HT2015: SC4
Statistical Data Mining and Machine Learning

Dino Sejdinovic
Department of Statistics
Oxford

http://www.stats.ox.ac.uk/~sejdinov/sdmml.html

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
```



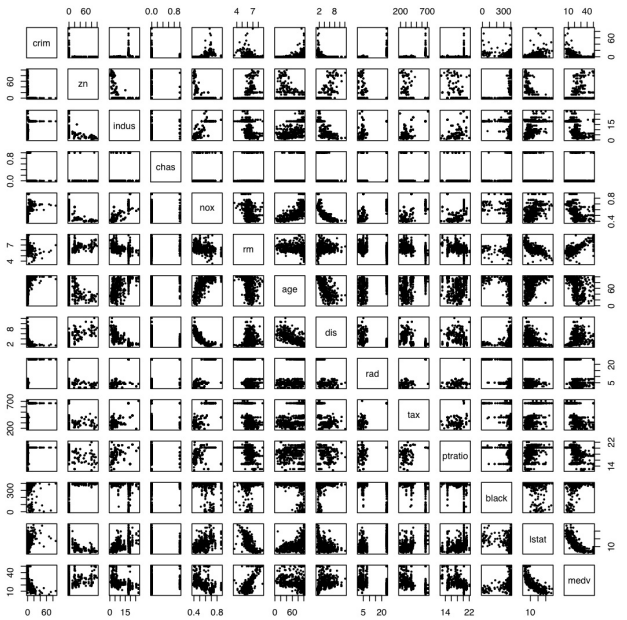
```
> randomForest          package:randomForest          R Documentation

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

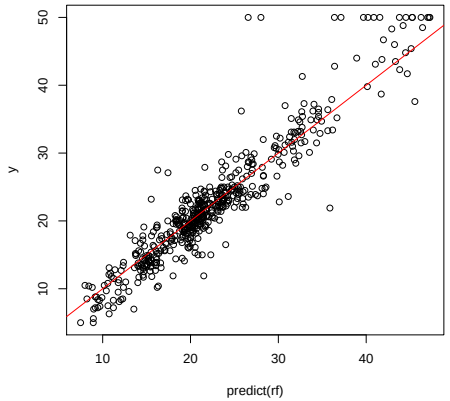
```
> plot( predict(rf), y)
> abline(c(0,1),col=2)
```

Same if treating the training data as new data

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

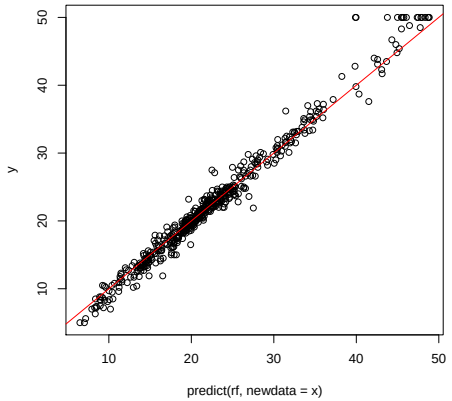

Out-of-bag error.

```
> 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"

- Despite the better predictive performance, single trees seem to have an edge over tree ensembles in terms of interpretability.
- How to interpret a forest of trees ?

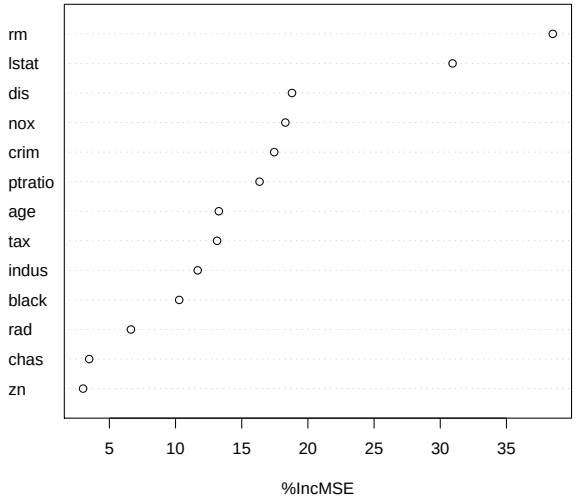
Idea: denote by $\hat{e}$ the out-of bag estimate of the loss when using the original data samples. For each variable $k \in \{1, \ldots, p\}$,

- 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)
```



Random Forests and Local Smoothing

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

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

  which is a local smoothing estimate.
- If the nodes contain different number of original observations, $\mathfrak{P}(x, x_i)$ 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 $\mathfrak{P}(x, x_i)$.
- Nearest neighbours and local smoothing techniques do not scale to very large datasets, and approximate techniques for these often rely on tree data structures, e.g. kd-trees, cover trees, ball trees. Random forests and other randomized trees can be interpreted in this way.

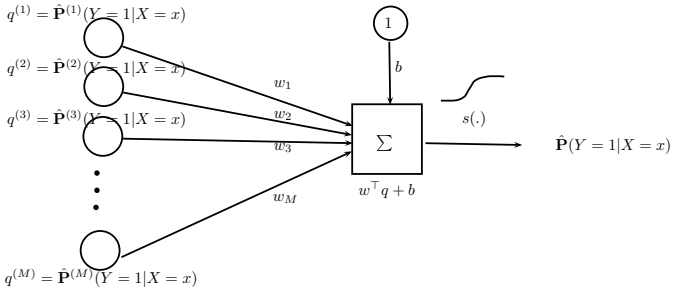
Ensemble Methods

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.



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

Dropout Training of Neural Networks

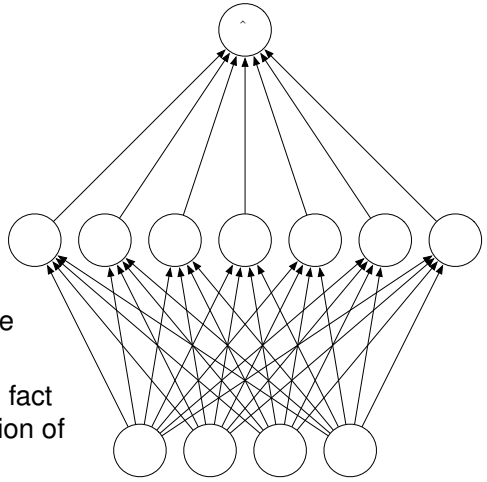
- Neural network with single layer of hidden units:
  - **Hidden unit activations**:

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

  - **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.

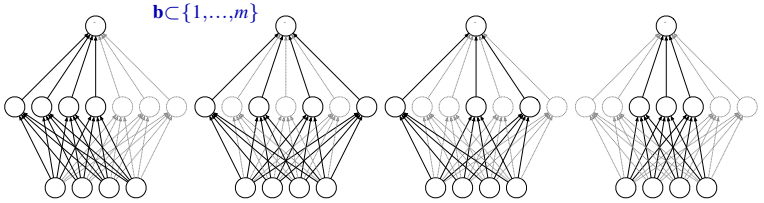


Hinton et al (2012).

Dropout Training of Neural Networks

- Model as an ensemble of networks:

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



- **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.

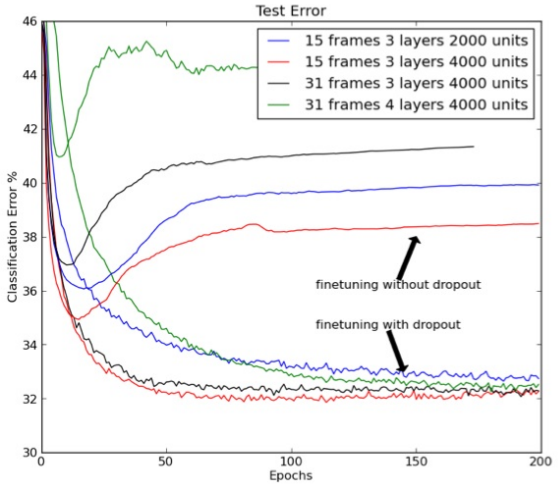


Figure from Hinton et al.

Boosting

Boosting

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

$$\hat{f}_t(x) = \sum_{\ell=1}^{t} \nu \hat{f}_\ell(x)$$

- 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 $\hat{f}_1$.
  2. For $t = 2, 3, \ldots, T$ do:
     1. Compute current residuals
     $$u_i = y_i - \hat{f}_{t-1}(x_i)$$
     2. Fit the residuals $\{(x_i, u_i)\}_{i=1}^n$, obtaining $\hat{f}_t(x)$.
- Boosting is a **bias-reduction technique**, as opposed to bagging and dropout.
- Boosting works well with simple base learners with low variance and high bias, e.g. decision stumps.
- Implemented in the `mboost` library.

Types of Boosting

- L_2**-Boosting**: the squared loss function in regression.

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

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

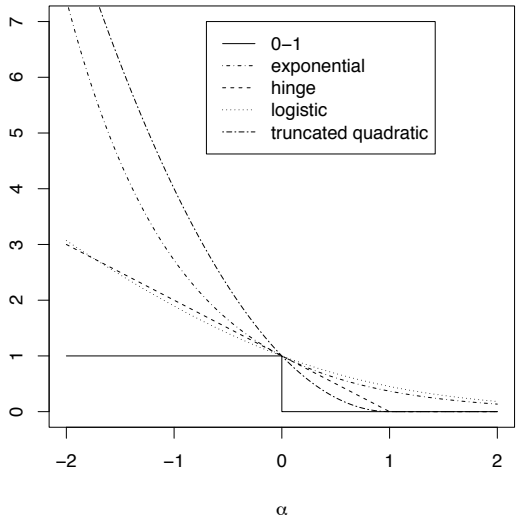
$$R(f) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-y_i f(x_i))).$$

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

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

Freund and Schapire (1995).

Boosting



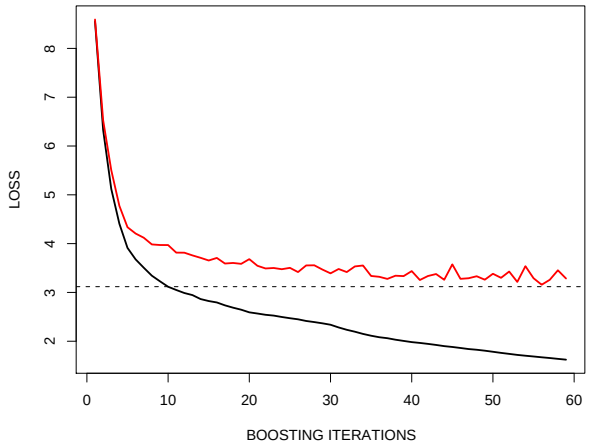
blackboost: Boosting of Regression Trees

```
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)
```



RF & Boosting: Summary

Both RF and Boosting are tree ensembles.

- Like RF, Boosting does not seem to overfit (the CV curve stays flat). This is not quite true, though: consider $\lim_{t\to\infty} \hat{f}_t(X_i)$. Needs early stopping!
- The stopping parameter T 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 usually 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.