Statistical Data Mining and Machine Learning
Hilary Term 2016

Dino Sejdinovic
Department of Statistics
Oxford

Slides and other materials available at:
http://www.stats.ox.ac.uk/~sejdinov/sdmml

Decision Trees

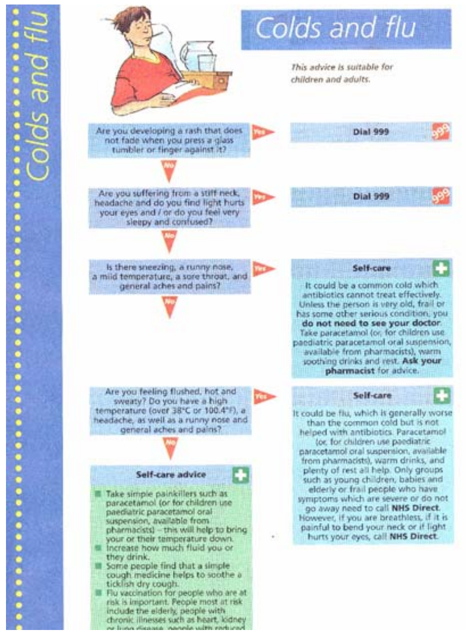
Classification and Regression Trees (**CART**)

- Denote input domain by $\mathcal{X}$ and let the output domain be $\mathcal{Y} = \{1, \ldots, K\}$ (classification) or $\mathcal{Y} = \mathbb{R}$ (regression).
- A decision tree gives a partition of $\mathcal{X}$ into R disjoint sets (regions) $\mathcal{P} = \{\mathcal{R}_1, \ldots, \mathcal{R}_R\}$, such that the fitted decision function is constant on each region $\mathcal{R}_j \subset \mathcal{X}$, $j = 1, \ldots, R$, i.e.

$$f_{\text{tree}}(x) = \beta_j, \forall x \in \mathcal{R}_j.$$

- Main strengths: easy to use, easy to interpret.
- Often serve as a starting point for powerful model combination and ensemble techniques: bagging, boosting (random forests).

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 ↓

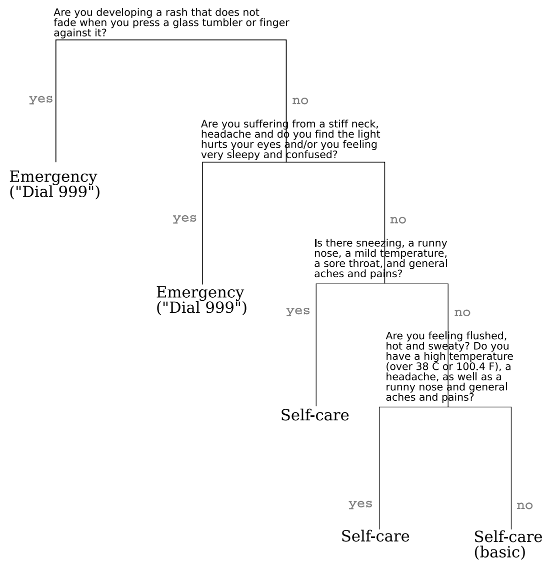
Are you suffering from a stiff neck, headache and do you find light hurts your eyes and / or do you feel very sleepy and confused?
Yes → **Dial 999**
No ↓

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 soothing drinks and rest. **Ask your pharmacist** for advice.
No ↓

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 → **Self-care**
It could be flu, which is generally worse than the common cold but is not helped with antibiotics. Paracetamol (or, for children use paediatric paracetamol oral suspension, available from pharmacists), warm drinks, and plenty of rest all help. Only groups such as young children, babies and elderly or frail people who have symptoms which are severe or do not go away need to call **NHS Direct**. However, if you are breathless, if it is painful to bend your neck or if light hurts your eyes, call **NHS Direct**.
No ↓

Self-care advice
- Take simple painkillers such as paracetamol (or for children use paediatric paracetamol oral suspension, available from pharmacists) – this will help to bring your or their temperature down.
- Increase how much fluid you or they drink.
- Some people find that a simple cough medicine helps to soothe a ticklish dry cough.
- Flu vaccination for people who are at risk is important. People most at risk include the elderly, people with chronic illnesses such as heart, kidney

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)

Decision Trees

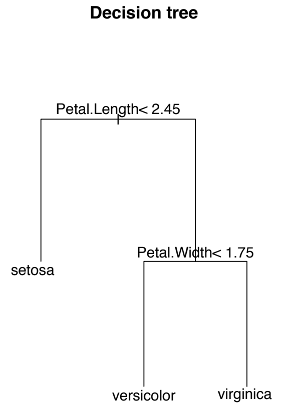
- A decision tree is a hierarchically organized structure, with each node splitting the data space into regions based on value of a single feature (attribute).
- Some terminology:
  - **Parent** of a node c is the node with an arrow pointing into c.
  - **Children** of a node c are those nodes which 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.
- Partition of $\mathcal{X}$ into R disjoint sets $(\mathcal{R}_1, \ldots, \mathcal{R}_R)$ is determined by the **leaves of the tree**.
- On each region $\mathcal{R}_j$ the same decision/prediction is made: $f_{\text{tree}}(x) = \beta_j$ for all $x \in \mathcal{R}_j$ - typically as a majority vote of the data items associated to that leaf (classification) or as their mean (regression)

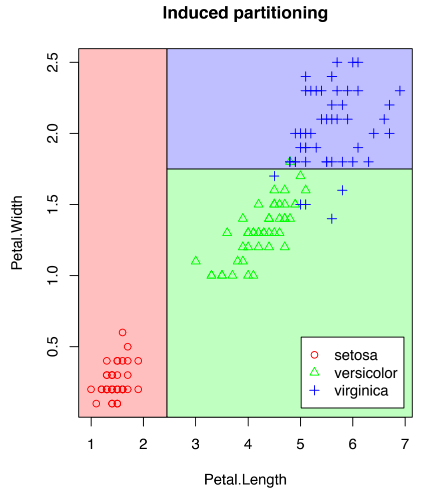
Example: Iris Data

```
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





Partition of $\mathcal{X}$ into R disjoint sets $(\mathcal{R}_1, \ldots, \mathcal{R}_R)$ is determined by the **leaves of the tree**.

Decision functions based on trees

- For regression problems, the parameterized function is

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

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

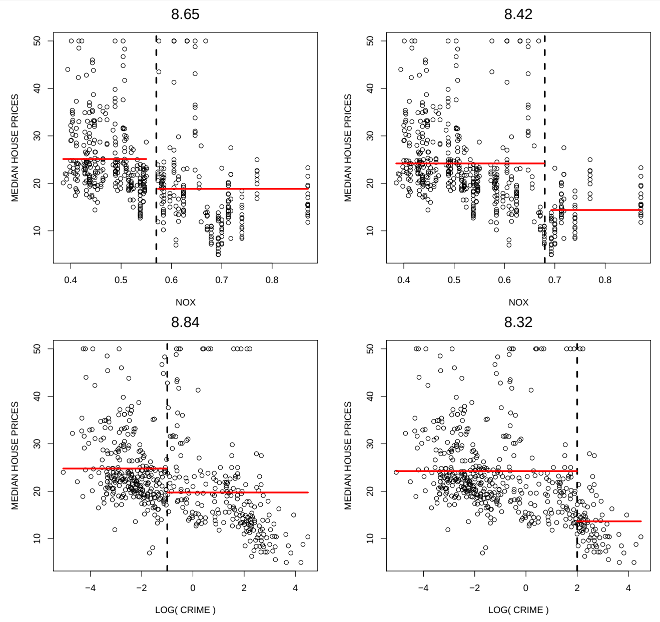
1. Start with $\mathcal{R}_1 = \mathcal{X} = \mathbb{R}^p$.
2. For each feature $j = 1, \ldots, p$, and for each value $v \in \mathbb{R}$ that we can split on:
   1. Split data set:
      $$I_< = \{i : x_i^{(j)} < v\} \qquad I_> = \{i : x_i^{(j)} \geq v\}$$
   2. Estimate parameters:
      $$\beta_< = \frac{\sum_{i \in I_<} y_i}{|I_<|} \qquad \beta_> = \frac{\sum_{i \in I_>} y_i}{|I_>|}$$
   3. Compute the **quality of split**, e.g., the square loss:
      $$\sum_{i \in I_<} (y_i - \beta_<)^2 + \sum_{i \in I_>} (y_i - \beta_>)^2$$
3. Choose split, i.e., feature j and value v, 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 sq.ft
indus	proportion of non-retail business acres per town
chas	Charles River dummy variable
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 town
lstat	percentage of lower status of the population
medv	median value of owner-occupied homes in USD 1000’s

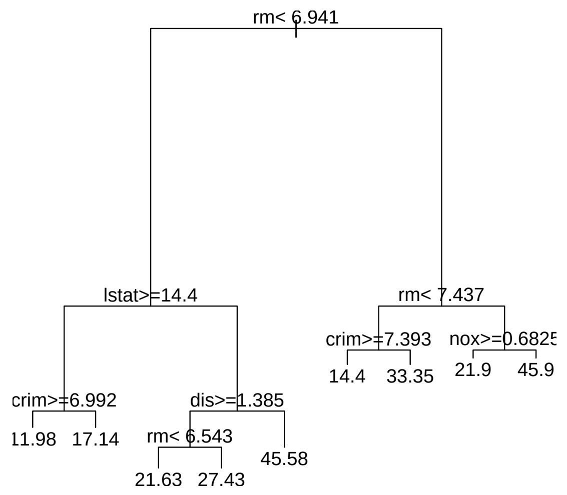
- Predict median house value.

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.



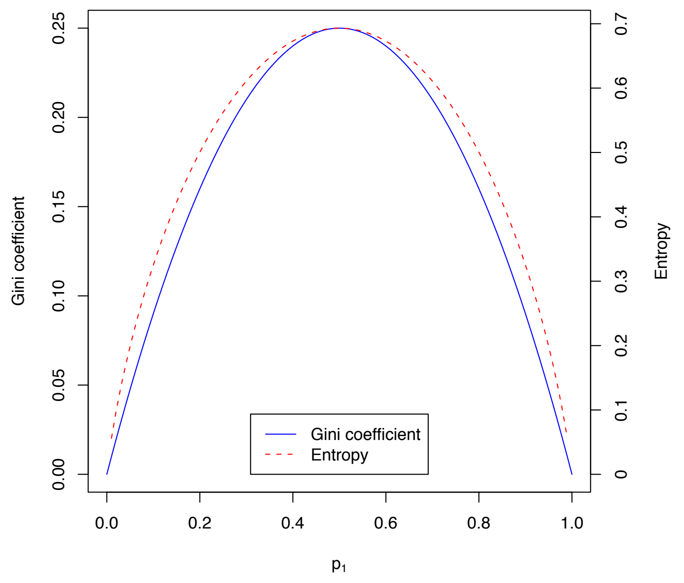
Growth Heuristics for Classification Trees

- For binary classification, the proportion of class 1 items in node corresponding to region $\mathcal{R}_j$ is given by

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

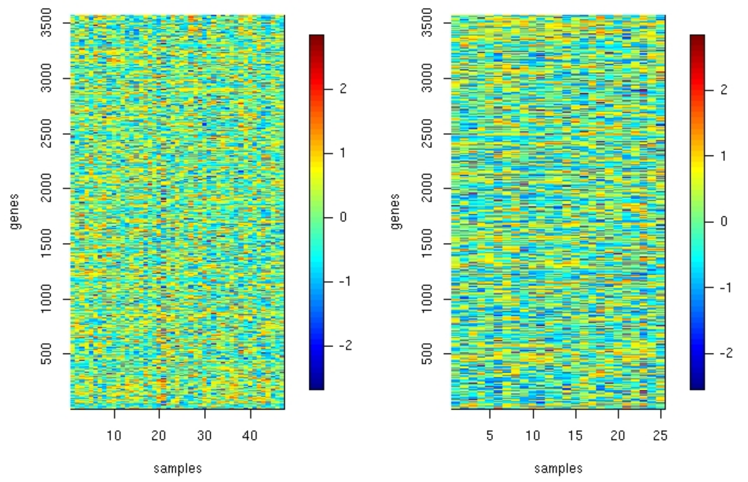
- A split is good if both sides are more **pure**, i.e. $\hat{\beta}_{j1}$ is closer to 0 or 1.
- Different measures of node impurity:
  - **Misclassification error**: $1 - \max\{\hat{\beta}_{j1}, 1 - \hat{\beta}_{j1}\}$.
  - **Gini impurity**: $2\hat{\beta}_{j1}(1 - \hat{\beta}_{j1})$.
  - **Entropy**: $-\hat{\beta}_{j1} \log \hat{\beta}_{j1} - (1 - \hat{\beta}_{j1}) \log(1 - \hat{\beta}_{j1})$.
- Gini and entropy preferred: differentiable and produce purer nodes.
- Extension to multi-class:
  - **Misclassification error**: $1 - \max_k \hat{\beta}_{jk}$.
  - **Gini impurity**: $\sum_{k=1}^{K} \hat{\beta}_{jk}(1 - \hat{\beta}_{jk})$.
  - **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?

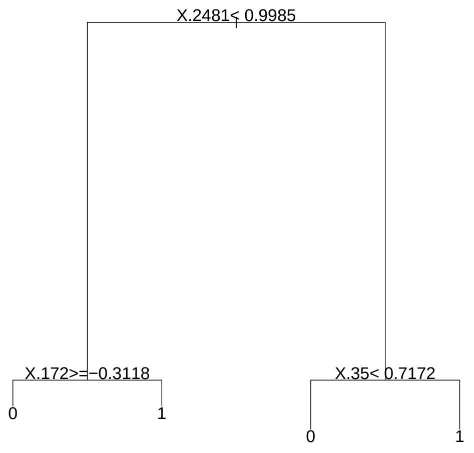
Example: Leukemia Prediction



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.



Model Complexity

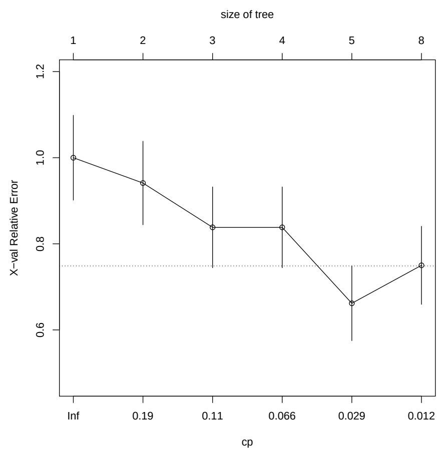
- When should tree growing be stopped?
- Will need to control complexity to prevent overfitting, and in general find optimal tree size with best predictive performance.
- A regularized objective

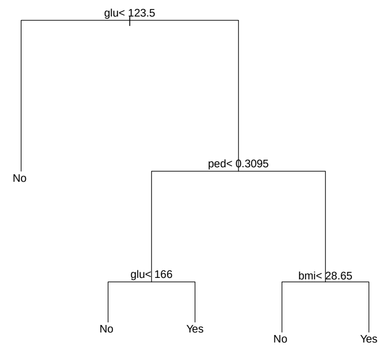
$$R^{\text{emp}}(T) + C \times \text{size}(T)$$

  - 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

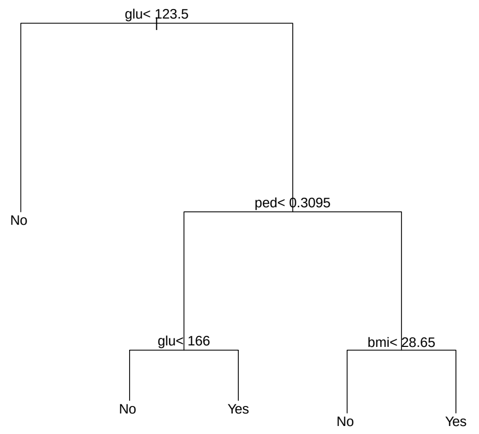
```
library(rpart); library(MASS); data(Pima.tr)
rp <- rpart(Pima.tr[,8] ~ ., data=Pima.tr[,-8],
                  control=rpart.control(xval=10)) ## 10-fold x-val
plotcp(rp) #visualise x-val results as a function of the complexity parameter
#select complexity parameter .029 since this results in lowest x-val
rp2 <- prune.rpart(rp,.029)
plot(rp2); text(rp2)
```





Bagging

Model Variability

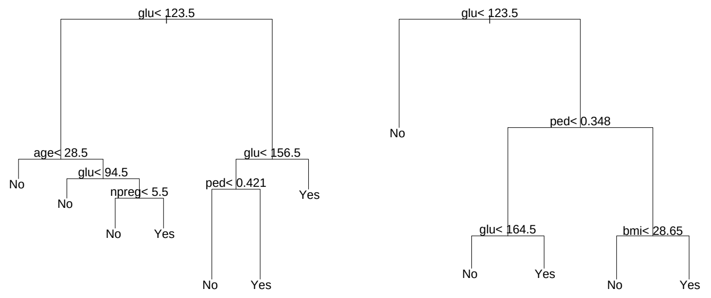


- Is the tree 'stable' if training data were slightly different?

Bootstrap for Classification Trees

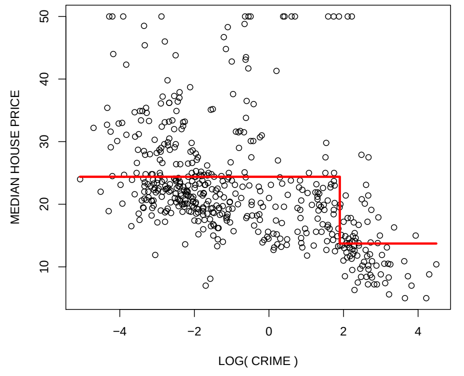
- The **bootstrap** is a way to assess the variance of estimators.
- Fit multiple trees, each on a **bootstrapped sample**. This is a data set obtained by **sampling with replacement** n times from training set.

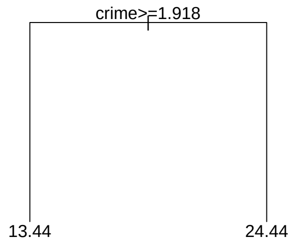
```
> n <- nrow(Pima.tr)
> bss <- sample(1:n, n , replace=TRUE)
> sort(bss)
[1]  2 4 4 5 6 7 9 10 11 12 12 12 12 13 13 15 15 20 ...

> 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.



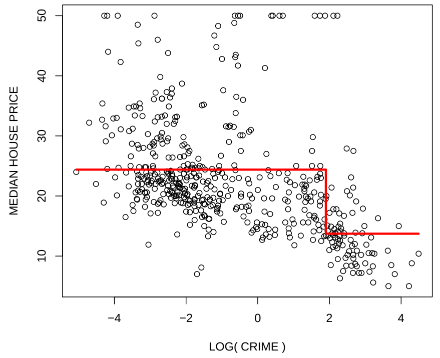


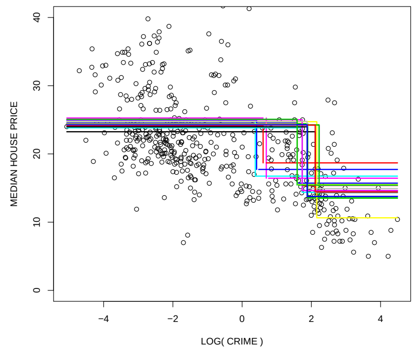
Bootstrap for Regression Trees

- 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.

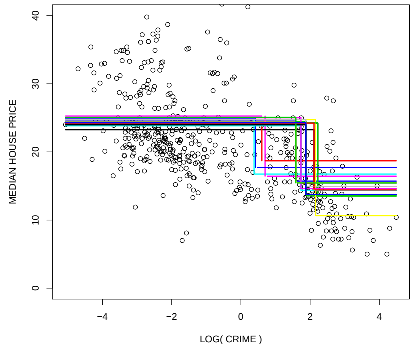
1. For $b = 1, \ldots, B$:
   1. Draw indices $(j_1, \ldots, j_n)$ from the set $\{1, \ldots, n\}$ with replacement.
   2. Fit the model, and form predictor $\hat{f}^b(x)$ based on bootstrap sample

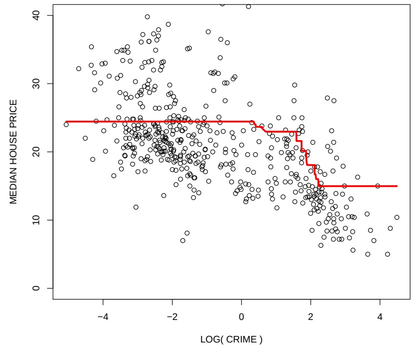
$$(x_{j_1}, y_{j_1}), \ldots, (x_{j_n}, y_{j_n})$$

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.
- Bagging reduces the variance of predictions, i.e. when taking expectations over a random dataset $\mathcal{D}$:

$$\mathbb{E}_{\mathcal{D}}\left[(\hat{f}(x) - \mathbb{E}_{\mathcal{D}}[\hat{f}(x)])^2\right] \;\geq\; \mathbb{E}_{\mathcal{D}}\left[(\hat{f}_{Bag}(x) - \mathbb{E}_{\mathcal{D}}[\hat{f}_{Bag}(x)])^2\right]$$

Variance Reduction in Bagging

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

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

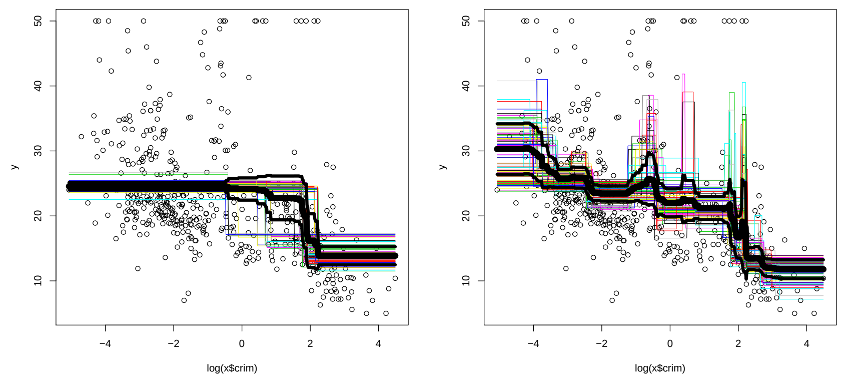
where expectation is with respect to datasets of size n.
- The squared-loss is:

$$\mathbb{E}_{\mathcal{D}}[(Y - \hat{f}_{ag}(X))^2 | X = x] = \mathbb{E}_{\mathcal{D}}[(Y - \bar{f}(X))^2 | X = x] + \mathbb{E}_{\mathcal{D}}[(\bar{f}(X) - \hat{f}_{ag}(X))^2 | X = x]$$
$$\to \mathbb{E}_{\mathcal{D}}[(Y - \bar{f}(X))^2 | X = x] \quad \text{as } B \to \infty.$$

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).

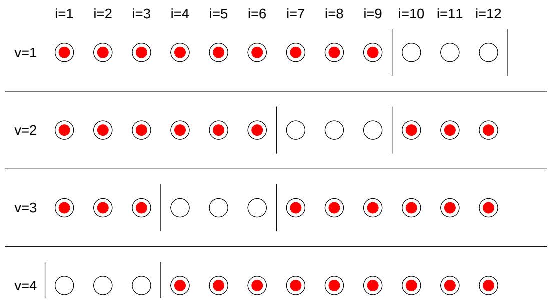
Variance Reduction in Bagging

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



Out-of-bag Test Error Estimation

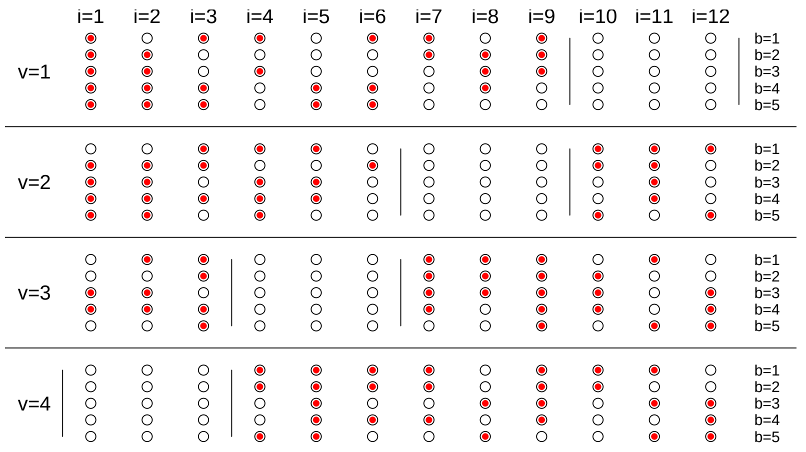
- How well does bagging to? Can we estimate generalization performance, and tune hyperparameters?
- Answer 1: cross-validation.



- For each $v = 1, \dots, V$,
  - 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

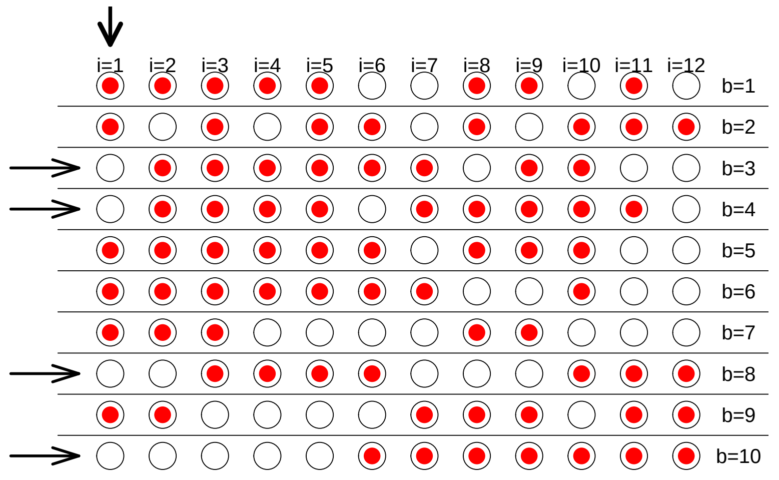
- But to fit $\hat{f}_{Bag}$ on the training set for each $v = 1, \dots, V$, we have to train on B bootstrap samples!



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

Out-of-bag Test Error Estimation

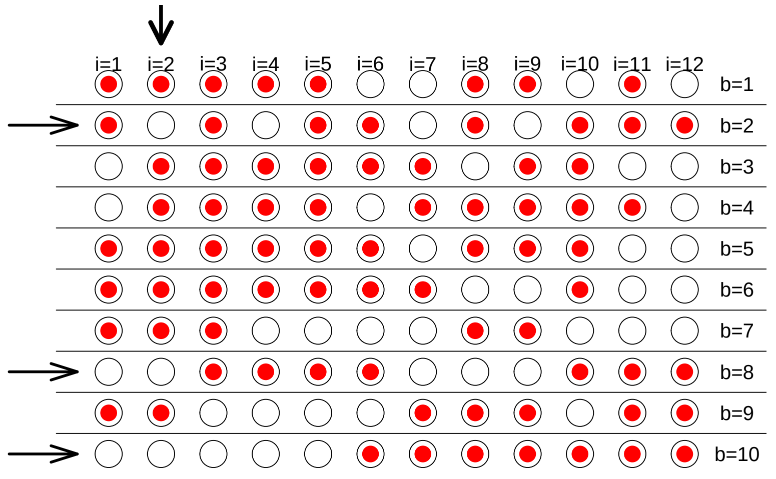
- Idea: test on the “unused” data points in each bootstrap iteration to estimate the test error.



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

Out-of-bag Test Error Estimation

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



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

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 at x_i:

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

- Out-of-bag risk:

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

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:

$$\pi^{\text{oob}} = \prod_{i=1}^{n} \left(1 - \frac{1}{n}\right) \overset{n \to \infty}{\longrightarrow} \frac{1}{e} \approx 0.367.$$

- Hence $\mathbb{E}[|\tilde{B}_i|] \approx 0.367B$
- 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

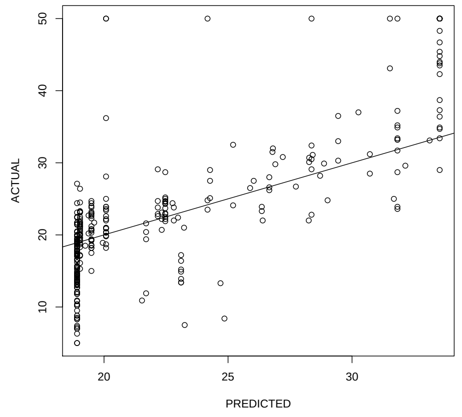
- Apply out of bag test error estimation to select optimal tree depth and assess performance of bagged trees for Boston Housing data.
- Use the entire dataset with $p = 13$ predictor variables.

```
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
```

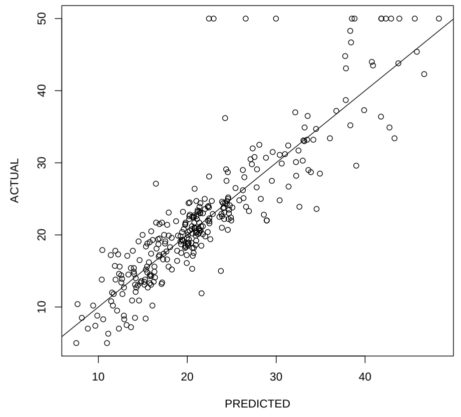
Example: Boston Housing Dataset

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

For depth $d = 1$.



For depth $d = 10$.



Example: Boston Housing Dataset

- Out-of-bag 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 $d = 30$.

Summary:

- Bagging reduces variance and prevents overfitting
- Often improves accuracy in practice.
- Bagged trees cannot be displayed as nicely as single trees and some of the interpretability of trees is lost.

Random Forests

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 random 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, 2001.

Random Forests

Comparison of 179 classifiers on 121 datasets. Random forests come top with SVMs close behind.

Rank	Acc.	κ	Classifier
32.9	82.0	63.5	parRF_t (RF)
33.1	**82.3**	**63.6**	rf_t (RF)
36.8	81.8	62.2	svm_C (SVM)
38.0	81.2	60.1	svmPoly_t (SVM)
39.4	81.9	62.5	rforest_R (RF)
39.6	82.0	62.0	elm_kernel_m (NNET)
40.3	81.4	61.1	svmRadialCost_t (SVM)
42.5	81.0	60.0	svmRadial_t (SVM)
42.9	80.6	61.0	C5.0_t (BST)
44.1	79.4	60.5	avNNet_t (NNET)

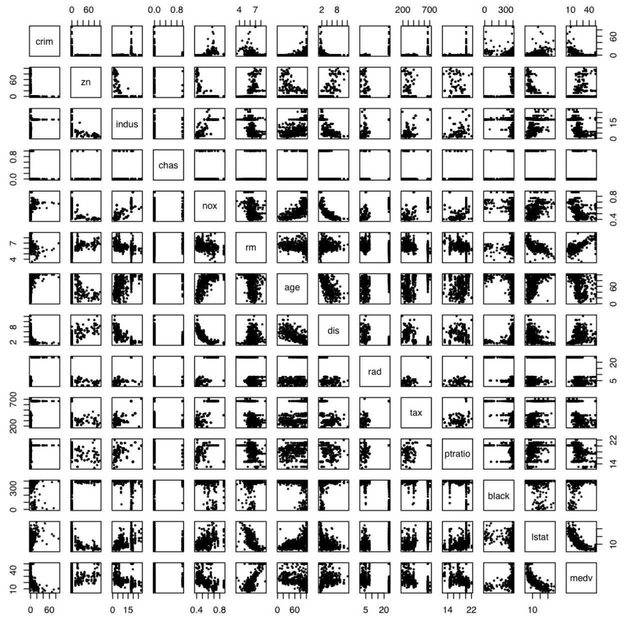
From Delgado et al, 2014

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=FALS
                 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

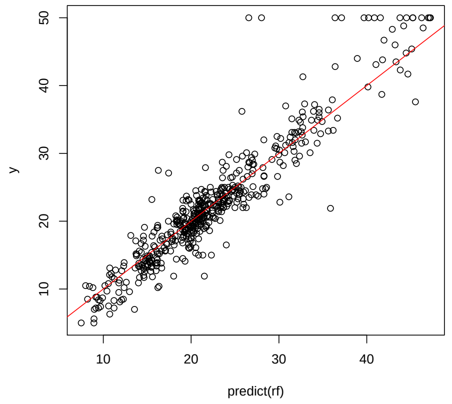
```
> plot( predict(rf,newdata=x), y)
```

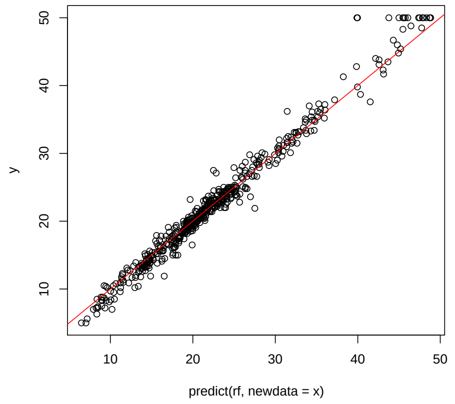
Out-of-bag error.

```
> 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
```

`mtry` is the only real tuning parameter and typically performance not sensitive to its choice (can use `tuneRF` to select it automatically).

Variable "Importance"

- Tree ensembles have better performance, but decision trees are more interpretable.
- How to interpret a forest of trees ?

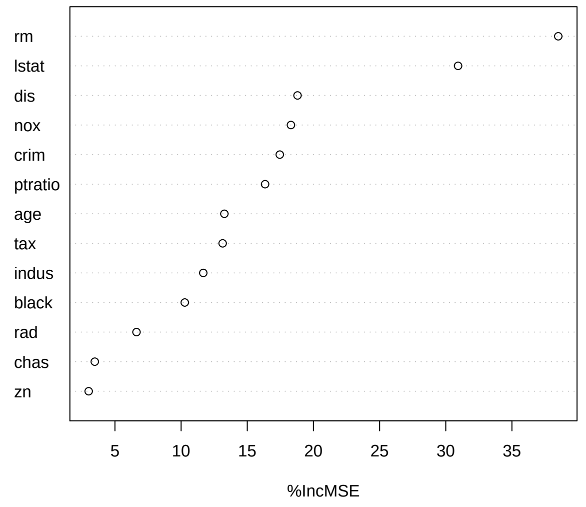
Idea: denote by $\hat{e}$ the out-of bag estimate of the loss when using the original data samples. For each variable $k \in \{1, \ldots, p\}$,

- permute randomly the k-th predictor variable to generate a new set of samples $(\tilde{X}_1, Y_1), \ldots, (\tilde{X}_n, Y_n)$, i.e., $\tilde{X}_i^{(k)} = X_{\tau(i)}^{(k)}$, for a permutation τ.
- 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 a random permutation of the k-th variable.

Example for Boston Housing data.

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

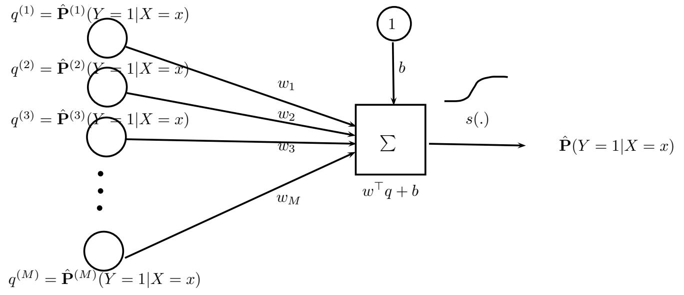


Ensemble Methods

- Bagging and random forests are examples of **ensemble methods**, where predictions are based on an ensemble of many individual predictors.
- Many other ensemble learning methods: boosting, stacking, mixture of experts, Bayesian model combination, Bayesian model averaging etc.
- Often gives significant boost to predictive performance.

Stacking

- Also called **stacked generalization**.
- Use the outputs of M learning algorithms as inputs to a **combiner learner**.
- Often, logistic regression is used as a combiner.


$q^{(1)} = \hat{\mathbf{P}}^{(1)}(Y = 1|X = x)$
$q^{(2)} = \hat{\mathbf{P}}^{(2)}(Y = 1|X = x)$
$q^{(3)} = \hat{\mathbf{P}}^{(3)}(Y = 1|X = x)$
$q^{(M)} = \hat{\mathbf{P}}^{(M)}(Y = 1|X = x)$
1
b
w_1
w_2
w_3
w_M
Σ
$w^\top q + b$
$s(.)$
$\hat{\mathbf{P}}(Y = 1|X = x)$

Top entries for the $1M Netflix competition used a form of stacking Sill et al, 2009

Boosting

Boosting

- Greedy ensemble learning algorithm, results in a weighted average of **weak learners** $f_T(x) = \sum_{t=1}^{T} \beta_t \phi(x; \theta_t)$ (usually, weak learners $\phi(x; \theta_t)$ are very simple models with low variance and high bias, e.g. decision stumps).
- Can be understood as **forward stagewise additive modelling**:
  - Initialise $f_0(x) = 0$.
  - At iteration $t = 1, \ldots, T$, add a new weighted weak learner into the model:

$$\begin{aligned}(\beta_t, \theta_t) &= \underset{\beta,\theta}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(y_i, f_{t-1}(x_i) + \beta\phi(x_i; \theta)\right) \\ f_t(x) &= f_{t-1}(x) + \beta_t \phi(x; \theta_t)\end{aligned}$$

  In practice, update $f_t(x) = f_{t-1}(x) + \nu\beta_t\phi(x; \theta_t)$ used instead with $\nu \in (0, 1)$, typically $\nu = 0.1$ (**shrinkage**).
- For trees, θ parametrises the split variables and split points at the internal nodes.

Types of Boosting

How to solve the subproblem of new weak learner addition depends on the loss function and is typically independent of the form of weak learners.

- L_2**-Boosting**: the squared loss function (regression, $y_i \in \mathbb{R}$)

$$L\left(y_i, f(x_i)\right) = \left(y_i - f(x_i)\right)^2,$$

- **LogitBoost**: logistic loss function (binary classification, $y_i \in \{-1, 1\}$)

$$L\left(y_i, f(x_i)\right) = \log(1 + \exp(-y_i f(x_i))),$$

- **AdaBoost**: exponential loss function (binary classification, $y_i \in \{-1, 1\}$)

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

Freund and Schapire (1995).

L_2-Boosting

In L_2-Boosting, new weak learners are obtained by fitting the residuals:

$$L\left(y_i, f_{t-1}(x_i) + \beta\phi(x_i;\theta)\right) = \left(y_i - f_{t-1}(x_i) - \beta\phi(x_i;\theta)\right)^2 = L(\underbrace{y_i - f_{t-1}(x_i)}_{r_{i,t}}, \beta\phi(x_i;\theta))$$

- Initialise $f_0(x) = 0$.
- For $t = 1, \ldots, T$, compute current residuals

$$r_{i,t} = y_i - f_{t-1}(x_i),$$

and fit the residuals $\{(x_i, r_{i,t})\}_{i=1}^n$ to obtain the term $\beta_t\phi(x;\theta_t)$ to be added to the expansion.

AdaBoost

Has an interpretation as reweighting the examples at each iteration based on the loss so far:

$$n\hat{R}_t = \sum_{i=1}^n \exp\left(-y_i(f_{t-1}(x_i) + \beta\phi(x_i;\theta))\right) = \sum_{i=1}^n w_{i,t} \exp\left(-y_i\beta\phi(x_i;\theta)\right),$$

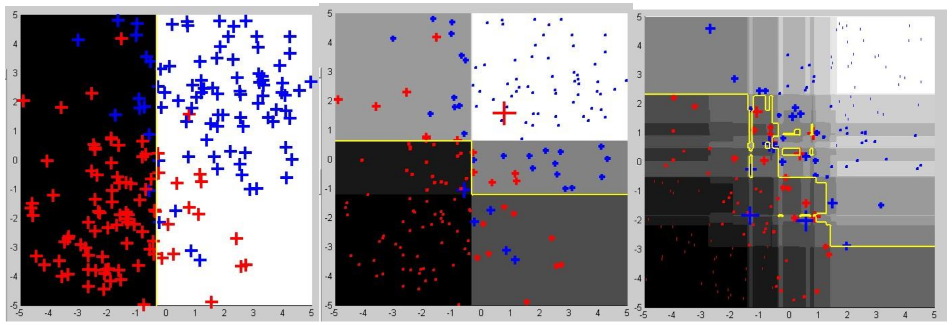
where $w_{i,t} = \exp\left(-y_i f_{t-1}(x_i)\right)$ is the weight applied to example i.

- Assume $\phi(x;\theta) \in \{-1, +1\}$ (individual classification rules) and denote $\phi_i = \phi(x_i;\theta)$. Then,

$$n\hat{R}_t = e^{\beta} \sum_{y_i \neq \phi_i} w_{i,t} + e^{-\beta} \sum_{y_i = \phi_i} w_{i,t} = (e^{\beta} - e^{-\beta}) \sum_{i=1}^n w_{i,t}\mathbf{1}\left(y_i \neq \phi_i\right) + e^{-\beta}\sum_{i=1}^n w_{i,t}$$

- Then, the solution at iteration t is given by:
  (i) $\theta_t = \text{argmin}_\theta \sum_{i=1}^n w_{i,t}\mathbf{1}\left(y_i \neq \phi(x_i,\theta)\right)$
  (ii) $\beta_t = \frac{1}{2}\log\frac{1-\text{err}_t}{\text{err}_t}$, where $\text{err}_t = \sum_{i=1}^n w_{i,t}\mathbf{1}\left(y_i \neq \phi(x_i,\theta_t)\right) / \sum_{i=1}^n w_{i,t}$

AdaBoost with decision stumps



The degree of blackness represents the confidence in the red class. The size of datapoints represents their weight. Decision boundary in yellow.
Left: After 1 iteration, Middle: After 3 iterations, Right: After 120 iterations.

Example from Murphy, p.560; generating script written by R.Stapenhurst

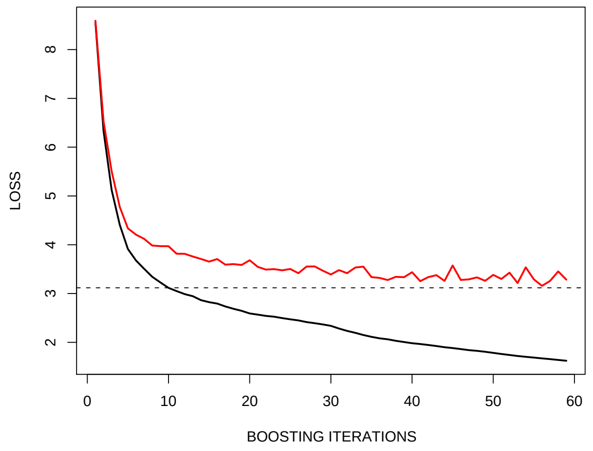
`blackboost`: Boosting of Regression Trees

```
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
}
```

blackboost: Boosting of Regression Trees

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)
```



Boosting: Summary

- Boosting is a **bias-reduction technique**, as opposed to bagging.
- Resistant to overfitting (the testing error typically stays flat for a large number of iterations - but will eventually go up).
- Can be understood as **functional gradient descent**, leading to a generic framework called **gradient boosting**.
- Relevant libraries: `mboost`, `gbm`, `xgboost`.

Further reading: Hastie et al, Chapter 10; Murphy, Section 16.4.