Random Forests

Stat 557
Heike Hofmann

Outline

- Growing Random Forests
- Parameters
- Results
- (Neural Networks)

Random Forests

- Breiman (2001), Breiman & Cutler (2004)
- Tree Ensemble built by randomly sampling cases and variables
- Each case classified once for each tree in the ensemble

How do Random Forests work

- Large number (at least 500) of 'different' trees is grown
- Each tree gives a classification for each record, i.e. the tree "votes" for that class.
- The forest determines the overall classification for each record by a majority vote.

Growing a Random Forest

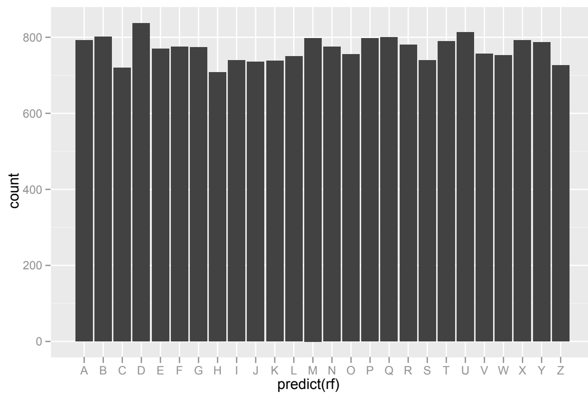
for sample size N and M explanatory variables $X_1, ..., X_M$

- draw bootstrap sample of data (i.e. draw sample of size N with replacement)
- at each node, select m << M variables at random and find best split.
- each tree is grown to the largest extent possible, i.e. no pruning!

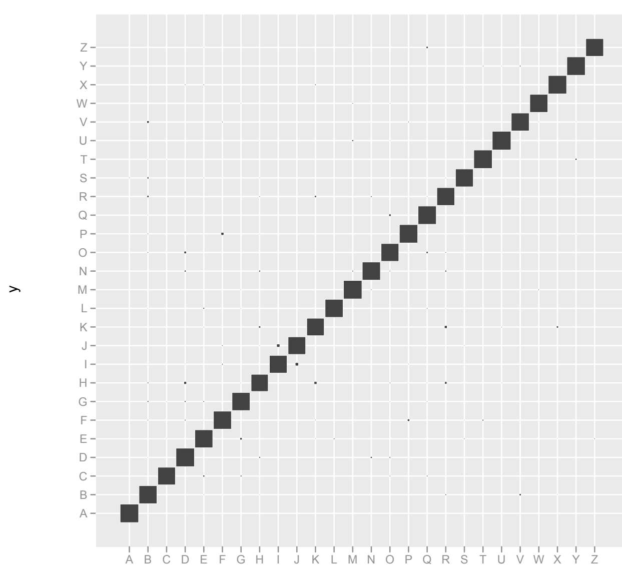
randomForest package

```
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,
             maxnodes = NULL,
             importance=FALSE, localImp=FALSE, nPerm=1,
             proximity, oob.prox=proximity,
             norm.votes=TRUE, do.trace=FALSE,
             keep.forest=!is.null(y) && is.null(xtest), corr.bias=FALSE,
             keep.inbag=FALSE, ...)
```

Results



Misclassification



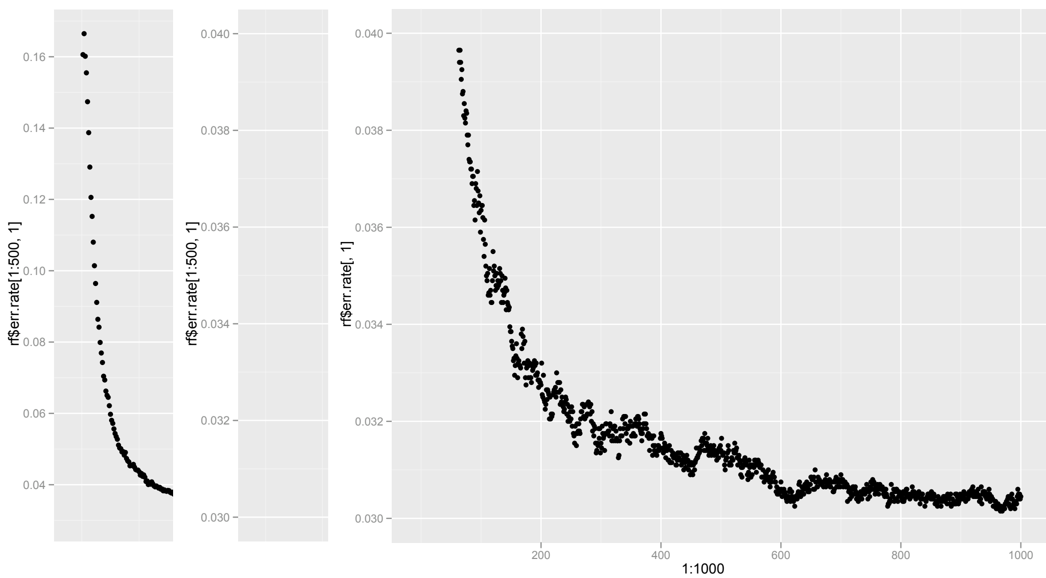
Forest Error

- Increasing correlation between any two trees increases the forest error rate.
- Trees with low individual error rates are stronger classifiers. Increasing strength of individual trees decreases the overall forest error rate.
- decreased m reduces both correlation and strength. "optimal" range of m usually quite wide.

Out of bag (oob) error

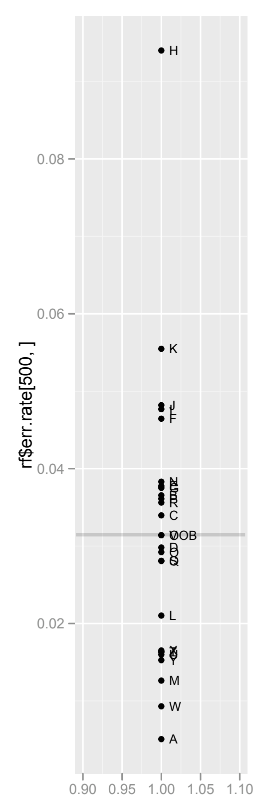
- Slight modification to bootstrap samples:
- for each tree, leave about 1/3 of data out of sample, then draw bootstrap sample of size N.
- use out-of bag data to get (running) unbiased estimate of classification error as each tree is added to forest.

running oob error-rate



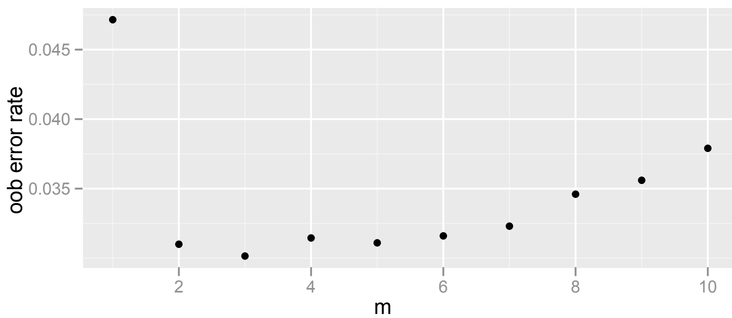
Class errors

- oob classification allows to assess error rates for each class



optimal choice of m

- based on oob error:



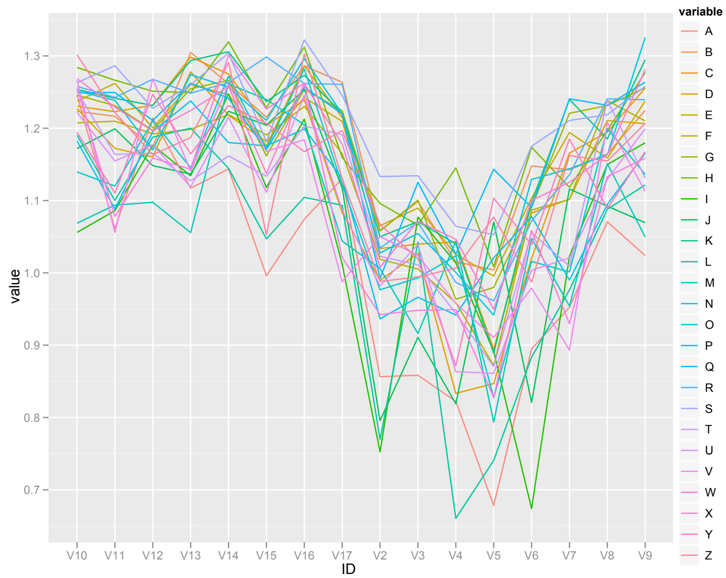
sqrt(M) works well in most cases

Variable Importance

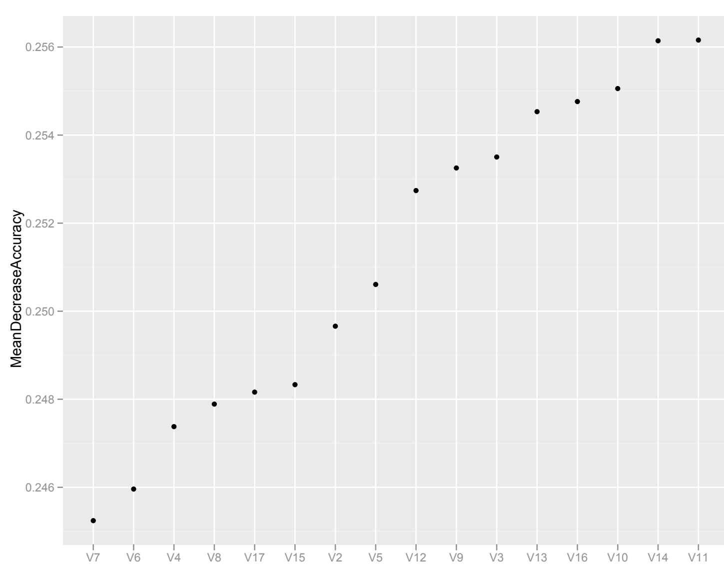
Permutation Criterion:

- based on out-of bag data
- for each tree, count # of correctly classified oob records
- permute values of variable m, re-count correctly classified oob records, subtract from first count
- for each variable, average over all trees

Importance of Variables



Mean Decrease Accuracy



Proximity

- N x N matrix of proximity values
- for each tree: if two records k and l are in the same leaf, increase proximity by one
- normalize proximity by dividing by number of trees

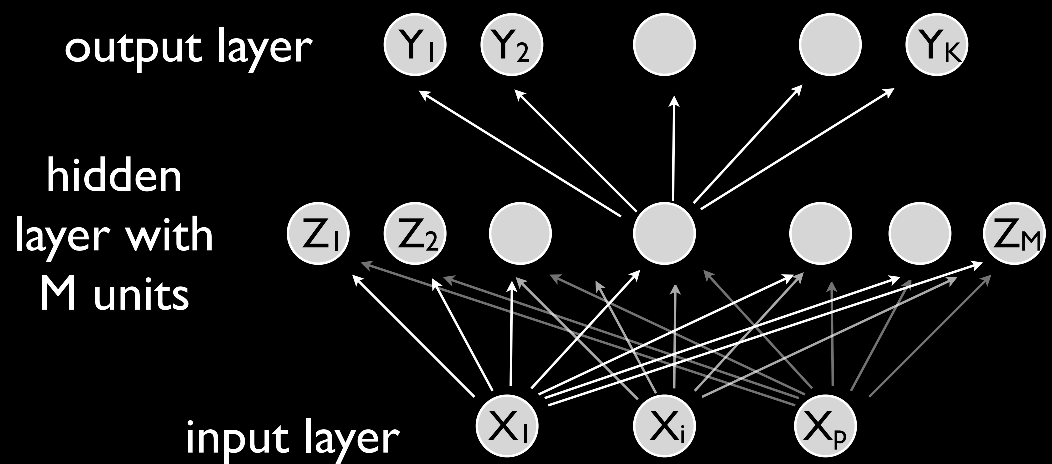
- size problematic

Neural Networks

- Historically used to model (biological) networks of neurons:
  - nodes represent neurons
  - edges represent nerves
  - network illustrates activity and flow of signals

Setup

- Response Y has K categories
- Network:



Formula Setup

- Relationship between layers:

$$
\begin{aligned}
Z_m &= \sigma(\alpha_{0m} + \alpha_m' X) \qquad m = 1, ..., M \\
T_k &= \beta_{0k} + \beta_k' Z \qquad k = 1, ..., K \\
f_k(X) &= g_k(T) \qquad k = 1, ..., K
\end{aligned}
$$

- where sigma is the *activation* function, e.g.

$$\sigma(\nu) = \frac{1}{1 + e^{-\nu}}$$

- g_k is final transformation between T and Y

Formula Setup

- g_k with continuous response usually chosen as identity, with categorical response usually *softmax:*

$$g_k(T) = \frac{e^{T_k}}{\sum_{\ell=1}^{K} e^{T_\ell}}$$

- i.e. estimates are positive and sum to 1

Issues with Neural Nets

- Model generally highly over-parametrized:

weights:
$$\{\alpha_{0m}, \alpha_m : m = 1, ..., M\} \qquad M(p+1)$$
$$\{\beta_{0k}, \beta_k : k = 1, ..., K\} \qquad K(M+1)$$

- Optimization problem convex & unstable -> convergence is tricky
- Over-parametrization leads to overfit at minimum

Fitting Strategies

- Standardize input variables X
- Pick starting values for alpha, beta close to zero (i.e. close to linear fit)
- Stop run before convergence (to avoid overfitting)
- Alternatively: use penalty on size of weights (decay)

$$\lambda \cdot \left(\sum \beta^2 + \sum \alpha^2 \right)$$

Fitting Strategies

- Pick large number of hidden units OR
  do cross-validation to figure out good size
- # parameters (and with it #units) bounded by sample size
- average results from set of networks (bagging)

Neural Networks are fickle

Choice of M is important

starting parameters are important - some models do not even come close to a good solution in 100 iterations