Data Mining (W4240 Section 001)
Bagging and Random Forests

Giovanni Motta

Columbia University, Department of Statistics

November 23, 2015

Outline

Outline

Model Fitting

What we have been doing:

- get data
- fit some models
- evaluate results
- choose best model, apply to test data

Today we consider an alternative approach:

- get data
- fit some models
- evaluate results
- combine models, apply to test data

Approaches of this type are generally called *ensemble methods*

Ensemble Methods

Ensembles methods use collections of models to get better predictive performance than any single model

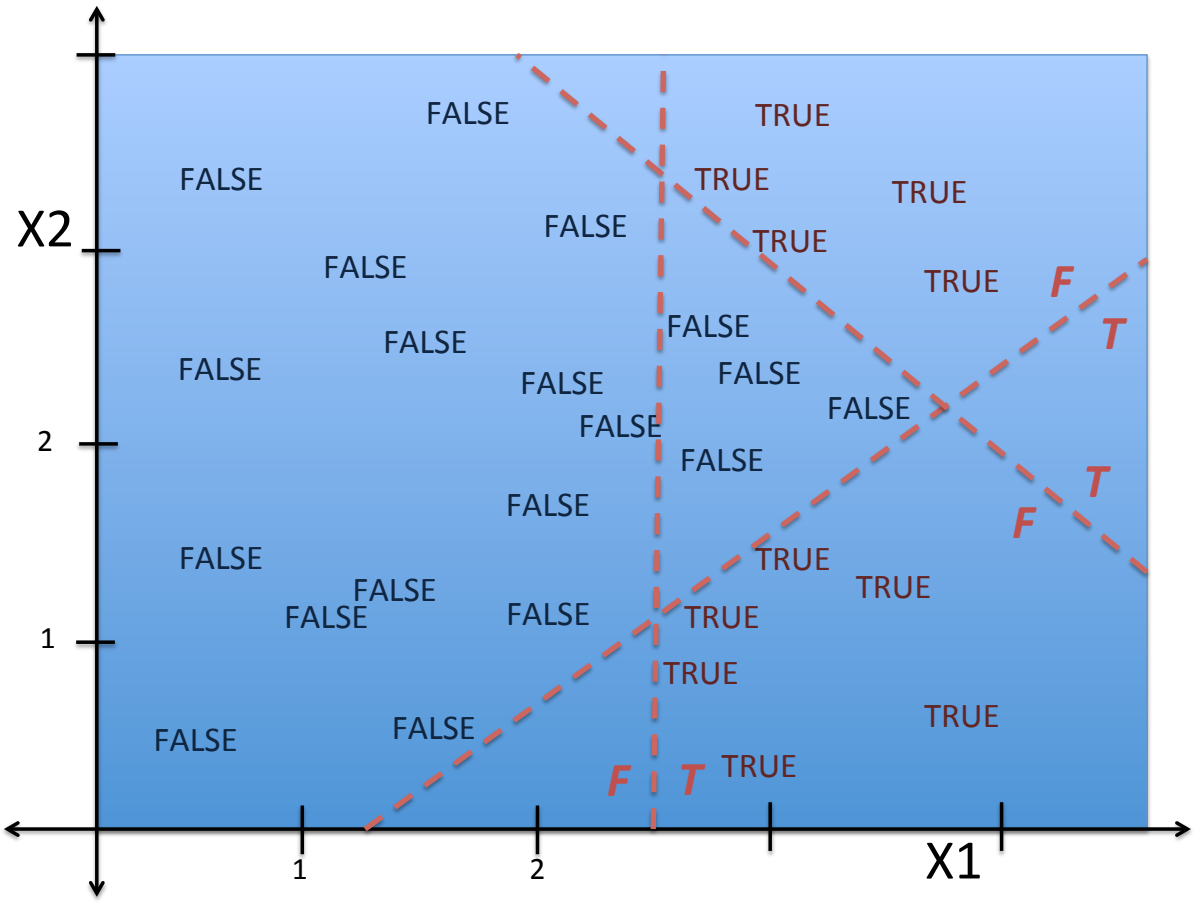
- get a collection of predictive models
- the ensemble predictor is an average of the underlying models

Why should this work?

- often easy to fit simple models well
- simple models have a limited hypothesis space
- if we average lots of *different* simple models, we can fit these well and have a large hypothesis space
- and we can reduce the variance of the estimator

Ensemble Methods

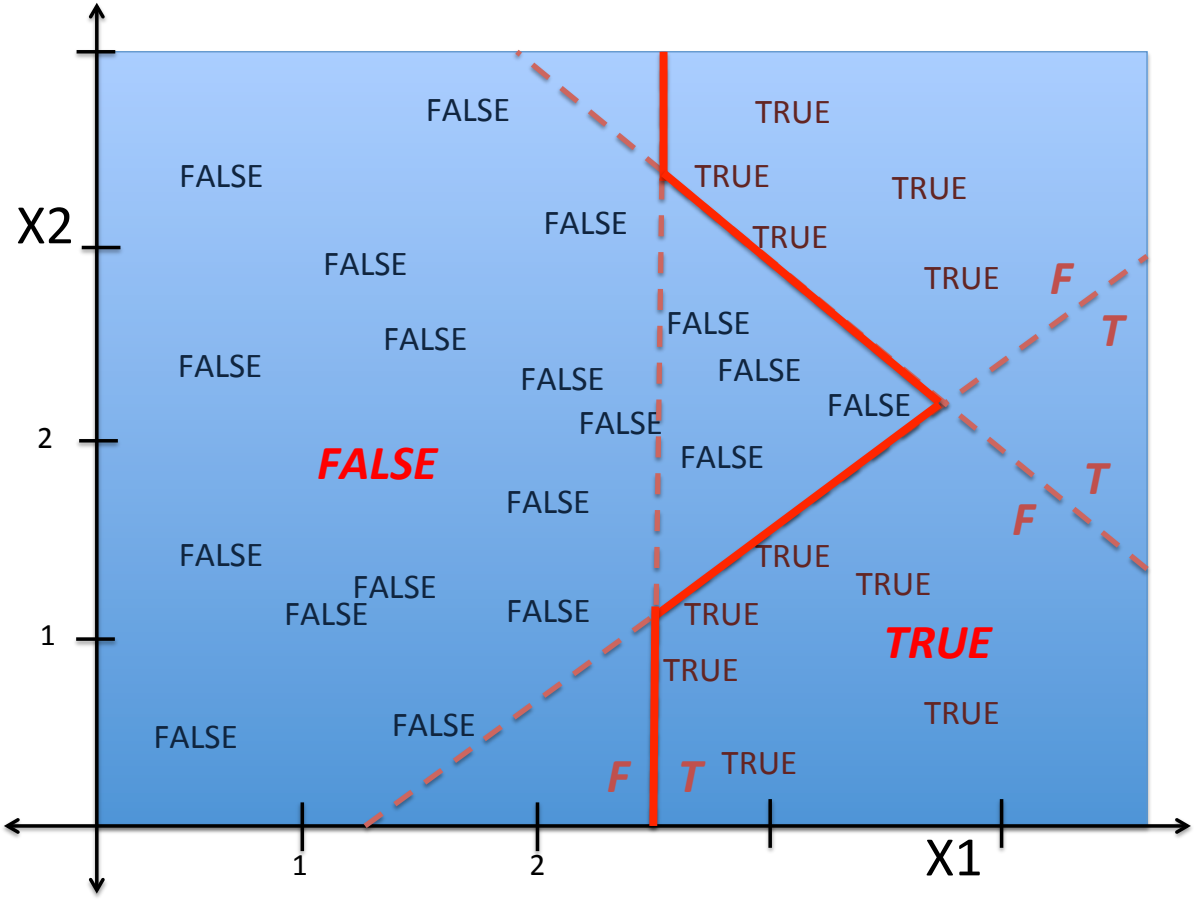
Let's think about linear classifiers: divide labels with a hyperplane



None divides the data particularly well, but better than random guessing... how can we combine them?

Ensemble Methods

Let's think about linear classifiers: divide labels with a hyperplane



Combine by assigning majority labels... now non-linear!

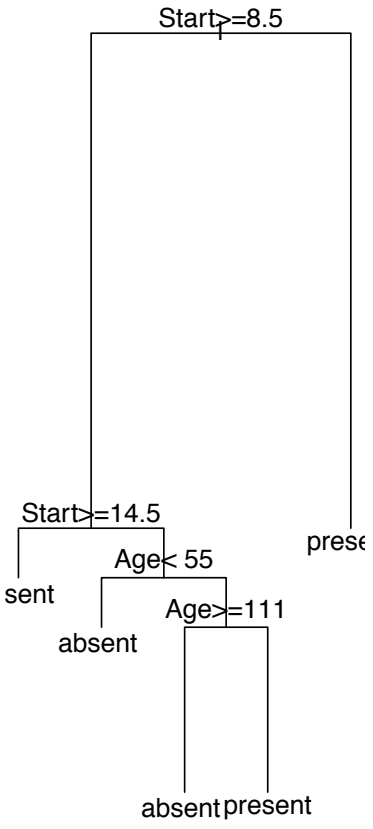
Outline

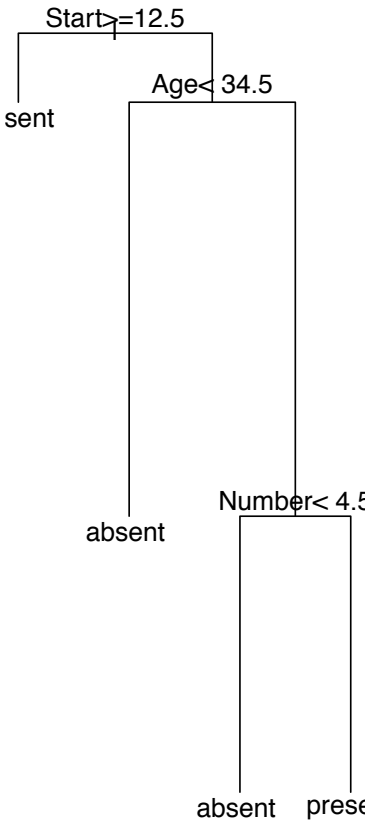
Ensemble Methods and Trees

Trees:

- trees are really flexible–good for regression and classification
- they tend to fit pretty well, but often do not have the best predictive error
- they are also *unstable*

Kyphosis data:





Ensemble Methods and Trees

Instability:

- small changes in data (or fitting method) produce big changes in outcome
- example: if we split the training data into two parts at random, and fit a decision tree to both halves, the results that we get could be quite different.
- this is good for ensembles! *diverse* results!

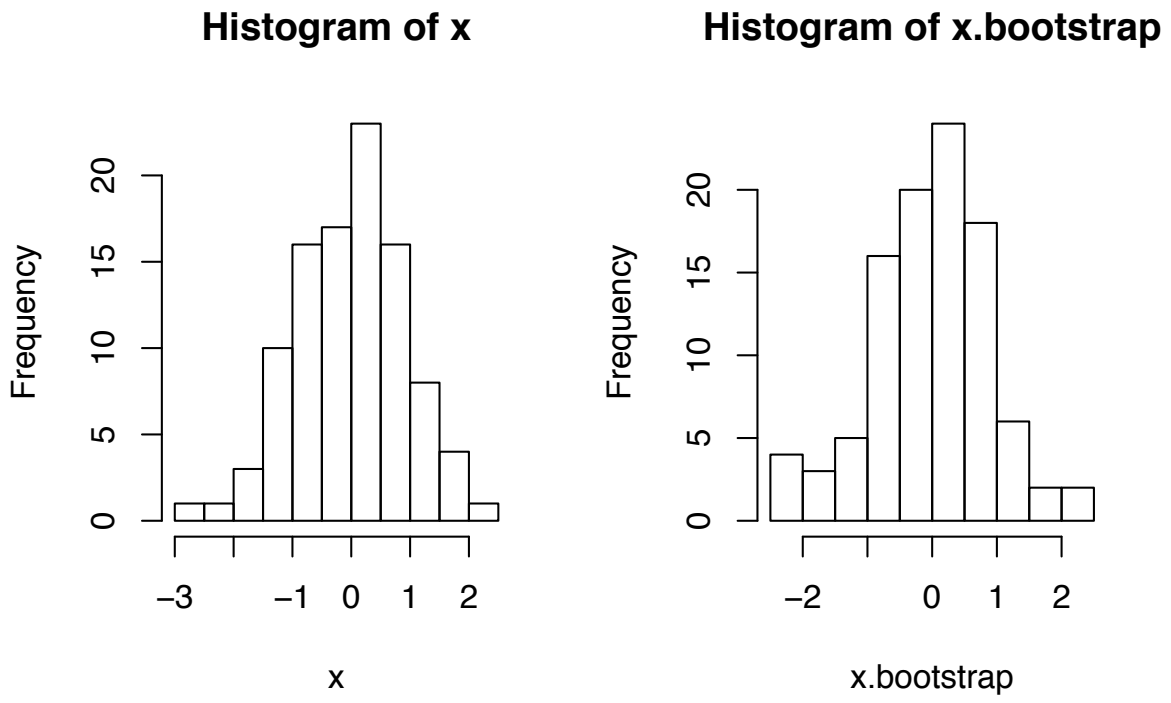
Idea: randomize the data (or tree building method) a bit to get even more diverse results!

Outline

Bagging

Idea: to get a set of trees, randomize the data

- use *bootstrapping* to get new datasets
- in bootstrapping, we create a new dataset by sampling with replacement from our current dataset



Bagging

Bagging procedure:

- use *bootstrapping* to get B new datasets (sampling with replacement from our current dataset)
- We then train our method on the $b-$th bootstrapped training set in order to get $\hat{f}_b^*(\mathbf{x})$, $b = 1, \ldots, B$ times
- fit a tree to every new dataset
- new estimator is average of bootstrap estimators

$$\hat{f}_{\text{bag}}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b^*(\mathbf{x})$$

We call this ensemble method *Bagging*: **B**oostrap **agg**regat**ing**

Bagging trees

- Trees are repeatedly fit to bootstrapped subsets of the observations (get B trees)
- These trees are grown deep (no pruning). Hence each individual tree has high variance, but low bias.
- Averaging these B trees reduces the variance!
  - regression: average the resulting B predictions
  - classification: record the class predicted by each of the B trees, and take a *majority vote*

Bagging

Example: Kyphosis data

```
# Make a bagged estimate for kyphosis

B <- 100
n.train <- 60
n.val <- length(kyphosis$Kyphosis)
n.test <- n.val - n.train
inds.train <- sample(1:n.val,n.train)
inds.test <- setdiff(1:n.val,inds.train)

kyphosis.train <- kyphosis[inds.train,]
kyphosis.test <- kyphosis[inds.test,]
```

Bagging

Example: Kyphosis data

```
# Make a bagged estimate for kyphosis

out.list <- list(B)
out.vals <- mat.or.vec(n.test,B)

base.fit <- rpart(Kyphosis ~ Age + Number + Start,
  data=kyphosis.train,parms=list(split='information'))
out.vals.base <- predict(base.fit,kyphosis.test,type="class")
```

Bagging

Example: Kyphosis data

```
# Make a bagged estimate for kyphosis

for (i in 1:B){
# Make a new training set
inds <- sample(1:n.train,n.train,replace=TRUE)
df.temp <- kyphosis.train[inds,]
fit.temp <- rpart(Kyphosis ~ .,
  data=df.temp,parms=list(split='information'))
out.list[[i]] <- fit.temp
out.vals[,i] <- predict(fit.temp,kyphosis.test,type="class")
}

bag.pred <- mat.or.vec(n.test,1)
for (i in 1:n.test){
n.temp <- mean(out.vals[i,])
if (round(n.temp)==2) bag.pred[i] <- "present"
else bag.pred[i] <- "absent"
}
```

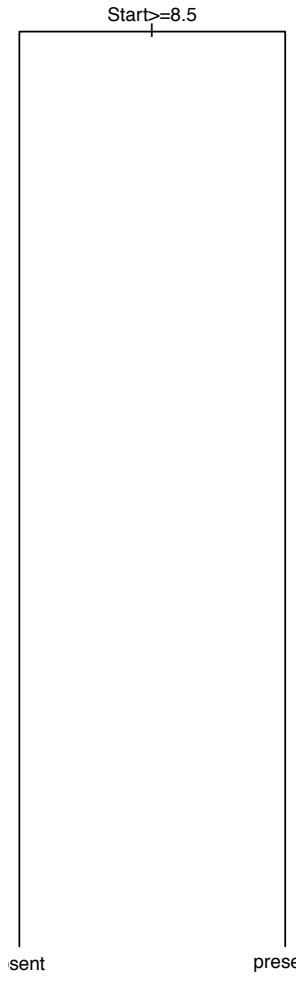
Bagging

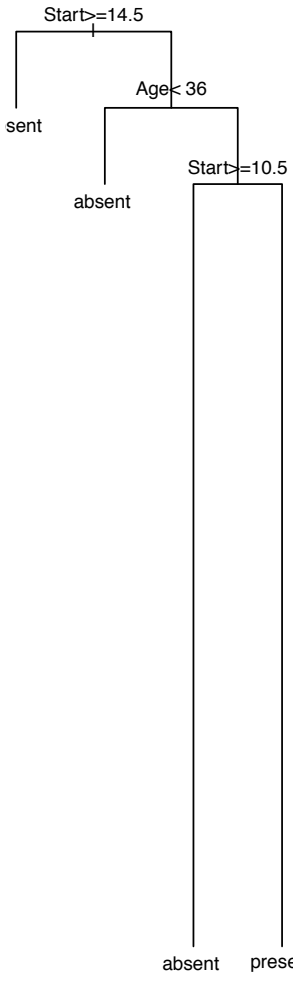
Example: Kyphosis data

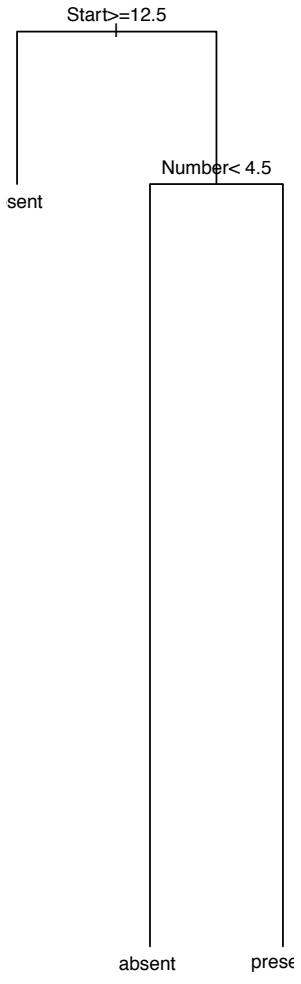
```
> mean((out.vals.base!=kyphosis.test$Kyphosis))
> mean((bag.pred!=kyphosis.test$Kyphosis))
> par(mfrow=c(1,4))
> plot(out.list[[1]])
> text(out.list[[1]],use.n=FALSE)
> plot(out.list[[2]])
> text(out.list[[2]],use.n=FALSE)
> plot(out.list[[3]])
> text(out.list[[3]],use.n=FALSE)
> plot(out.list[[4]])
> text(out.list[[4]],use.n=FALSE)
```

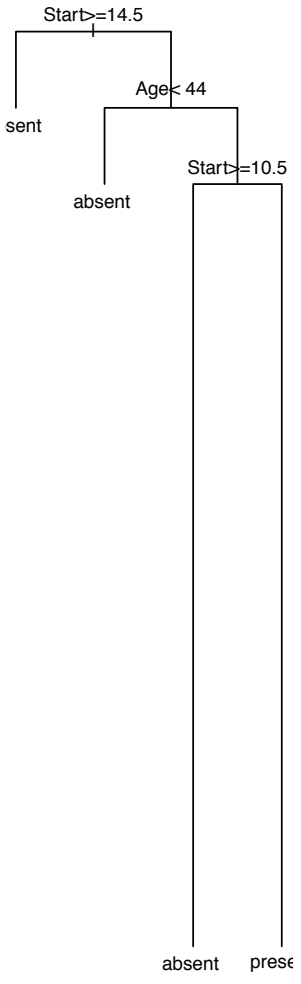
Bagging

Example: Kyphosis data









Bagging

Why does bagging work?

Recall:

Var(f1)/B+Σcov

$$MSE = (Estimator\ bias)^2 + (Estimator\ variance)$$

- bias is about the same between estimators
- averaging over (diverse!) estimators reduces variance
- ...which means lower predictive error

Bagging

Pros:

- easy to implement
- works better than trees on their own
- very fast and parallelizable

Cons:

- tends not to work as well as boosting (yet to come)
- works best with high variance, low bias, low correlation estimators

Bagging

Ensembles tend to work best with "not too complicated" estimators. Why?

Bagging

Ensembles tend to work best with "not too complicated" estimators. Why?

- simpler estimators are often less correlated
- may cover a larger part of the hypothesis space

Can we get these characteristics with bagged trees?

Out-of-bag Error

$\because$Bootstrap only use 2/3 observations

- Each bagged tree makes use of $\approx 2/3$ of the observations
- OOB: the remaining $1/3$ of the observations not used to fit a given bagged tree
- predict the response for the $i-$th observation using each of the trees in which that observation was OOB: get $\approx B/3$ predictions for the $i-$th observation.
- To obtain a single prediction for the $i-$th observation, we average these predicted responses (regression) or take a majority vote (classification is the goal).
- Obtain the OOB prediction for each of the n observations, and compute the overall OOB-MSE (for a regression problem) or classification error (for a classification problem) can be computed.

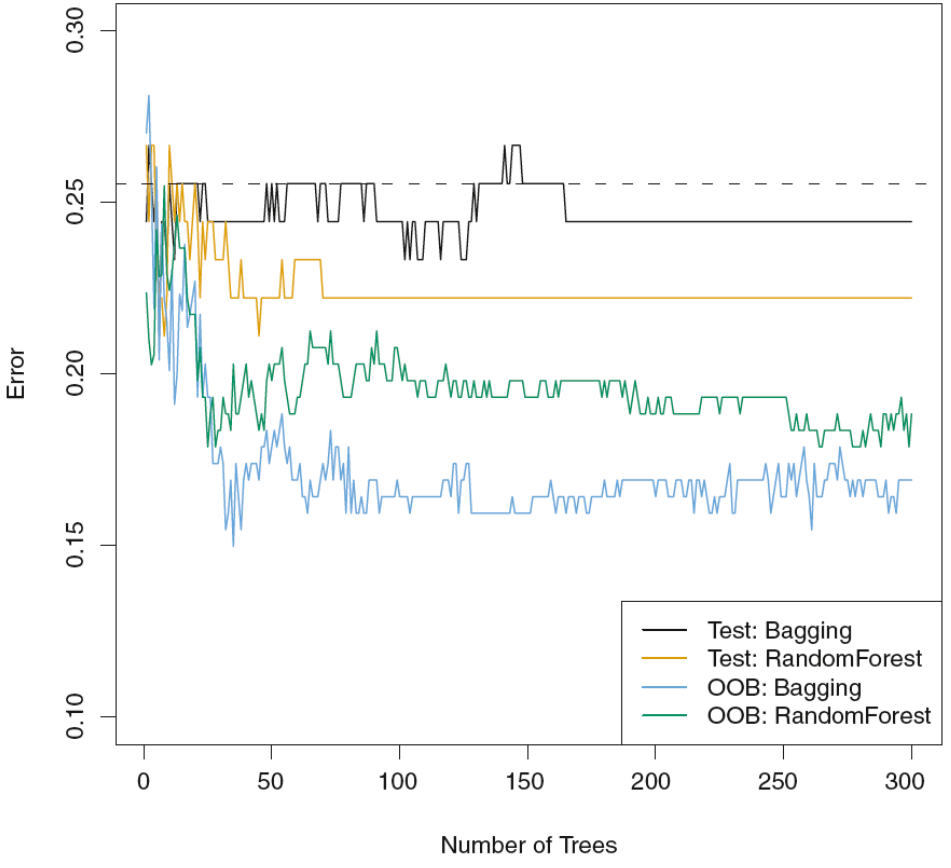
Out-of-bag Error

Leave One Out CV $\approx$ Out Of Bag

- **OOB error**: valid estimate of the **test error** for the bagged model (the response for each observation is predicted using only the trees that were not fit using that observation).
- If B is sufficiently large, **OOB error** is equivalent to **LOO-CV-error**:
- The OOB approach for estimating the test error is particularly convenient when performing bagging on large data sets for which cross-validation would be computationally onerous.

Heart data: OOB Error vs Random Forest with $m = \sqrt{p}$

Black and orange: Test Error as a function of B (number of bootstrapped training sets used).
Dashed : Test Error resulting from a single classification tree.
Green and Blue: OOB error



Variable importance

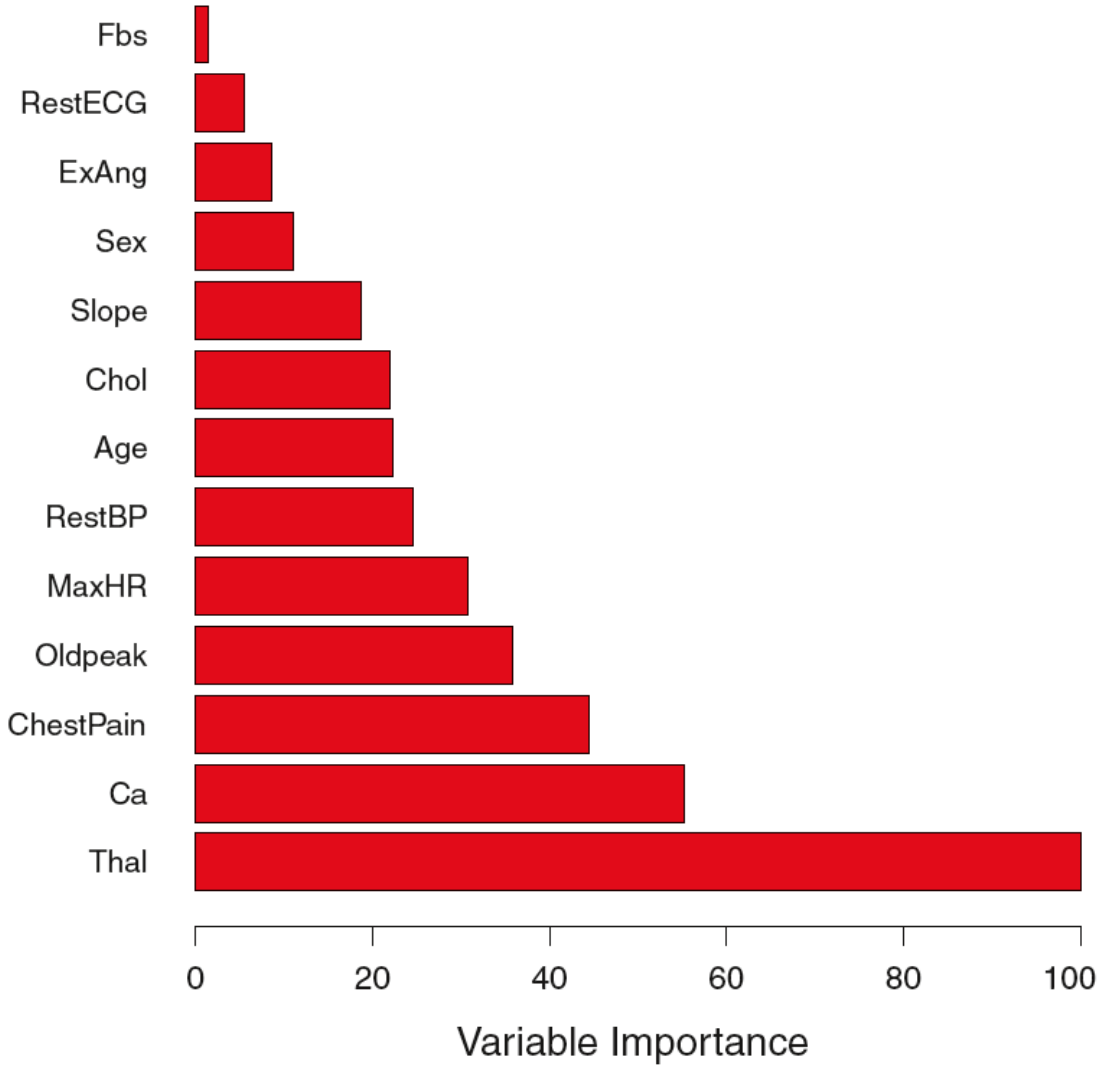
Total RSS: $\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})$

Gini index (K classes): $\sum_{k=1}^{K} p_{mk}(1 - p_{mk})$ Used for importance of predictor

where p_{mk} = proportion of training observations in the $m-$th region coming from the $k-$th class.

- The collection of bagged trees difficult to interpret
- Nevertheless, one can obtain an overall summary of the importance of each predictor
  - Bagging regression trees: we record the total amount that RSS is decreased due to splits over a given predictor, averaged over all B trees.
  - Bagging classification trees: add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.
- A large value indicates an important predictor.

Heart data: Variable importance



Foreshadowing

- Bagging: we average all our estimators together
- Don't we believe that some will be more useful than others?
- Could we create a weighted average of estimators?
- Next time...

Outline

Random Forests

- Random forests improve over bagged trees by decorrelating the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- However: each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors ($m < p$, usually $m = \sqrt{p}$).
- The split is allowed to use only one of those m predictors
- Note: at each split in the tree, the algorithm is not even allowed to consider a majority of the available predictors. Why?

Random Forests

- Random forests improve over bagged trees by decorrelating the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- However: each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors ($m < p$, usually $m = \sqrt{p}$).
- The split is allowed to use only one of those m predictors
- Note: at each split in the tree, the algorithm is not even allowed to consider a majority of the available predictors. Why?
- The predictions from the bagged trees will be highly correlated (in particular when some predictors are stong).
- Indeed: **averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities**

Random Forests

Suppose we have a strong predictor

- most or all of the trees will use this strong predictor in the top split
- Random forests: on average $(p-m)/p$ of the splits will not even consider the strong predictor, and so other predictors will have more of a chance.
- This process as decorrelates the trees, thereby making the average of the resulting trees less variable and hence more reliable.
- if a random forest is built using $m = p$, then this amounts simply to bagging.
- Small value of m: helpful when we have a large number of correlated predictors.

Random Forests

Want to *decorrelate* bagged trees. What if we grow trees randomly?

- follow same branching path if inputs are the same
- small changes in inputs can lead to large changes in outputs
- so what if we force the trees to split on different attributes?

Randomly select a subset of attributes that it can split on!

Random Forests

Algorithm of Random Forests!!!

Idea: another way to get diverse estimators is by randomizing the tree generation

- randomly select a small subset of features before each split
- fit a tree on the reduced data
- don't prune!
- do this B times
- average results to get new predictor
- called *random forests*

Random Forests

Why does this work?

- suppose we have B random variables, $X_1, \ldots, X_B$
- each has variance σ^2
- pairwise correlation ρ
- set $Y = \frac{1}{B} \sum_k X_k$

Then,

$$\mathbb{E}[Y] = \frac{1}{B} \sum_{k=1}^{B} \mathbb{E}[X_k]$$

$$\mathrm{Var}(Y) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

If ρ≈1 : Var(Y)≈sigma^2

If ρ≈0 : Var(Y)≈sigma^2/B (Most cases in random sample)

Random Forests

Variable importance:

- how important is each variable to the model?
- at each tree split, the improvement in the split criterion is the importance given to the splitting variable
- add importance over all trees

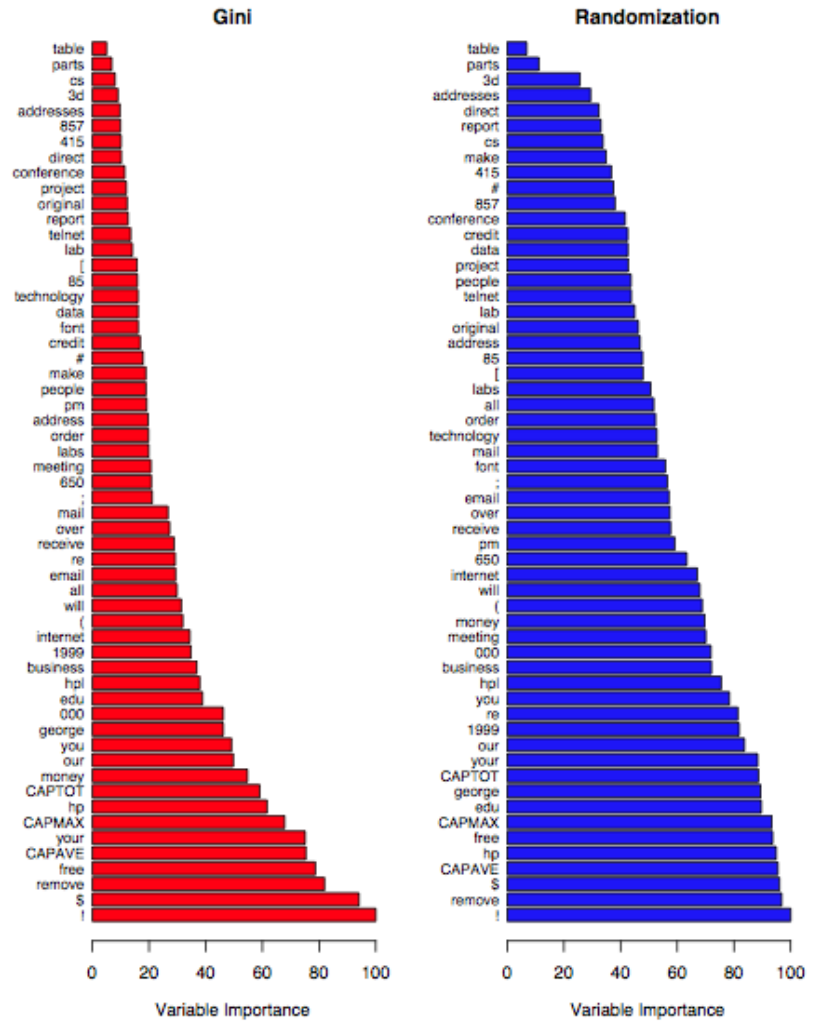


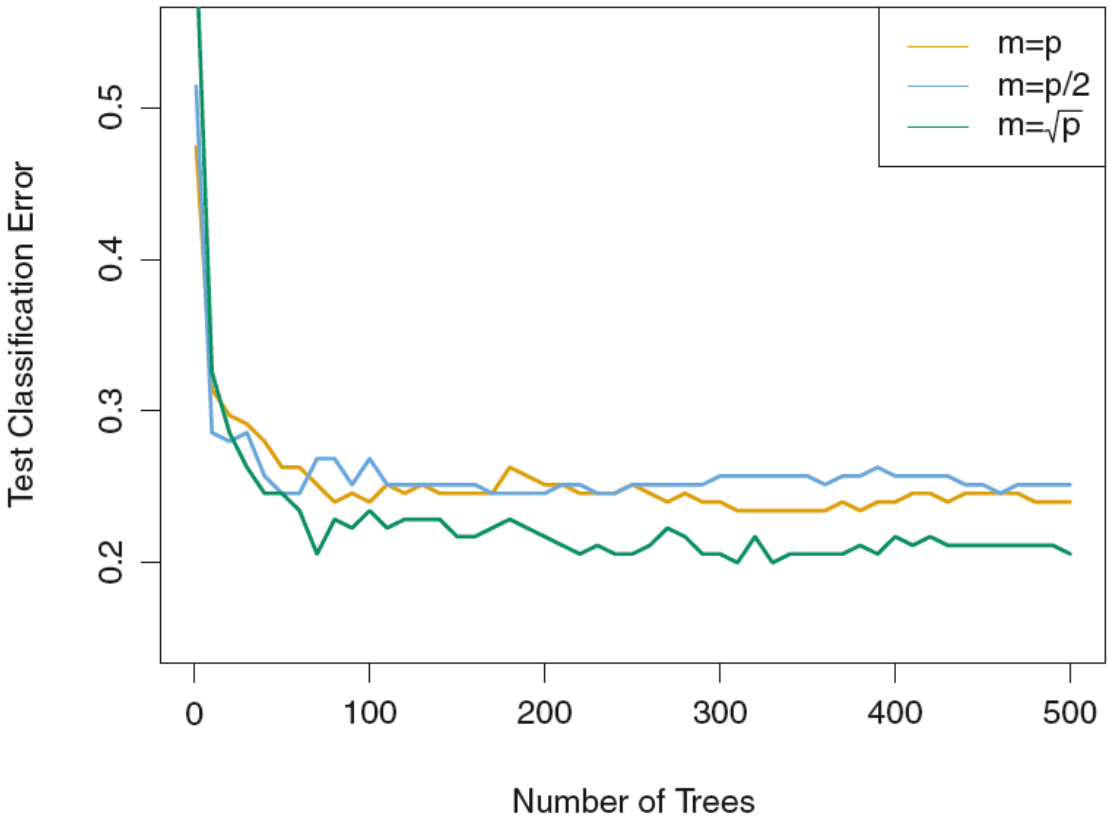
FIGURE 15.5. *Variable importance plots for a classification random forest grown on the* **spam** *data. The left plot bases the importance on the Gini splitting index, as in gradient boosting. The rankings compare well with the rankings produced by gradient boosting (Figure 10.6 on page 354). The right plot uses* OOB *randomization to compute variable importances, and tends to spread the importances more uniformly.*

Gene expression

- 4,718 genes measured on tissue samples from 349 patients.
- each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer.
- Our goal: use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.
- We apply random forests to the training set for three different values of the number of splitting variables m.
- the choice $m = \sqrt{p}$ gave a small improvement in test error over bagging ($m = p$) in this example.
- As with bagging, random forests will not overfit if we increase B, so in practice we use a value of B sufficiently large for the error rate to have settled down.

Gene expression: test error

Random forests: $m < p$
Bagging: $m = p$



Random Forests

To implement random forests in R, use `randomForest` package

- it works very similarly to `rpart`

```
> library(randomForest)
> tree.rf <- randomForest(Kyphosis ~ Age + Number + Start,
  data=kyphosis.train)
> out.rf <- predict(tree.rf,kyphosis.test)
```

Ensembles and Trees

- Ensembles are a great way to deal with the stability issues of trees
- Usually produce results with better predictive error
- Random forests are a great black box method for classification and regression–often a good first (or second or third) method to try
- Scales well to large datasets ($\mathcal{O}(10^7)$ observations, thousands of covariates)
- However, results are not as interpretable as individual trees