000 0.34000000)
   2) glu< 123.5 109 15 No (0.86238532 0.13761468)
     4) age< 28.5 74  4 No (0.94594595 0.05405405) *
     5) age>=28.5 35 11 No (0.68571429 0.31428571)
      10) glu< 90 9  0 No (1.00000000 0.00000000) *
      11) glu>=90 26 11 No (0.57692308 0.42307692)
        22) bp>=68 19  6 No (0.68421053 0.31578947) *
        23) bp< 68 7  2 Yes (0.28571429 0.71428571) *
   3) glu>=123.5 91 38 Yes (0.41758242 0.58241758)
     6) ped< 0.3095 35 12 No (0.65714286 0.34285714)
      12) glu< 166 27  6 No (0.77777778 0.22222222) *
      13) glu>=166 8  2 Yes (0.25000000 0.75000000) *
     7) ped>=0.3095 56 15 Yes (0.26785714 0.73214286)
      14) bmi< 28.65 11  3 No (0.72727273 0.27272727) *
      15) bmi>=28.65 45  7 Yes (0.15555556 0.84444444) *
```

Example: Pima Indians Diabetes Dataset

```
> plot(rp); text(rp)
```



Model Complexity

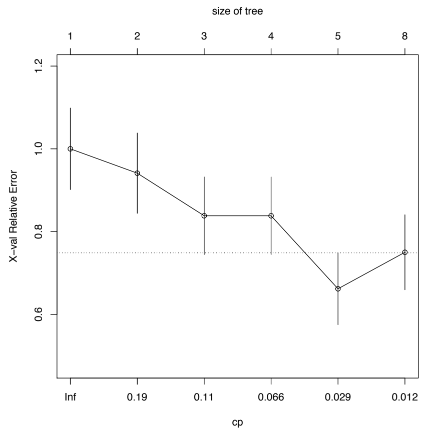
- When should tree growing be stopped?
- Will need to control complexity to prevent overfitting, and in general find optimal tree size with best predictive performance.
- Consider a regularized objective

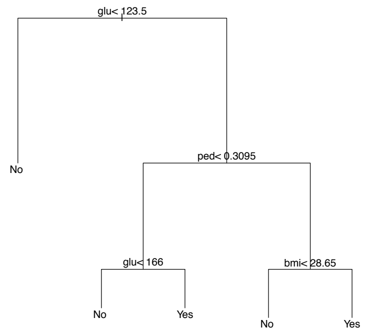
$$R(T) + C \times \text{size}(T)$$

  - Grow the tree from scratch and stop once the criterion objective starts to increase.
  - First grow the full tree and prune nodes (starting at leaves), until the objective starts to increase.
- Second option is preferred as the choice of tree is less sensitive to “wrong” choices of split points and variables to split on in the first stages of tree fitting.
- Use cross validation to determine optimal C.

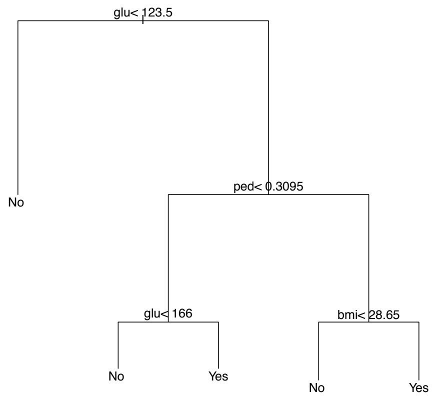
Model Complexity

```
> rp <- rpart(Pima.tr[,8] ~ ., data=Pima.tr[,-8],
                   control=rpart.control(xval=10)) ## 10-fold CV
> plotcp(rp)
> rp2 <- prune.rpart(rp,.029)
> plot(rp2); text(rp2)
```





Model Variability

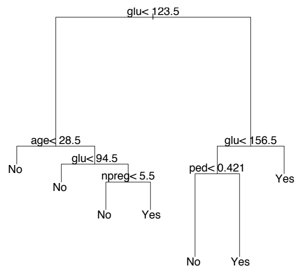


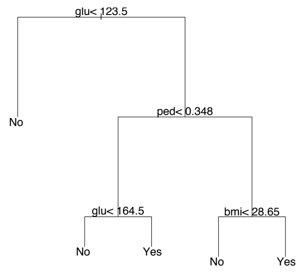
- Is the tree ‘stable’ if training data were slightly different?

Bootstrap

- The **bootstrap** is a way to assess the variance of estimators.
- Fit multiple trees, each on a **bootstrap sample**. This is a data set obtained by **sampling with replacement** n times from training set.

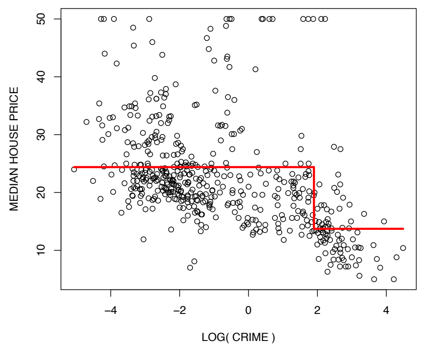
```
> n <- nrow(Pima.tr)
> bss <- sample(1:n, n , replace=TRUE)
> sort(bss)
[1]   2  4  4  5  6  7  9 10 11 12 12 12 12 13 13 15 15 20 ...

> tree_boot <- rpart(Pima.tr[bss,8] ~ ., data=Pima.tr[bss,-8],
                     control=rpart.control(xval=10)) ## 10-fold CV
```

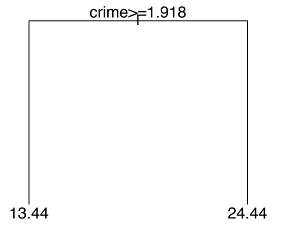




Bootstrap for Regression Trees

- Regression for Boston housing data.
- Predict median house prices based only on crime rate.
- Use decision **stump**—the simplest tree with a single split at root.



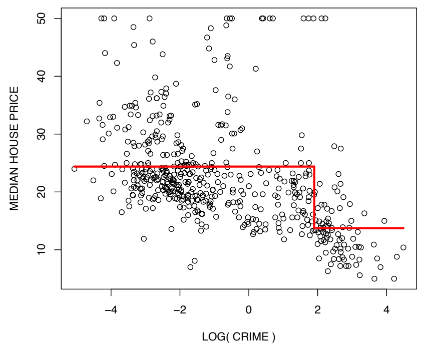


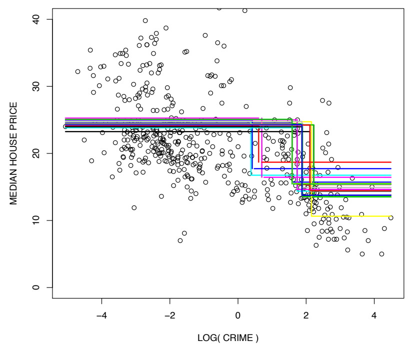
Bootstrap for Regression Trees

- We fit a predictor $\hat{f}(x)$ on the data $(x_i, y_i)_{i=1}^n$.
- Assess the variance of $\hat{f}(x)$ by taking $B = 20$ bootstrap samples of the original data, and obtaining bootstrap estimators

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

- Each tree $\hat{f}^b$ is fitted on the resampled data $(x_{j_i}, y_{j_i})_{i=1}^n$ where each j_i is chosen randomly from $\{1, \ldots, n\}$ with replacement.





Bagging

- **Bagging** (**B**ootstrap **Agg**regation): average across all B trees fitted on different bootstrap samples.

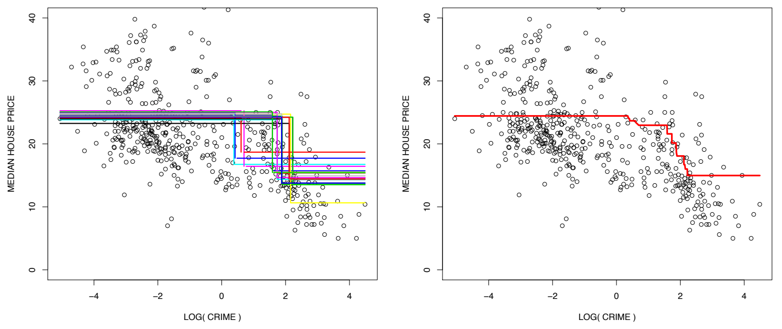
1. For $b = 1, \ldots, B$:
   1.1 Draw indices $(j_1, \ldots, j_n)$ from the set $\{1, \ldots, n\}$ with replacement.
   1.2 Fit the model, and form predictor $\hat{f}^b(x)$ based on bootstrap sample

$$(x_{j_1}, y_{j_1}), \ldots, (x_{j_n}, y_{j_n})$$

2. Form bagged estimator

$$\hat{f}_{Bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x)$$

Bagging



- Bagging smooths out the drop in the estimate of median house prices.
- Empirically, bagging seems to reduce the variance of $\hat{f}$, i.e.

$$\mathbb{E}\big[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2 | X = x\big] \quad \geq \quad \mathbb{E}\big[(\hat{f}_{Bag}(x) - \mathbb{E}[\hat{f}_{Bag}(x)])^2 | X = x\big]$$

Variance Reduction in Bagging

- Suppose, in an ideal world, our estimators $\hat{f}^b$ are based on independent samples of size n from the true joint distribution of X, Y.
- The aggregated estimator would then be

$$\hat{f}_{ag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x) \to \bar{f}(x) = \mathbb{E}[\hat{f}(x)] \quad \text{as } B \to \infty$$

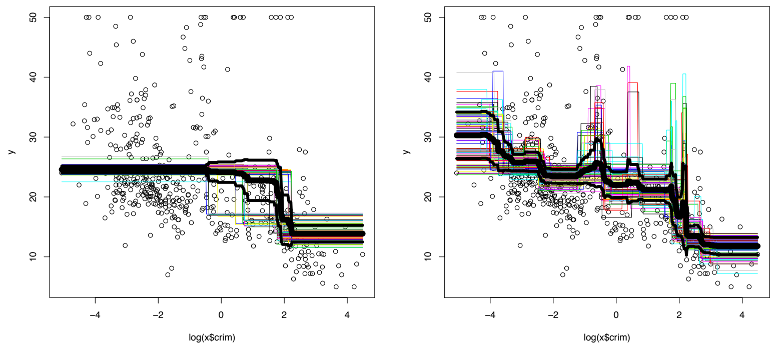
where expectation is with respect to datasets of size n.
- The squared-loss is:

$$\begin{aligned}\mathbb{E}[(Y - \hat{f}_{ag}(x))^2 | X = x] &= \mathbb{E}[(Y - \bar{f}(x))^2 | X = x] + \mathbb{E}[(\bar{f}(x) - \hat{f}_{ag}(x))^2 | X = x] \\ &\to \mathbb{E}[(Y - \bar{f}(x))^2 | X = x]\end{aligned}$$

Aggregation reduces the squared loss by eliminating variance of $\hat{f}(x)$.
- In bagging, variance reduction still applies at the cost of a small increase in bias.
- Bagging is most useful for flexible estimators with high variance (and low bias).

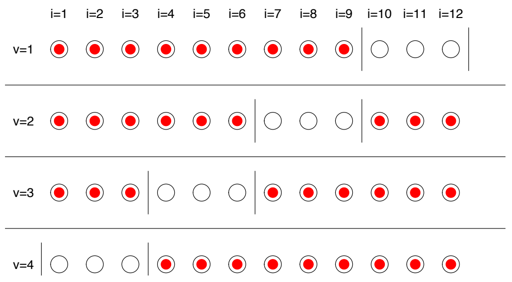
Variance Reduction in Bagging

- Deeper trees have higher complexity and variance.
- Compare bagging trees of depth 1 and 3.



Out-of-bag Test Error Estimation

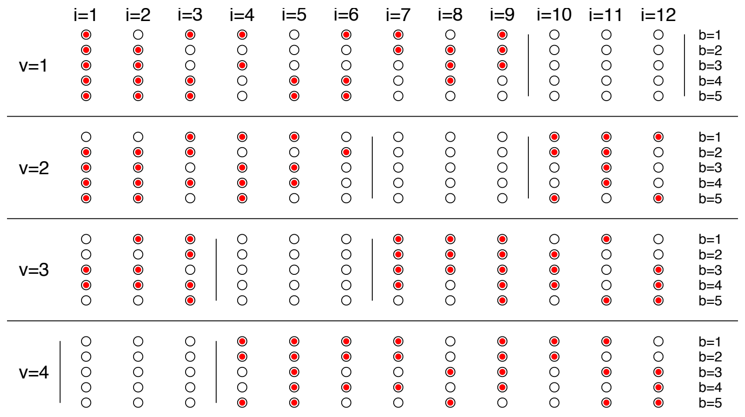
- How well does bagging to? Can we estimate generalization performance, and tune hyperparameters?
- Answer 1: cross-validation.



- For each $v = 1, \ldots, V$,
  - fit $\hat{f}_{Bag}$ on the training samples.
  - predict on validation set.
- Compute the CV error by averaging the loss across all test observations.

Out-of-bag Test Error Estimation

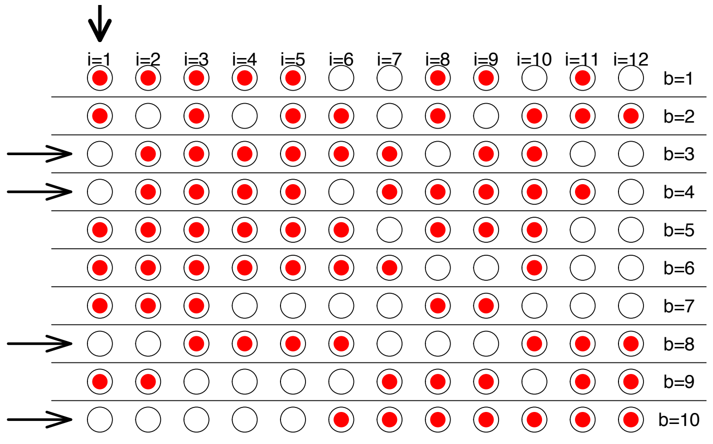
- But to fit $\hat{f}_{Bag}$ on the training set for each $v = 1, \ldots, V$, we need another set of B bootstrap samples!



- Answer 2: **Out-of-bag** test error estimation.

Out-of-bag Test Error Estimation

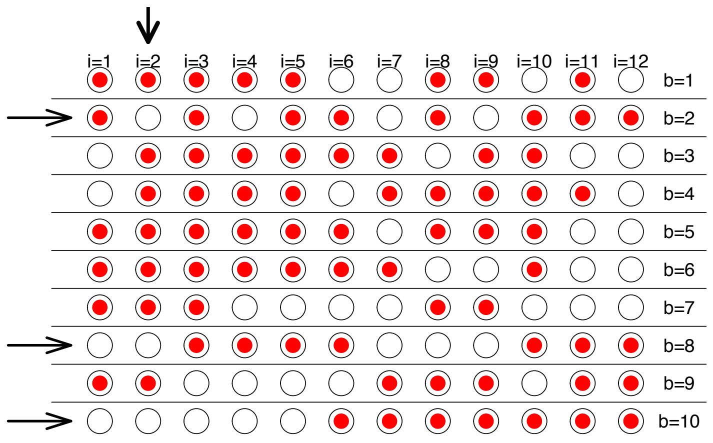
- Idea: test on the “unused” data points in each bootstrap iteration to estimate the test error.



$$\hat{f}^{oob}(x_1) = \frac{1}{4} \sum_{b \in \{3,4,8,10\}} \hat{f}^{b}(x_1)$$

Out-of-bag Test Error Estimation

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



$$\hat{f}^{oob}(x_2) = \frac{1}{3} \sum_{b \in \{2,8,10\}} \hat{f}^{b}(x_2)$$

Out-of-bag Test Error Estimation

- For each $i = 1, \ldots, n$, the out-of-bag sample is:

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

- Construct the out-of-bag estimate:

$$\hat{f}^{oob}(x_i) \ = \ \frac{1}{|\tilde{B}_i|} \sum_{b \in \tilde{B}_i} \hat{f}^b(i_i)$$

- Estimate the test error as

$$\widehat{R}_{test} = \frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{f}^{oob}(x_i))$$

Out-of-bag Test Error Estimation

- We need $|\tilde{B}_i|$ to be reasonably large for all $i = 1, \ldots, n$.
- The probability π^{oob} of an observation NOT being included in a bootstrap sample $(j_1, \ldots, j_n)$ (and hence being 'out-of-bag') is:

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

- Hence $\mathbb{E}[|\tilde{B}_i|] \approx 0.367B$ for all $i = 1, \ldots, n$.
- 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.

Example: Boston Housing Dataset

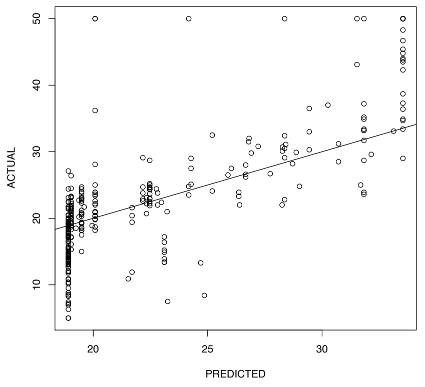
- Apply out of bag test error estimation to select optimal tree depth and assess performance of bagged trees for Boston Housing data.
- Use the entire dataset with $p = 13$ predictor variables.

```
n <- nrow(BostonHousing)    ## n samples
X <- BostonHousing[,-14]
Y <- BostonHousing[,14]
B <- 100
maxdepth <- 3
prediction_oob <- rep(0,length(Y))      ## vector with oob predictions
numbertrees_oob <- rep(0,length(Y))     ## number pf oob trees
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
```

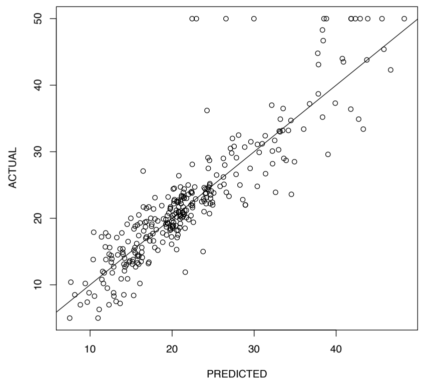
Example: Boston Housing Dataset

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

For depth $d = 1$.



For depth $d = 10$.



Example: Boston Housing Dataset

- Out-of-bag estimate of test error as a function of tree depth d:

tree depth d	1	2	3	4	5	10	30
single tree $\hat{f}$	60.7	44.8	32.8	31.2	27.7	26.5	27.3
bagged trees $\hat{f}_{Bag}$	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.
- Bagging does not always improve accuracy, but often does in practice.

Random Forests and Extremely Randomized Trees

- **Random forests** are similar to bagged decision trees with a few key differences:
  - For each split point, the search is not over all p variables but just over *mtry* randomly chosen ones (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).
  - Random forests tend to produce better predictions than bagging.
  - Results often not sensitive to the only tuning parameter *mtry*.
  - Implemented in `randomForest` library.
- Even more random methods, e.g. **extremely randomized trees**:
  - For each split point, sample *mtry* variables each with a value to split on, and pick the best one.
  - Often works even when *mtry* equals 1!
- Often produce state-of-the-art results, and top performing methods in machine learning competitions.

Breiman (2001)., Geurts et al (2006).

Random 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”.

Random Forests and Nearest Neighbours

- Let $P(x, x_i) \in [0, 1]$ be the proportion of trees for which a vector x falls into the same final leaf node as the training vector x_i. $P(x, x_i)$ is a proximity value, and tends to be large when x and x_i are close by.
- If every leaf node contains the same number of training samples, the prediction of random forests (in regression mode) at x is:

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

  which is a weighted (approximate) nearest neighbour scheme.
- If the nodes contain different number of original observations, $P(x, x_i)$ is a weighted proportion of trees, where the weight of a tree is inversely proportional to the number of samples in the leaf node containing x.
- For classification, the prediction will be the weighted majority vote, where again weights are proportional to the proximities $P(x, x_i)$.
- kNN does not scale well to very large datasets, and computational geometry techniques for approximately finding nearest neighbours often rely on tree data structures, e.g. kd-trees, cover trees, ball trees. Random forests and other randomized trees can also be thought of similarly.

Ensemble Methods

- Bagging and random forests are examples of **ensemble methods**, where predictions are based on an ensemble of many individual predictors.
- Bayesian posterior averaging can also be thought of as an ensemble method:

$$p(y|x, \text{Data}) = \int p(y|x,\theta)p(\theta|\text{Data})d\theta$$
$$\approx \frac{1}{B}\sum_{b=1}^{B} p(y|x,\theta^b)$$

  where θ^b are samples drawn from posterior $p(\theta|\text{Data})$.
- Many other ensemble learning methods: boosting, stacking, mixture of experts, Bayesian model combination etc.
- Often gives significant boost to predictive performance.

Dropout Training of Neural Networks

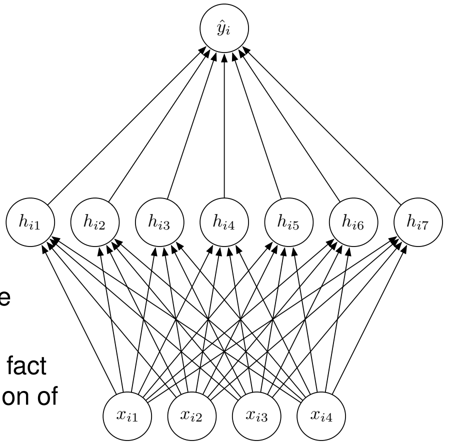
- Neural network with single layer of hidden units:
  - **Hidden unit activations**:

$$h_{ik} = s\left(b_k^h + \sum_{j=1}^{p} W_{jk}^h x_{ij}\right)$$

  - **Output probability**:

$$\hat{y}_i = s\left(b^o + \sum_{k=1}^{m} W_k^o h_{ik}\right)$$

- Large, overfitted, networks often have co-adapted hidden units.
- What each hidden unit learns may in fact be useless, e.g. predicting the negation of predictions from other units.
- Can prevent co-adaptation by randomly **dropping out** units from network.

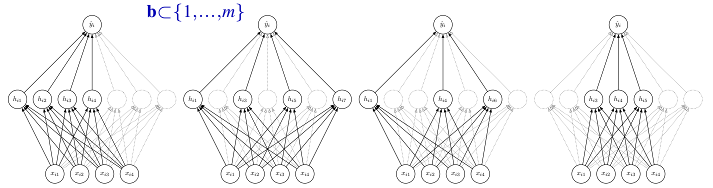


Hinton et al (2012).

Dropout Training of Neural Networks

- Model as an ensemble of networks:

$$p(y_i = 1|x_i,\theta) = \sum_{\mathbf{b}\subset\{1,\dots,m\}} q^{|\mathbf{b}|}(1-q)^{n-|\mathbf{b}|} p(y_i = 1|x_i,\theta, \text{drop out units } \mathbf{b})$$



- **Weight-sharing** among all networks: each network uses a subset of the parameters of the full network (corresponding to the retained units).
- Training by stochastic gradient descent: at each iteration a network is sampled from ensemble, and its subset of parameters are updated.

Dropout Training of Neural Networks

Classification of phonemes in speech.

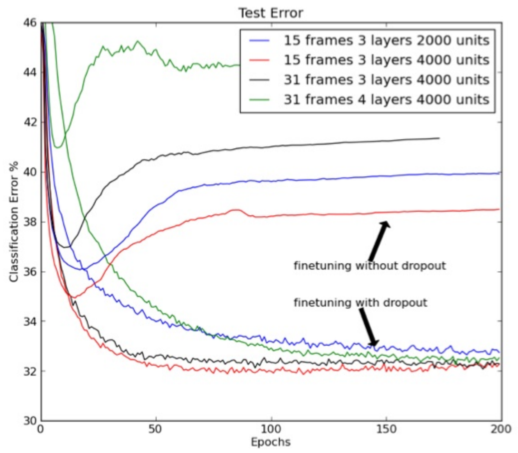


Figure from Hinton et al.

Boosting

- **Boosting** is an iterative ensemble learning technique. At iteration t, the predictor is (with $0 < \nu < 1$, typically small, say $\nu = 0.1$):

$$\hat{f}_t(x) = \sum_{m=1}^{t} \nu \hat{g}_m(x)$$

- For regression, L_2-boosting works as follows:
  1. Fit a first function to the data $(x_i, y_i)_{i=1}^n$ with **base learner**, yielding $\hat{g}_1(x)$.
  2. For $t = 2, 3, \ldots, T$ do:
     - 2.1 Compute residuals

$$u_i = y_i - \hat{f}_{t-1}(x_i)$$

     - 2.2 Fit the residuals $(x_i, u_i)_{i=1}^n$, obtaining $\hat{g}_t(x)$.
- Boosting is a bias-reduction technique, as opposed to bagging and dropout.
- Boosting works well with simple base learners, e.g. decision stumps, with low variance and high bias.
- Implemented in the `mboost` library.

Boosting

- Boosting can be viewed as functional gradient descent.
- Say we wish to minimize empirical risk with differentiable loss function,

$$R(f) = \frac{1}{n}\sum_{i=1}^{n} L(y_i - f(x_i))$$

- We can calculate $\nabla_f R(f)$ and take a gradient descent step:

$$\hat{f}_t(x) = \hat{f}_{t-1}(x) - \nu \nabla_f R(\hat{f}_{t-1})$$

- We use a base learner to approximate $\nabla_f R(\hat{f}_{t-1})$.
  1. With $\hat{f}_0 \equiv 0$, fit a first function to $\{(x_i, -\nabla_f L(y_i, \hat{f}_0(x_i)))_{i=1}^n$, yielding $\hat{g}_1(\cdot)$.
  2. Take gradient descent step:

$$\hat{f}_1(x) = \nu \hat{g}_1(x)$$

  3. For $t = 2, 3, \ldots, T$ do:
     - 3.1 Compute empirical gradient $u_i = -\nabla_f L(y_i, \hat{f}_{t-1}(x_i))$.
     - 3.2 Fit the gradient $(x_i, u_i)_{i=1}^n$, yielding $\hat{g}_t(x)$.
     - 3.3 Set

$$\hat{f}_t(x) = \hat{f}_{t-1}(x) + \nu \hat{g}_t(x)$$

Boosting

- Obtain L_2-Boosting when using the quadratic loss function

$$R(f) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2.$$

- Obtain LogitBoost when using the logistic loss function (binary classification, $y_i \in \{-1, 1\}$),

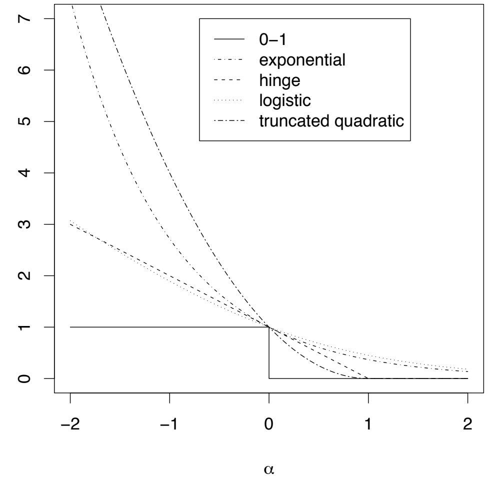
$$R(f) = \frac{1}{n}\sum_{i=1}^{n}\log(1 + \exp(-y_i f(x_i))).$$

- Obtain AdaBoost (the original boosting algorithm) when using the exponential loss (again $y_i \in \{-1, 1\}$)

$$R(f) = \frac{1}{n}\sum_{i=1}^{n}\exp(-y_i f(x_i)).$$

Freund and Schapire (1995).

Boosting



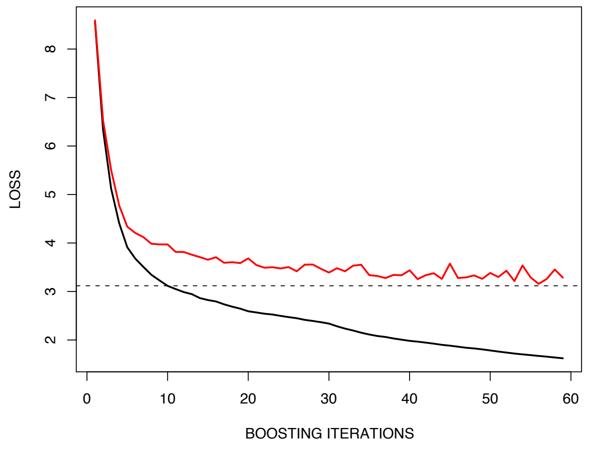
Boosting

```
library(mboost)
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 validation 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
}
```

Boosting

Plot of validation error in red and training error in black as functions of iteration.

```
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)
```



Machine Learning

- The last 8 weeks has been a whirlwind tour of basic machine learning techniques:
  - Unsupervised learning: PCA, MDS, Isomap, K-means, mixture model, EM algorithm.
  - Supervised learning: LDA, naïve Bayes, kNN, decision trees, logistic regression, SVMs, kernel methods, Gaussian processes, neural networks, deep learning.
  - Conceptual framework: prediction, generalization, overfitting and regularization.
  - Theory: decision theory, statistical learning theory, Bayesian framework.
  - Cross validation, model selection, model averaging and model combination (bagging, random forests, boosting).
- Further explorations:
  - Machine learning summer schools, videolectures.net.
  - Conferences: NIPS, ICML, UAI, AISTATS.
  - Mailing list: ml-news.