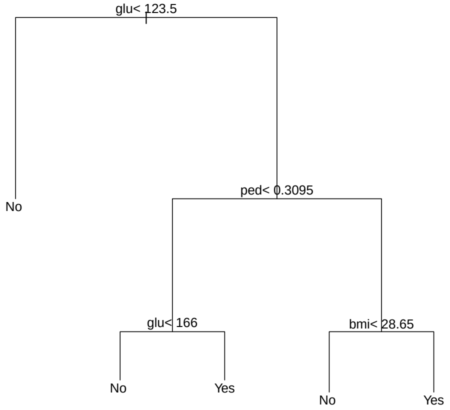
Statistical Machine Learning

Pier Francesco Palamara
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
http://www.stats.ox.ac.uk/~palamara/SML20_BDI.html

Bagging

Model Variability

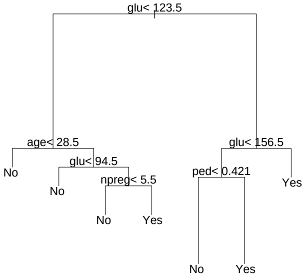


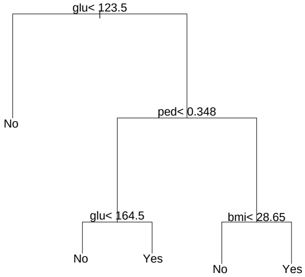
- Is the tree ‘stable’ if training data were slightly different?

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.

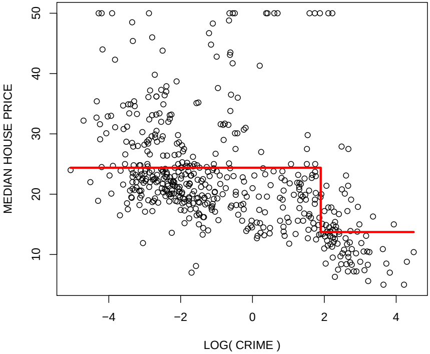
```
> n <- nrow(Pima.tr)
> bss <- sample(1:n, n , replace=TRUE)
> sort(bss)
[1]  2 4 4 5 6 7 9 10 11 12 12 12 12 13 13 15 15 20 ...

> 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.



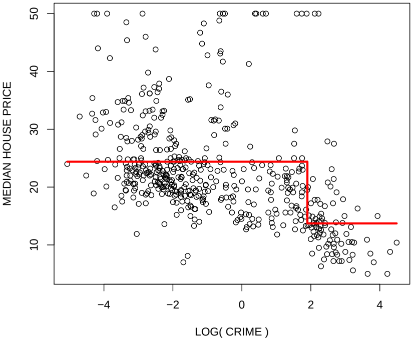
crime>= 1.918

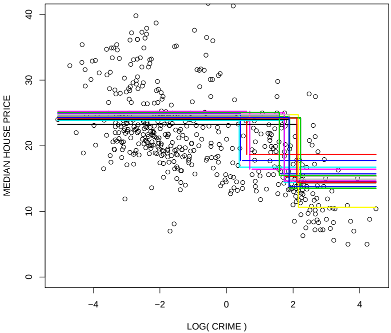
Bootstrap for Regression Trees

- 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.

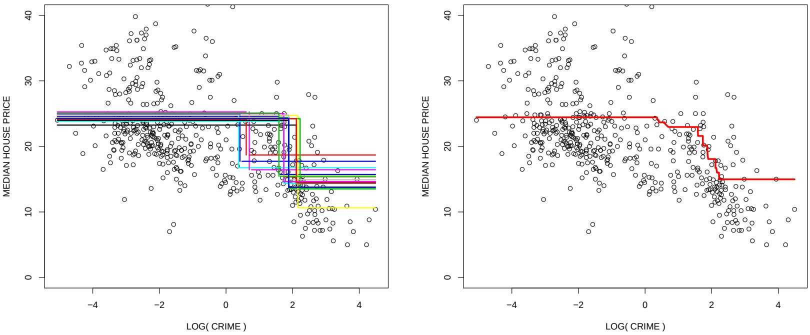
1. For $b = 1, \ldots, B$:
    1. Draw indices $(j_1, \ldots, j_n)$ from the set $\{1, \ldots, n\}$ with replacement.
    2. Fit the model, and form predictor $\hat{f}^b(x)$ based on bootstrap sample

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

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

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

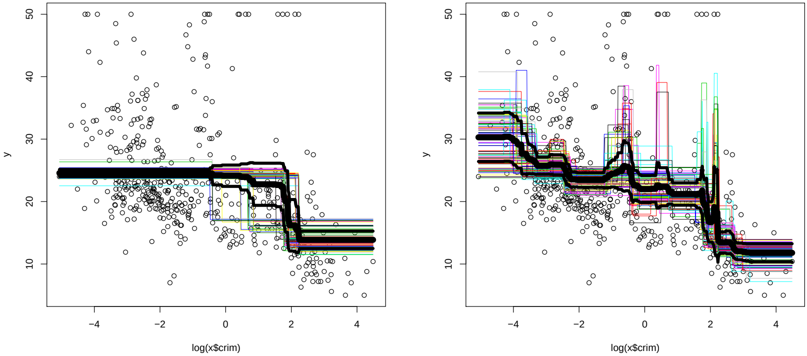
  where expectation is with respect to datasets of size n.
- The squared-loss is:

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

  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).

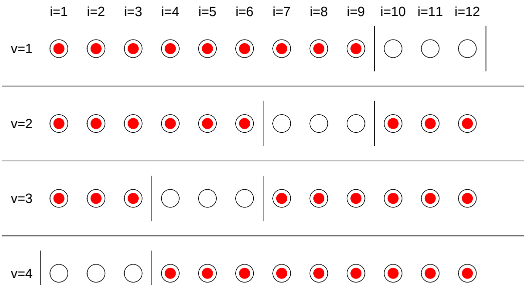
Variance Reduction in Bagging

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



Out-of-bag Test Error Estimation

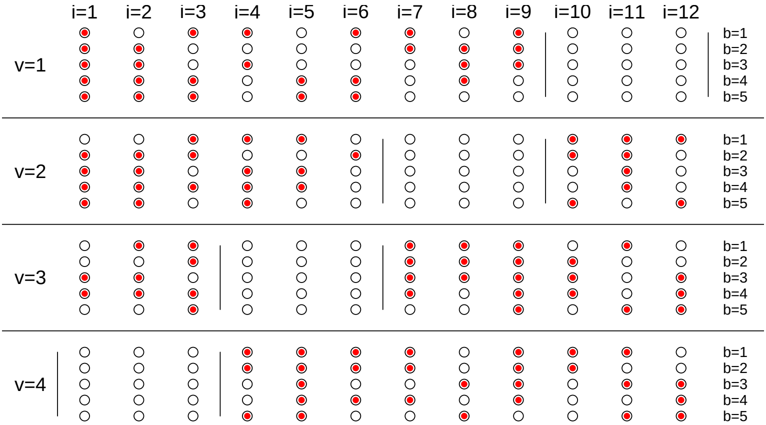
- How well does bagging to? Can we estimate generalization performance, and tune hyperparameters?
- Answer 1: cross-validation.



- 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 have to train on B bootstrap samples!

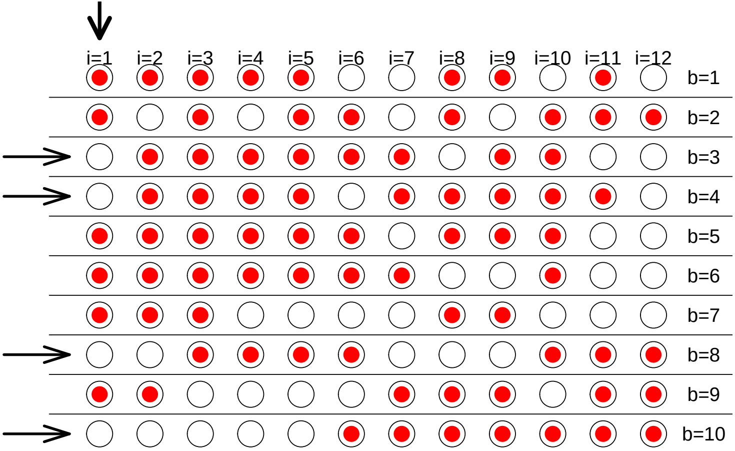


	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8	i=9	i=10	i=11	i=12	
v=1	●	○	●	●	○	●	●	○	●	○	○	○	b=1
	●	●	○	○	○	○	●	●	●	○	○	○	b=2
	●	●	○	●	○	○	○	●	●	○	○	○	b=3
	●	●	●	○	●	●	○	●	○	○	○	○	b=4
	●	●	●	○	●	●	○	○	○	○	○	○	b=5
v=2	○	○	●	●	●	○	○	○	○	●	●	●	b=1
	●	●	●	○	○	●	○	○	○	●	●	○	b=2
	●	●	○	●	●	○	○	○	○	○	●	○	b=3
	●	●	●	●	●	○	○	○	○	○	●	○	b=4
	●	●	○	●	○	○	○	○	○	●	○	●	b=5
v=3	○	●	●	○	○	○	●	●	●	○	●	○	b=1
	○	○	●	○	○	○	●	●	●	●	○	○	b=2
	●	●	○	○	○	○	●	●	●	●	○	●	b=3
	●	●	●	○	○	○	●	○	●	●	○	●	b=4
	○	○	●	○	○	○	○	○	●	○	●	●	b=5
v=4	○	○	○	●	●	●	●	○	●	●	●	○	b=1
	○	○	○	●	●	●	●	○	●	●	○	○	b=2
	○	○	○	○	●	○	○	●	●	○	●	●	b=3
	○	○	○	○	●	●	●	○	●	○	○	●	b=4
	○	○	○	●	●	○	○	●	○	○	●	●	b=5

- 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.

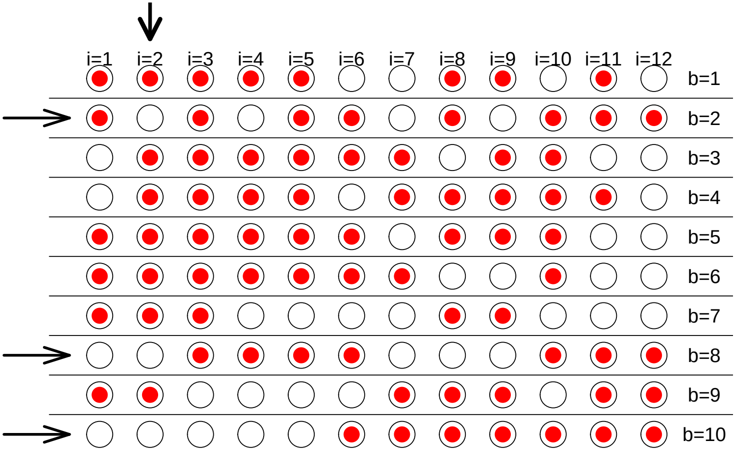


i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8	i=9	i=10	i=11	i=12	
●	●	●	●	●	○	○	●	●	○	●	○	b=1
●	○	●	○	●	●	○	●	○	●	●	●	b=2
○	●	●	●	●	●	●	○	●	●	○	○	b=3
○	●	●	●	●	○	●	●	●	●	●	○	b=4
●	●	●	●	●	●	○	●	●	●	○	○	b=5
●	●	●	●	●	●	●	○	○	●	○	○	b=6
●	●	●	○	○	○	○	●	●	○	○	○	b=7
○	○	●	●	●	●	○	○	○	●	●	●	b=8
●	●	○	○	○	○	●	●	●	○	●	●	b=9
○	○	○	○	○	●	●	●	●	●	●	●	b=10

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

Out-of-bag Test Error Estimation

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



i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8	i=9	i=10	i=11	i=12	
●	●	●	●	●	○	○	●	●	○	●	○	b=1
●	○	●	○	●	●	○	●	○	●	●	●	b=2
○	●	●	●	●	●	●	○	●	●	○	○	b=3
○	●	●	●	●	○	●	●	●	●	●	○	b=4
●	●	●	●	●	●	○	●	●	●	○	○	b=5
●	●	●	●	●	●	●	○	○	●	○	○	b=6
●	●	●	○	○	○	○	●	●	○	○	○	b=7
○	○	●	●	●	●	○	○	○	●	●	●	b=8
●	●	○	○	○	○	●	●	●	○	●	●	b=9
○	○	○	○	○	●	●	●	●	●	●	●	b=10

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

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 $i = 1, \dots, n$.
- 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:

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

- 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

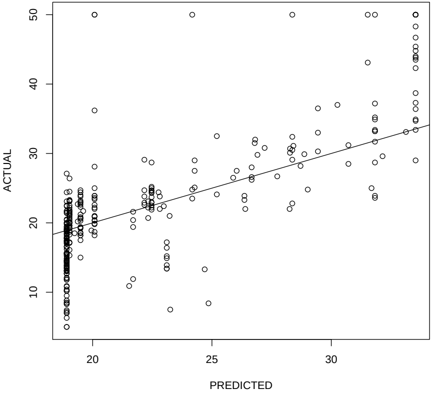
- Apply out of bag test error estimation to select optimal tree depth and assess performance of bagged trees for Boston Housing data.
- Use the entire dataset with $p = 13$ predictor variables.

```
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
```

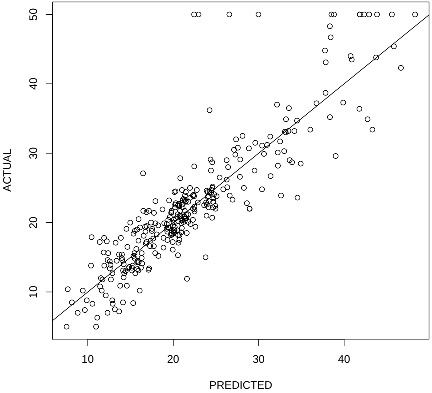
Example: Boston Housing Dataset

```
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

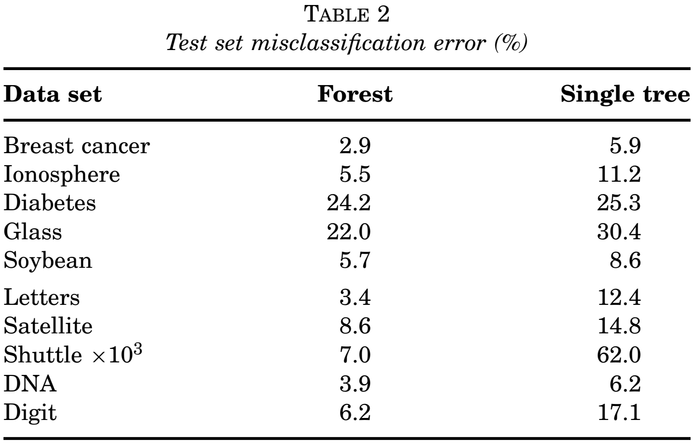


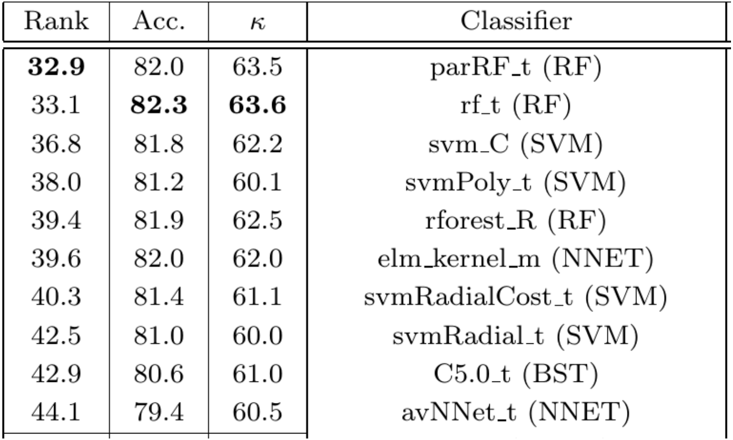
TABLE 2
Test set misclassification error (%)

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)

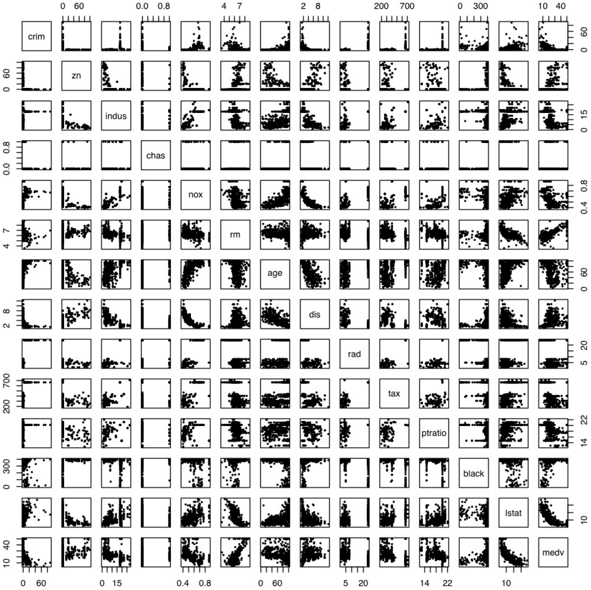
From Delgado et al, 2014

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

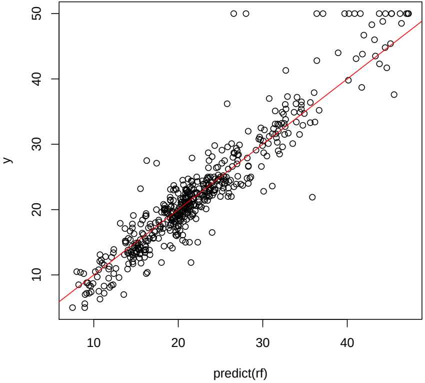
```
> plot( predict(rf), y)
> abline(c(0,1),col=2)
```

Same if treating the training data as new data

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

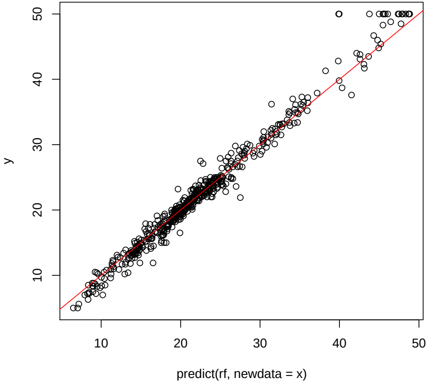
Out-of-bag error.

```
> 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 ?

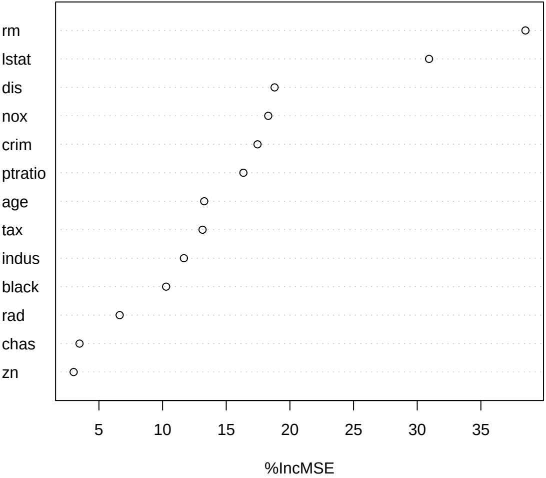
Idea: denote by $\hat{e}$ the out-of bag estimate of the loss when using the original data samples. For each variable $k \in \{1, \dots, p\}$,

- permute randomly the k-th predictor variable to generate a new set of samples $(\tilde{X}_1, Y_1), \dots, (\tilde{X}_n, Y_n)$, 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.

Microsoft Kinect Pose Recognition

Kinect algorithm CVPR Paper

Video