Trees and Forests

Thiemo Fetzer

University of Chicago & University of Warwick

May 18, 2017

Regression Trees

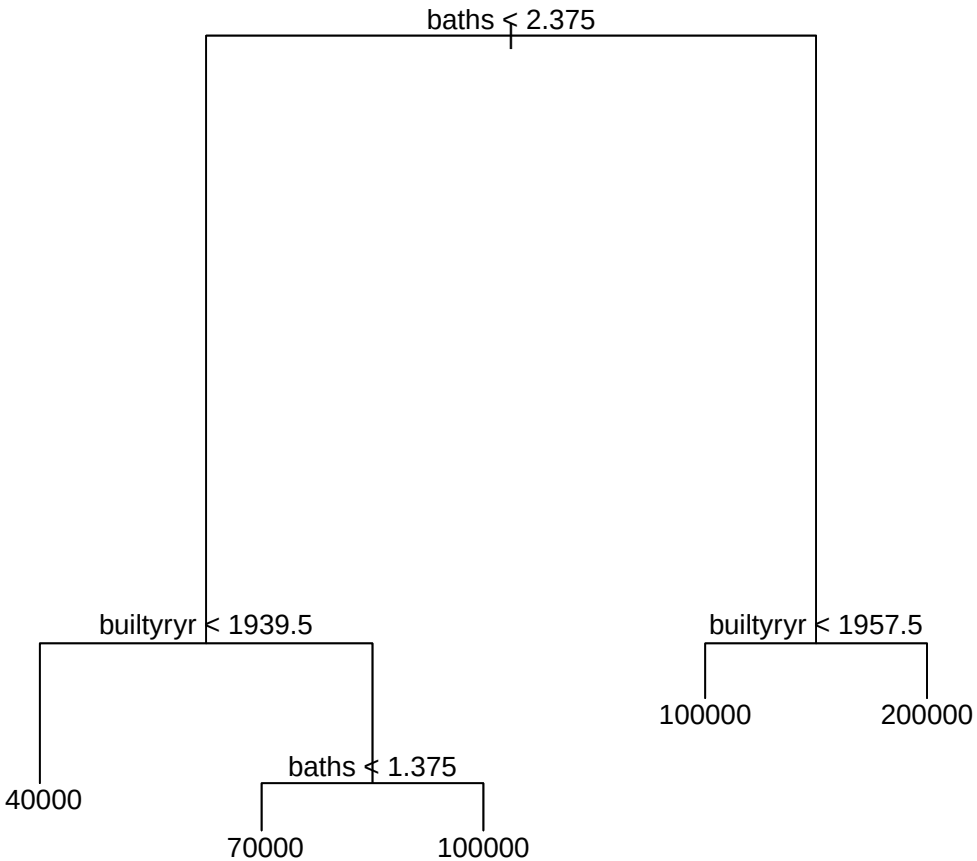
- Linear regression is a global model, where there is a single predictive formula holding over the entire data-space. Very difficult to capture or adequately model non-linearities, e.g. interactions.
- The idea of Regression Trees is to sub-divide, or partition, the feature space into smaller regions, where the interactions are explicit.
- Divide the feature space, that is the $\mathbf{X}$ into J distinct regions that do not overlap, call them $R_1, ..., R_J$.
- For every test observation x_j that falls into a region R_i, the prediction $\hat{f}(x_j)$ is simply the mean of the training observations y_i that fall into the region R_i

$$\hat{y}_{R_j} = 1/n_j \sum_{i \in R_j} y_i$$

where n_j is number of training observations that fall into R_j.

Regression Trees: Hedonic Pricing Example

```
library(tree)
set.seed(1312)
SAMPLE<-sample(1:nrow(HOUSES),18000)
TRAINING<-HOUSES[SAMPLE]
res<-tree(soldprice~builtyryr+baths, data=TRAINING)
plot(res)
text(res)
```



Top Down Greedy Binary Recursive Splitting

Define two half planes as $R_1(j, s) = \{X | X_j < s\}$ and $R_2(j, s) = \{X | X_j \geq s\}$.

Algorithm (*Binary Recursive Splitting*)

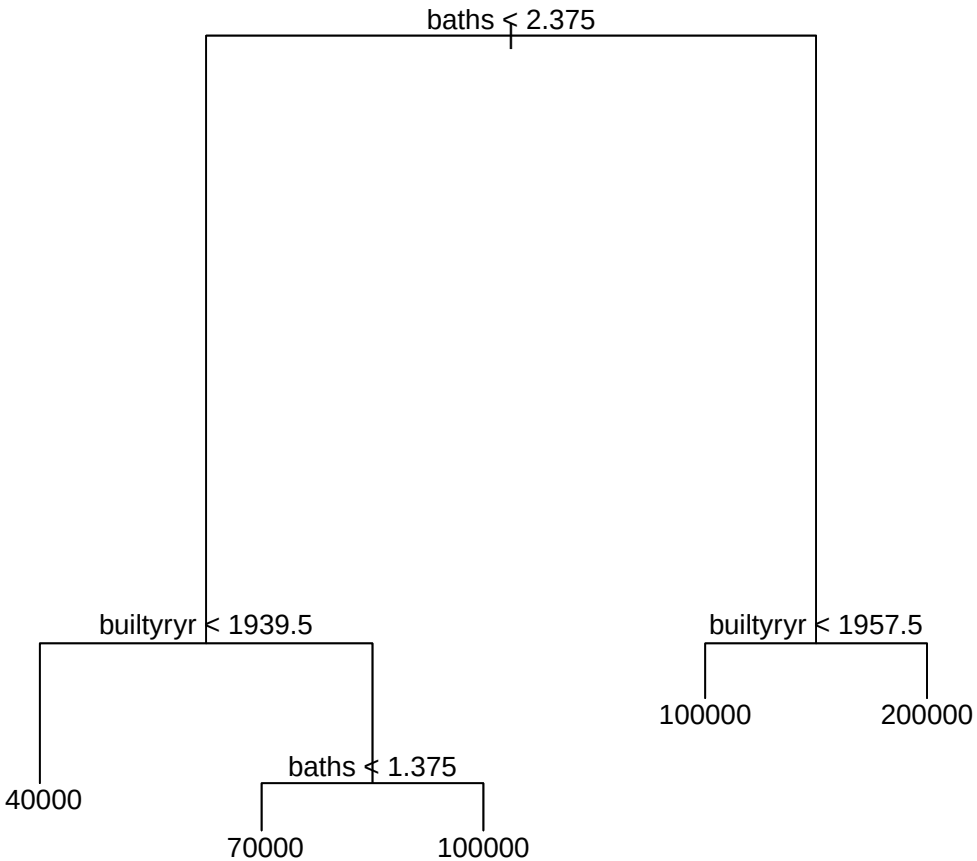
1. *Initialize with empty tree* $\mathcal{T}$
2. *For each* $j = 1, ..., k$, *find the* s *that minimizes*

$$\min_s RSS_j = \sum_{i: x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

3. *Choose the* j *and corresponding optimal* $\tilde{s}$ *at which the value of* RSS_j *is lowest and split the data at this point into two half planes* R_1, R_2.
4. *For each new half plane, repeat (2)-(4) so long as cutting criterion is not reached.*

Non-Linearities

```
library(tree)
set.seed(1312)
SAMPLE<-sample(1:nrow(HOUSES),18000)
TRAINING<-HOUSES[SAMPLE]
res<-tree(soldprice~builtyryr+baths, data=TRAINING)
plot(res)
text(res)
```



Regression Trees can generate non-linear models

- In the above example, its easy to show that we have interactive effects.
- We can, in fact, write the tree as a sequence of and interaction between different indicator functions. We have $X_1 =$ Bathrooms, $X_2 =$ Built Year

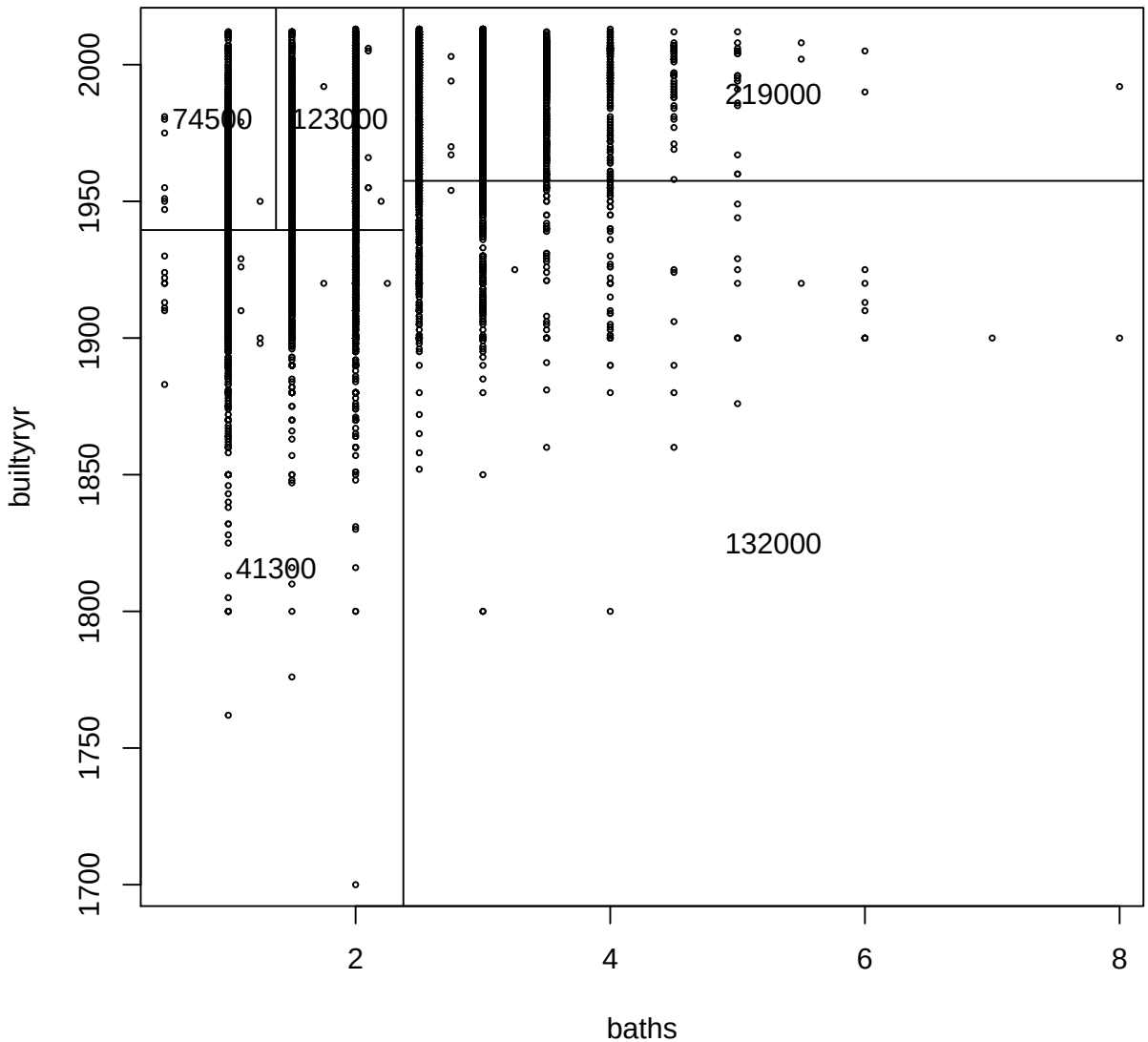
$$I_1 = \mathbf{I}(X_1 \leq 2.375)$$
$$I_2 = \mathbf{I}(X_1 \leq 1.375)$$
$$I_3 = \mathbf{I}(X_2 \leq 1939.5)$$
$$I_4 = \mathbf{I}(X_2 \leq 1957.5)$$

- Our regression tree is a model of the form:

$$\begin{aligned} Y &= \beta_0 \times \mathbf{I}(X_1 \leq 2.375) \times \mathbf{I}(X_2 \leq 1939.5) \\ &+ \beta_1 \times \mathbf{I}(X_1 \leq 2.375) \times \mathbf{I}(X_2 > 1939.5) \times \mathbf{I}(X_1 \leq 1.375) \\ &+ \beta_2 \times \mathbf{I}(X_1 \leq 2.375) \times \mathbf{I}(X_2 > 1939.5) \times \mathbf{I}(X_1 > 1.375) \\ &+ \beta_3 \times \mathbf{I}(X_1 > 2.375) \times \mathbf{I}(X_2 < 1957.5) \\ &+ \beta_4 \times \mathbf{I}(X_1 > 2.375) \times \mathbf{I}(X_2 < 1957.5) \end{aligned}$$

Regression Trees: Hedonic Pricing Example

```
partition.tree(res)
points(TRAINING[, c("baths","builtyryr"),with=F], cex=.4)
```



How to build a tree? Binary Recursive Splitting

- As before, we want to minimize prediction error RSS, computed as

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

- There is an uncountably large ways that you could split the predictor space into $R_1, ..., R_J$ boxes.
- We follow an algorithmic approach known as *greedy recursive binary splitting*.
- At each step, identify the variable j and the split point s such that the resulting reduction in training RSS is the largest.
- Perform this step iteratively on the sub trees until you are left with a mininum number of observations in each subtree (there are alternative stopping criteria)

Bias vs Variance for Large Trees

- In the extreme case, each terminal node will contain only a single observation.
- This obviously results in bad prediction performance on the test set due to high variance, as we are overfitting the data.
- So we could do two things:
  - Require each cut to result in a significant reduction in RSS. Why is that not a good idea?
  - Build a very large tree, and then remove (or combine) regions - **tree pruning**

Pruning of Trees

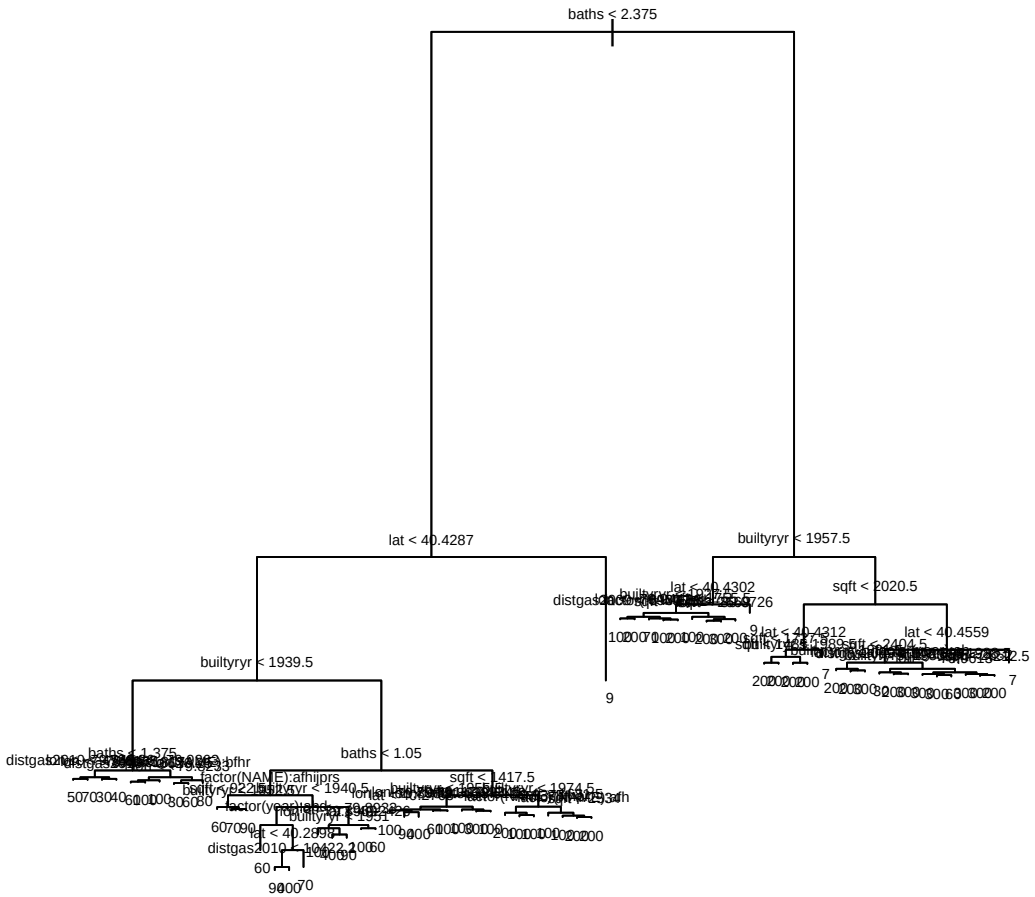
- Starting out from a large tree $\mathcal{T}$, we could "prune" the tree, i.e. remove and combine regions from the bottom and creating subtrees.
- We want to select a subtree that has lowest test error. For any subtree, we could estimate test error using cross-validation, but this is (often) not feasible due to large number of trees (like best subset selection)
- *Cost complexity pruning* considers a sequence of trees that are a function of a tuning parameter α.

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

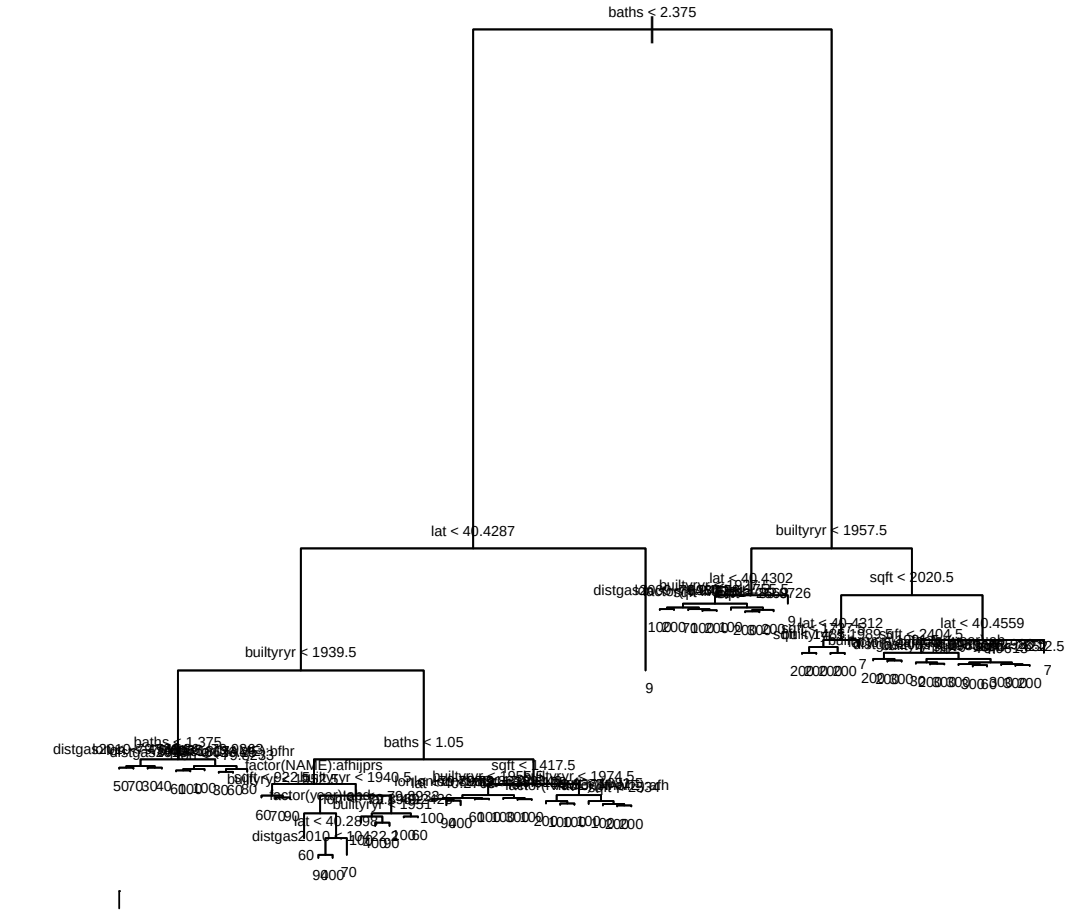
$|T|$ is the number of terminal nodes in a tree, i.e. the number of boxes, R_m is rectangle pertaining to terminal node m and $\hat{y}_{R_m}$ is mean of training observations in R_m.

Complex "Bushy" Tree

```
res<-tree(soldprice/1000~ sqft+baths+builtyryr+distgas2009+distgas2010+lat+lon+factor(NAME)+factor(year), data=TRAINING,
          control=tree.control(nrow(TRAINING),mincut=2,minsize=5,mindev=0.0005))
plot(res)
text(res, cex=.5)
```

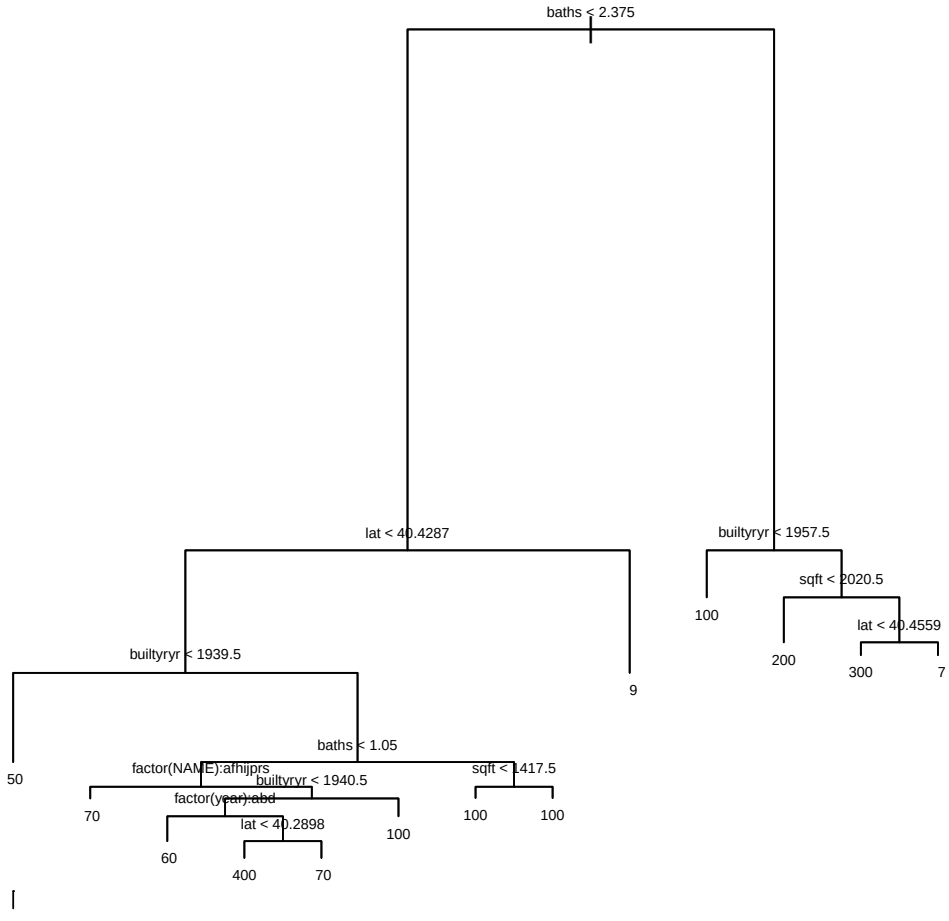


Pruning a "Bushy" Tree



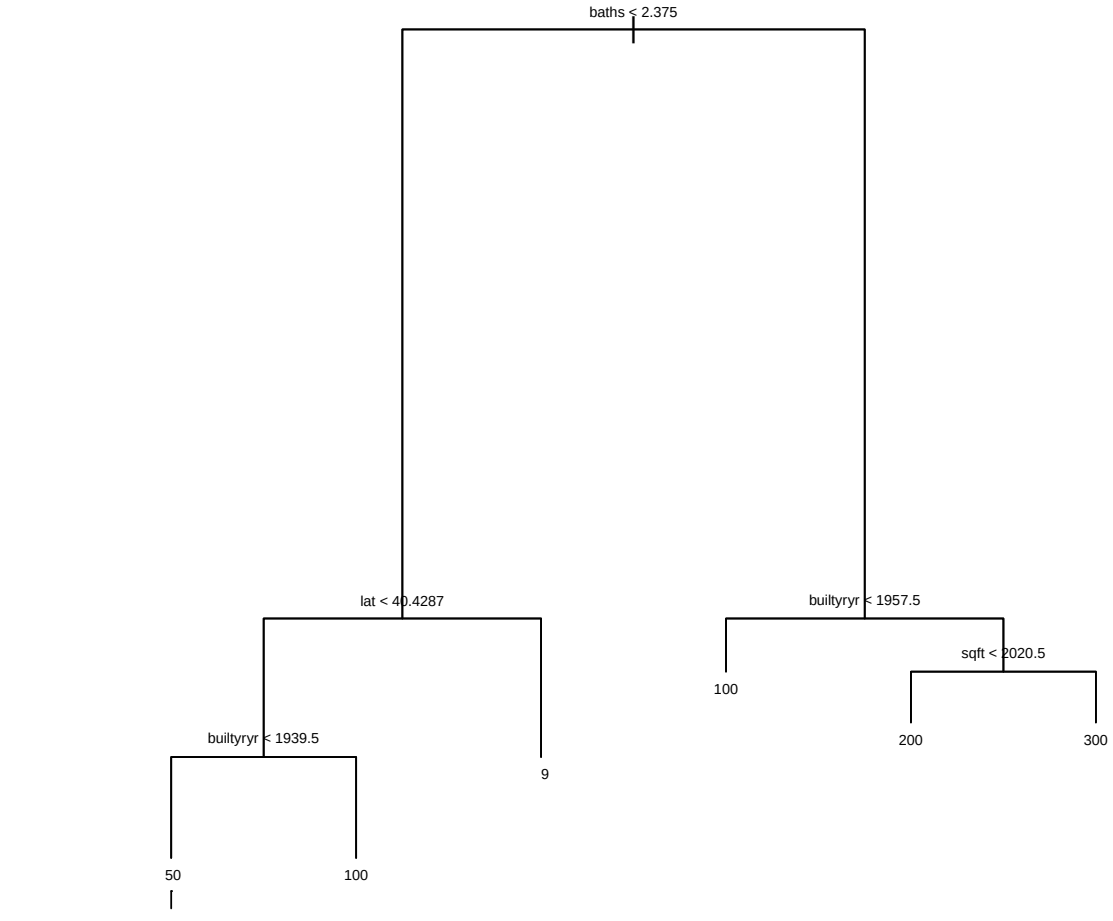
Cost Complexity 0

Pruning a "Bushy" Tree



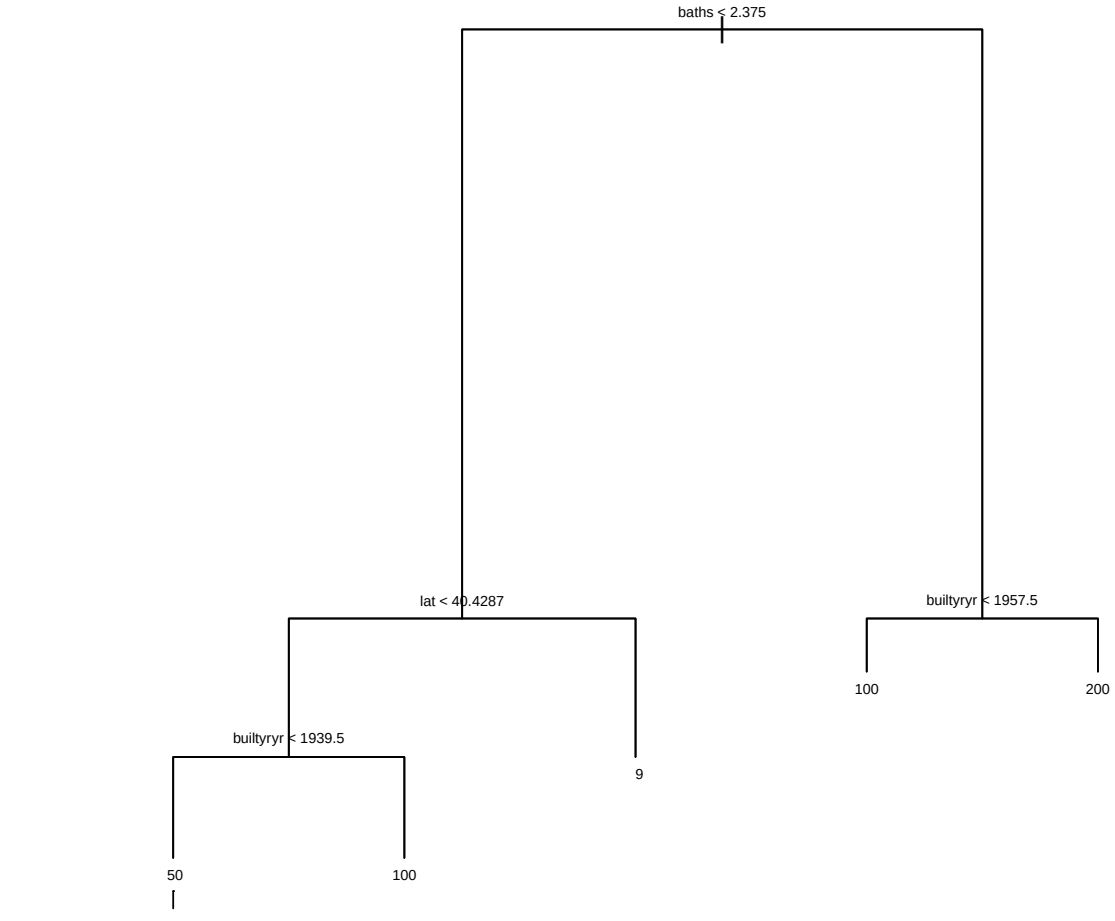
Cost Complexity 975000

Pruning a "Bushy" Tree



Cost Complexity 3475000

Pruning a "Bushy" Tree

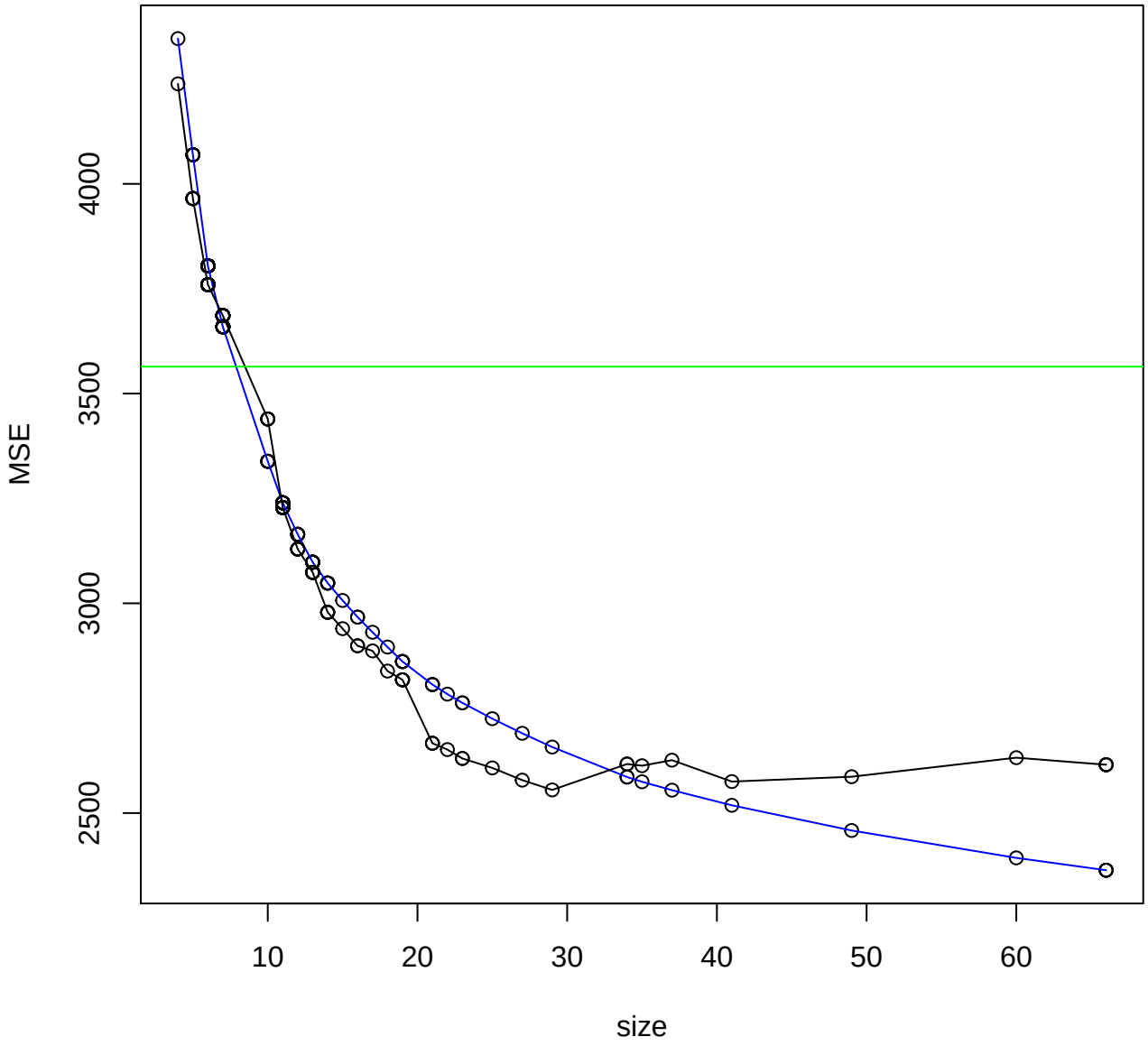


Cost Complexity 4975000

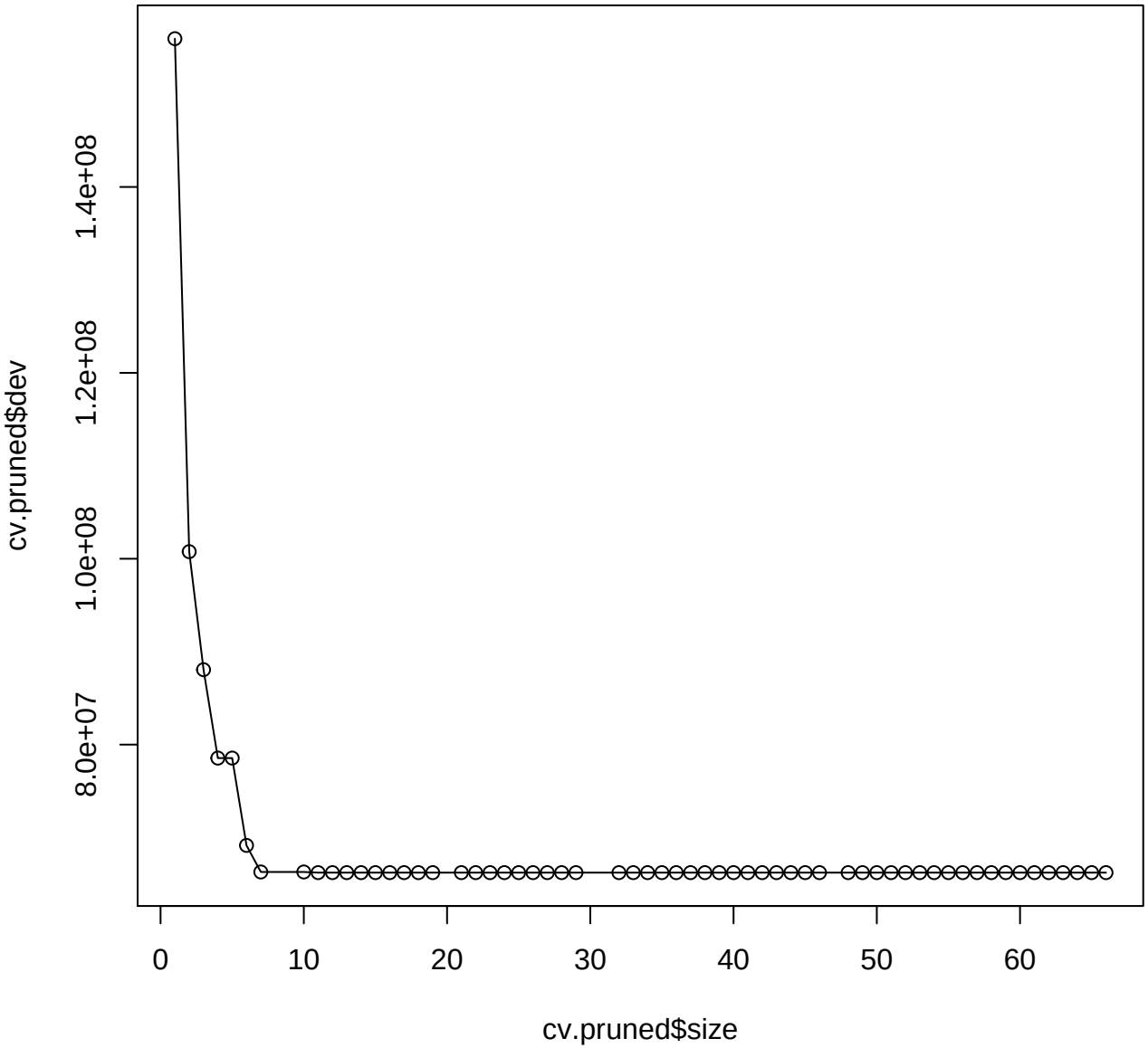
Cross Validation Error, Test Error and Training Error

- Hence the idea is to vary α and have the computer compute different subtrees, one for each value of α.
- As $\alpha \to 0$, we obtain the unpruned tree.
- As $\alpha \to \infty$, we obtain a tree with a single terminal node.
- How to decide on optimal α? Two approaches...
  - We can perform a *validation set* approach to compute test and training error and based on the minimum point of test error, choose
  - Alternatively, we can obtain a sequence of subtrees and then compute the K-fold cross validation error.
- Present each in turn for our example

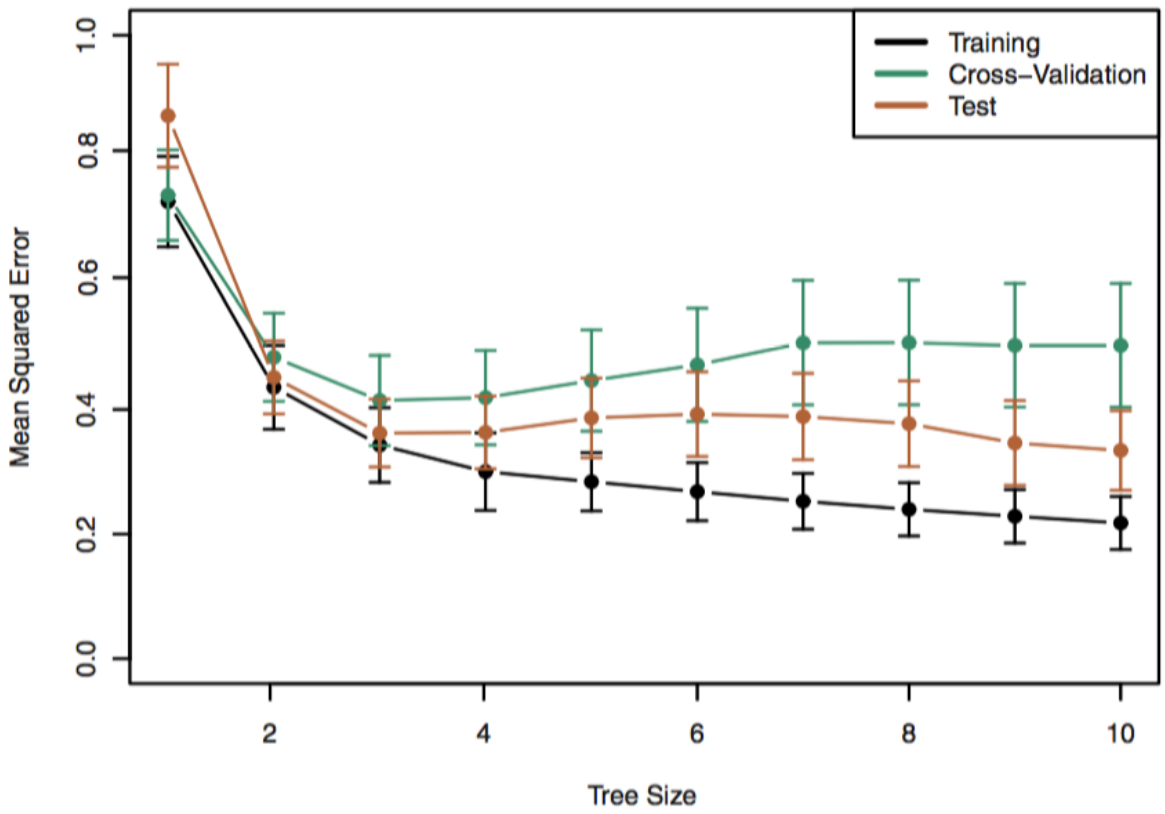
Validation Set Approach: Training and Test MSE



10 Fold Cross Validation



More Common Pattern: CV, Validation Set and Training Error



Building Regression Trees

Algorithm (*Building Regression Trees*)

1. *Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.*
2. *Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of* α
3. *Use K-fold cross-validation to choose* α*. That is, divide the training observations into K folds. For each* $k = 1, \ldots, K$*:*
   - 3.1 *Repeat Steps 1 and 2 on all but the kth fold of the training data.*
   - 3.2 *Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of* α
4. *Average the results for each value of* α*, and pick to minimize the average error.*
5. *Return the subtree from Step 2 that corresponds to the chosen value of* α*.*

Regression Trees versus Linear Models

- We can write a Regression Tree predictive model as

$$f(X) = \sum_{i=1}^{M} c_m \mathbf{I}(X \in R_m)$$

- The basis functions are indicator functions and c_m is the estimated average of the response variable.
- It becomes clear from the previous examples, that regression trees are highly non-linear as they allow interactive effects between variables. How?
- Rectangular partitioning of predictor space may be adequate, but in some cases, the partition may be non-linear.

Regression Trees Pros and Cons

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than do the regression approaches seen so far.
- Trees can easily handle qualitative predictors without the need to create dummy variables.
- Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression methods.
- Additionally, trees can be very non-robust. In other words, a small change in the data can cause a large change in the final estimated tree.

Next, we explore Bagging and Boosting as an approach to aggregate many regression trees into Forests.

Plan

Classification Trees

Bagging and Random Forests

Classification Trees

- The only fundamental difference between regression trees and classification trees is that our response varibale is not numeric, but categorical.
- As such, our objective function is not to minimize RSS, but rather to minimize classification error rate or a measure of *node purity*.
- Our *decision rule* on labelling a new observation will require us to label a test observation $\mathbf{x}_j$ falling into region R_i as having the label k if label k is the most commonly occuring among the training observations falling into region R_i.

Objective Function

- That is, we assign to a new observation the **modal** class of our training data that falls in region R_i
- What is our expected "misclassification error rate" (our alternative to RSS)?

$$E = 1 - \max_k(\hat{p}_{jk})$$

where $\hat{p}_{jk}$ represents the proportion of training observations in the j-th region that are from the kth class.

Alternative Objective Function

- It turns out that misclassification error rate is often not preferred as it produces splits with high node impurity.
- Rather than using misclassification error as a measure of quality of a particular split, there are two alternatives, in particular the *Gini Index*

$$G = \sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk})$$

- as a measure for total variance of a split region j, the *Gini index* takes on a small value if all of the $\hat{p}_{jk}$'s are close to zero or one. [see if there is a single class, this is a bell shaped curve]
- Its clear that this captures a notion of *node purity* - a small value of G indicates that a node j contains predominantly observations from a single class.

Alternative Objective Function (2)

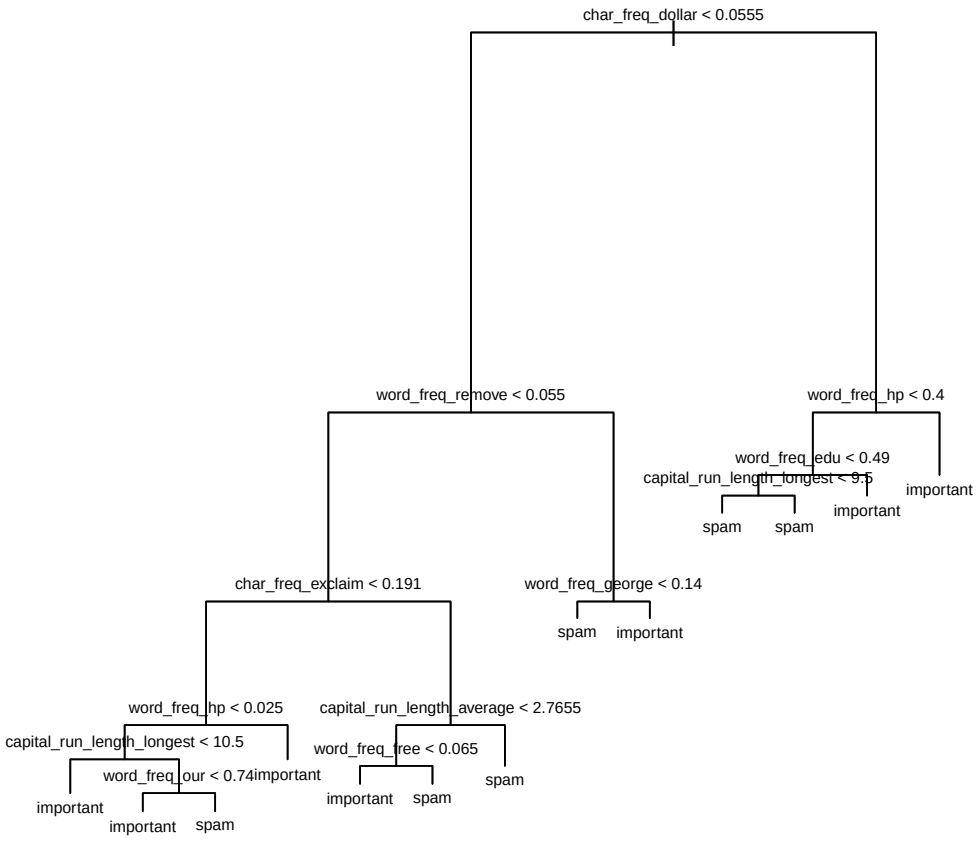
- An alternative to the Gini index is *cross-entropy*, given by

$$D = -\sum_{k=1}^{K} \hat{p}_{jk} \log \hat{p}_{jk}$$

- Using L'Hospital rule, one can show that $\lim_{p \to 0} D \to 0$ and $\lim_{p \to 1} D \to 0$
- Therefore, like the Gini index, the cross-entropy will take on a small value if the j-th node is pure. In fact, it turns out that the Gini index and the cross-entropy are quite similar numerically.

Spam Classification Example

```
library(tree)
 email <- read.csv("R/spam.csv")
email$spam <- factor(email$spam,levels=c(0,1),labels=c("important","spam"))
validation<-sample(1:nrow(email), 500)
res<-tree(spam~., data=email[-validation])

plot(res)
text(res, cex=0.6)
```



Test Error

```
pred<-predict(res, email[validation,], type="class")

table(pred, email[validation,]$spam)
```

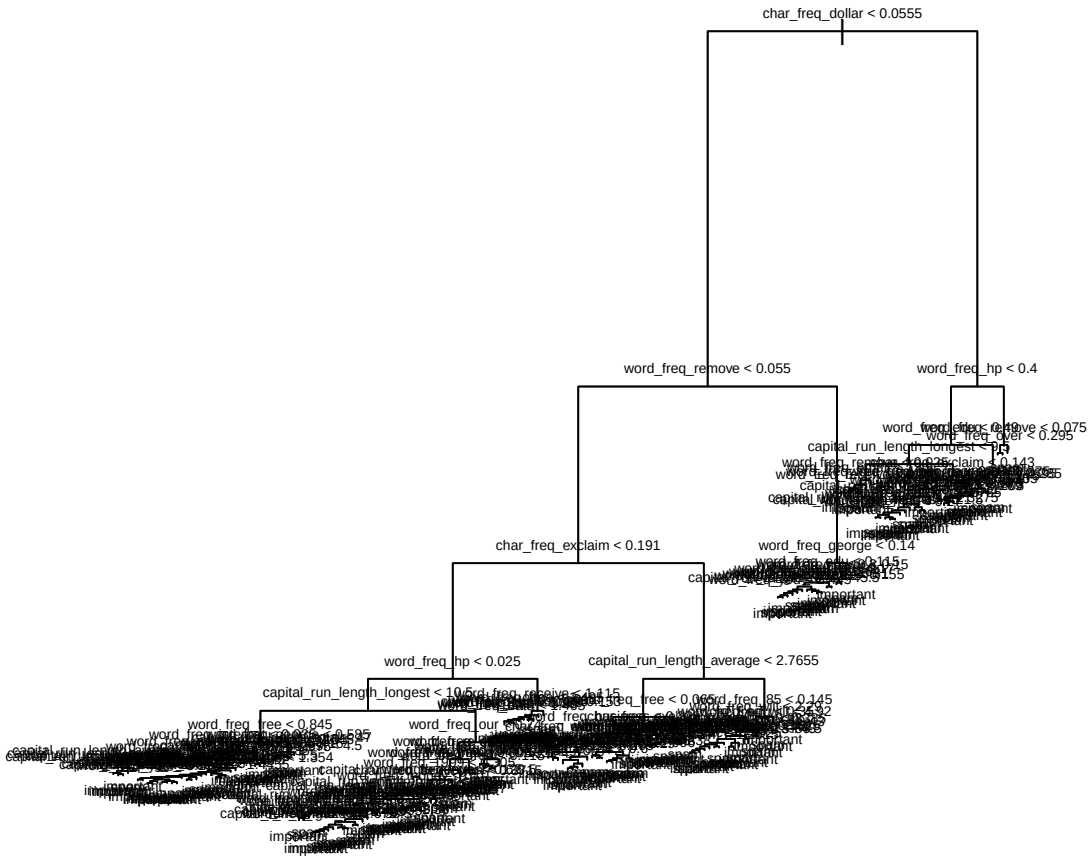
```
##
## pred        important spam
##   important       282   26
##   spam             20  172
```

```
sum(diag(2) * table(pred, email[validation,]$spam))/500
```

```
## [1] 0.908
```

Controlling the Size of the Tree

```
res<-tree(spam~., data=email[-validation],mincut=2,minsize=5,mindev=0.0005)

plot(res)
text(res, cex=0.5)
```



Controlling the Size of the Tree

The size of the tree can be chosen by a range of arguments
`tree.control( mincut = 5, minsize = 10, mindev = 0.01)`

mincut The minimum number of observations to include in either child node. This is a weighted quantity; the observational weights are used to compute the number. The default is 5.

minsize The smallest allowed node size: a weighted quantity. The default is 10.

mindev The within-node deviance must be at least this times that of the root node for the node to be split.

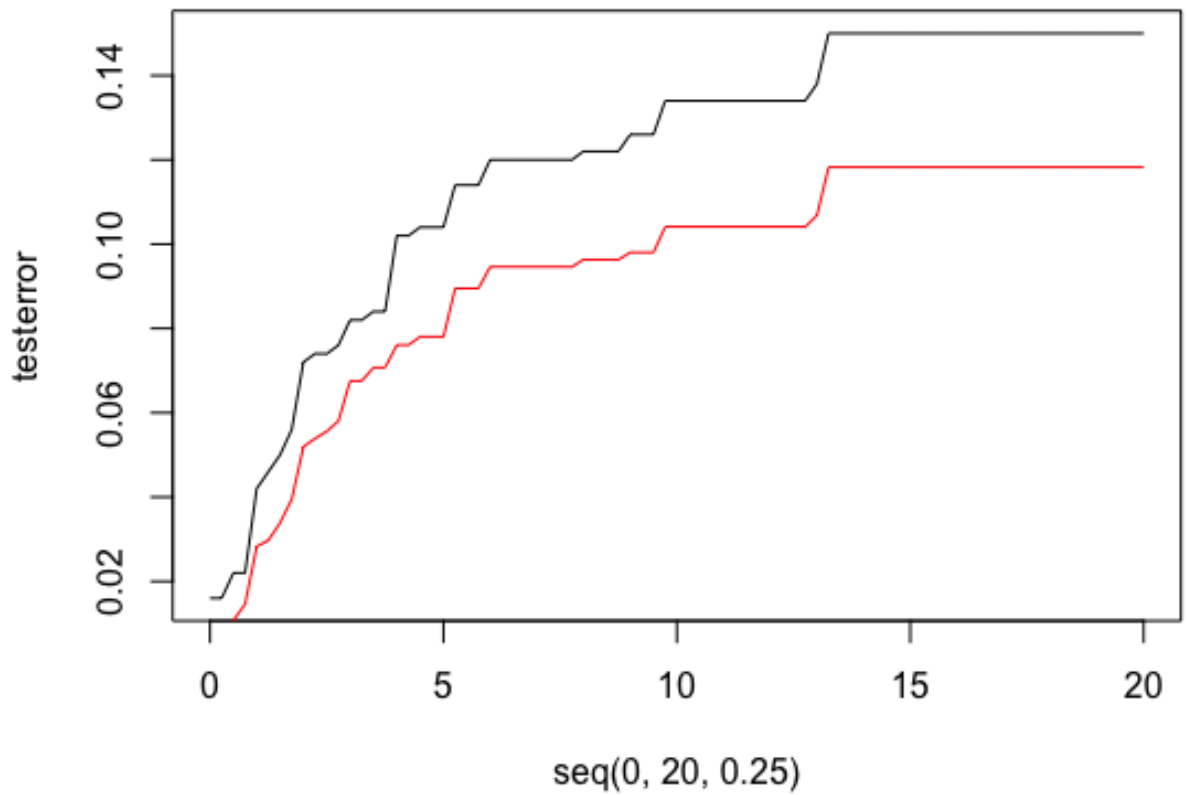
To produce a tree that fits the (training) data perfectly, set `mindev = 0` and `minsize = 1`.

Bias- vs Variance Trade-Off

```
#superbushy
res<-tree(spam~., data=email[-validation],minsize=2,mindev=0.0001)
testerror<-NULL
trainerror<-NULL
for(a in seq(0,20,0.25)) {
#ONLY PLOT SOME AND NOT AH
prune.k<-prune.tree(res, newdata=email[-validation,], method="misclass", k=a)

pred<-predict(prune.k, email[validation,], type="class")

testerror<-c(testerror,1-sum(diag(2) * table(pred, email[validation,]$spam))/500)
pred<-predict(prune.k, email[-validation,], type="class")

trainerror<-c(trainerror,1-sum(diag(2) * table(pred, email[-validation,]$spam))/4101)

}
```

Bias- vs Variance Trade-Off

```
plot(seq(0,20,0.25),testerror, type="l")
lines(seq(0,20,0.25),trainerror, col="red")
```



Plan

Classification Trees

Bagging and Random Forests

Aggregation for Variance Reduction

- General result: for some observations $Z_1, ..., Z_n$ that are drawn *independently* and identically from a distribution with the same variance, you know that

$$Var(1/n \sum_{i=1}^{n} Z_i) = \frac{\sigma^2}{n}$$

- Note this only holds for Z_i, Z_j being independent draws.
- Regression Trees/ Classification Trees tend to have high variance and low bias.
- We can reduce the variance of statistical learning methods, by taking many (independently drawn) training sets from the population, build separate classification model and then average the results.
- This is the core idea of bootstrapping, which is particularly useful for trees and we study it in this context.

Bagged Trees

- Suppose you calculate $\hat{f}^1(x), ..., \hat{f}^B(x)$ from B different randomly selected training sets, your prediction is given as

$$\hat{f}_{avg}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x)$$

- We sample from our set of observatins n with replacement, so some observations may randomly appear multiple times.
- We compute B regression trees using B bootstrapped training sets, and average the resulting predictions.
- *No need to prune* these trees. An individual tree has high variance, but low bias.
- Averaging across these B trees *reduces the variance*.
- Intuition: Variance in prediction due to individual trees fitting (non existing) patterns in the error term ϵ. Each individual bootstrapped tree fits a different pattern, which washes out as we average.

Bagged Trees and Bias Variance Tradeoff

$$E(f(x_0) - f(\hat{x_0}))^2 = Var(\hat{f}(x_0)) + [Bias(\hat{f}(x_0))]^2 + Var(\epsilon)$$

- Variance in prediction due to individual trees fitting (non existing) patterns in the error term ϵ, that is $Var(\hat{f}(x_0))$ captures $Var(\epsilon)$
- Each individual bootstrapped tree fits a different pattern in the noise, which washes out as we average.
- There is a second source of variance in $Var(\hat{f}(x_0))$, coming from the fact that, mechanically, the predictions across bootstrapped samples are correlated.
- Random forests deals with this issue in a clever way by *decorrelate* individual $\hat{f}^b$.

Random Forests

- The problem: our different bootstrapped $\hat{f}^b$'s are correlated mechanically, since they are obtained using the same set of observations. This increases the variance of our prediction.
- Correlation arises from using the same set of X's in the construction of each bootstrapped tree.
- One way to avoid this is to try to *decorrelate* individual $\hat{f}^b$'s
- To avoid this, when building these trees, 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.
- Usually $m \approx \sqrt{p}$ is chosen.
- This mechanically decorrelates the fitted values.

Random Forests

- The problem: our different bootstrapped $\hat{f}^b$'s are correlated mechanically, since they are obtained using the same set of observations. This increases the variance of our prediction.
- Correlation arises from using the same set of X's in the construction of each bootstrapped tree.
- One way to avoid this is to try to *decorrelate* individual $\hat{f}^b$'s
- To avoid this, when building these trees, 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.
- Usually $m \approx \sqrt{p}$ is chosen.
- This mechanically decorrelates the fitted values.

Random Forest Illustration

```
library(randomForest)
```

```
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package:  'randomForest'
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed(12022017)
dim(email)
```

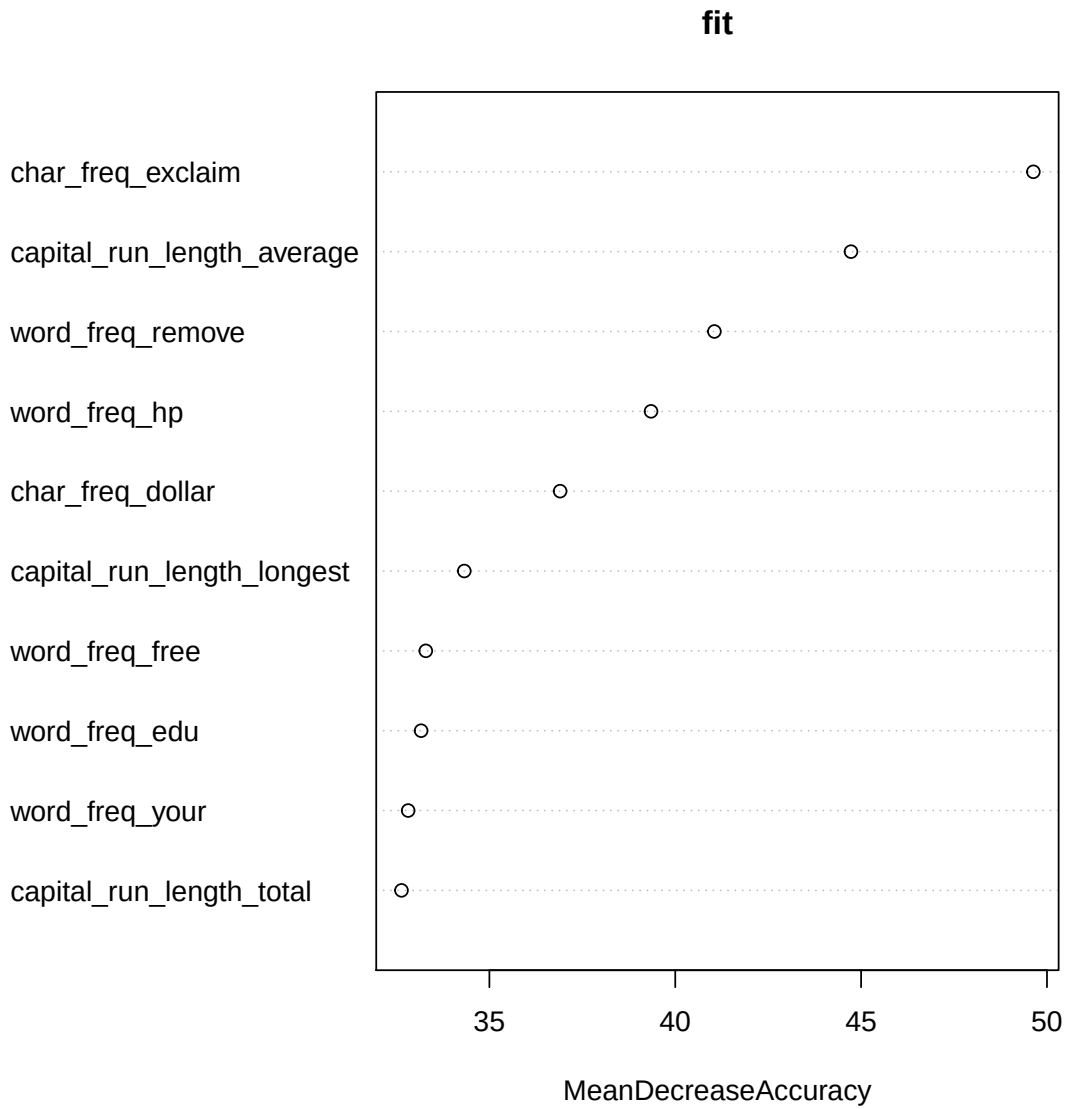
```
## [1] 4601   58
```

```
fit <- randomForest(as.factor(spam) ~ .,
                    data=email[-validation,],
                    importance=TRUE,
                    ntree=500)
fit
```

```
##
## Call:
##  randomForest(formula = as.factor(spam) ~ ., data = email[-validation,      ], importance = TRUE, ntree = 5
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 4.58%
## Confusion matrix:
##           important spam class.error
## important      2416   70      0.0282
## spam            118 1497      0.0731
```

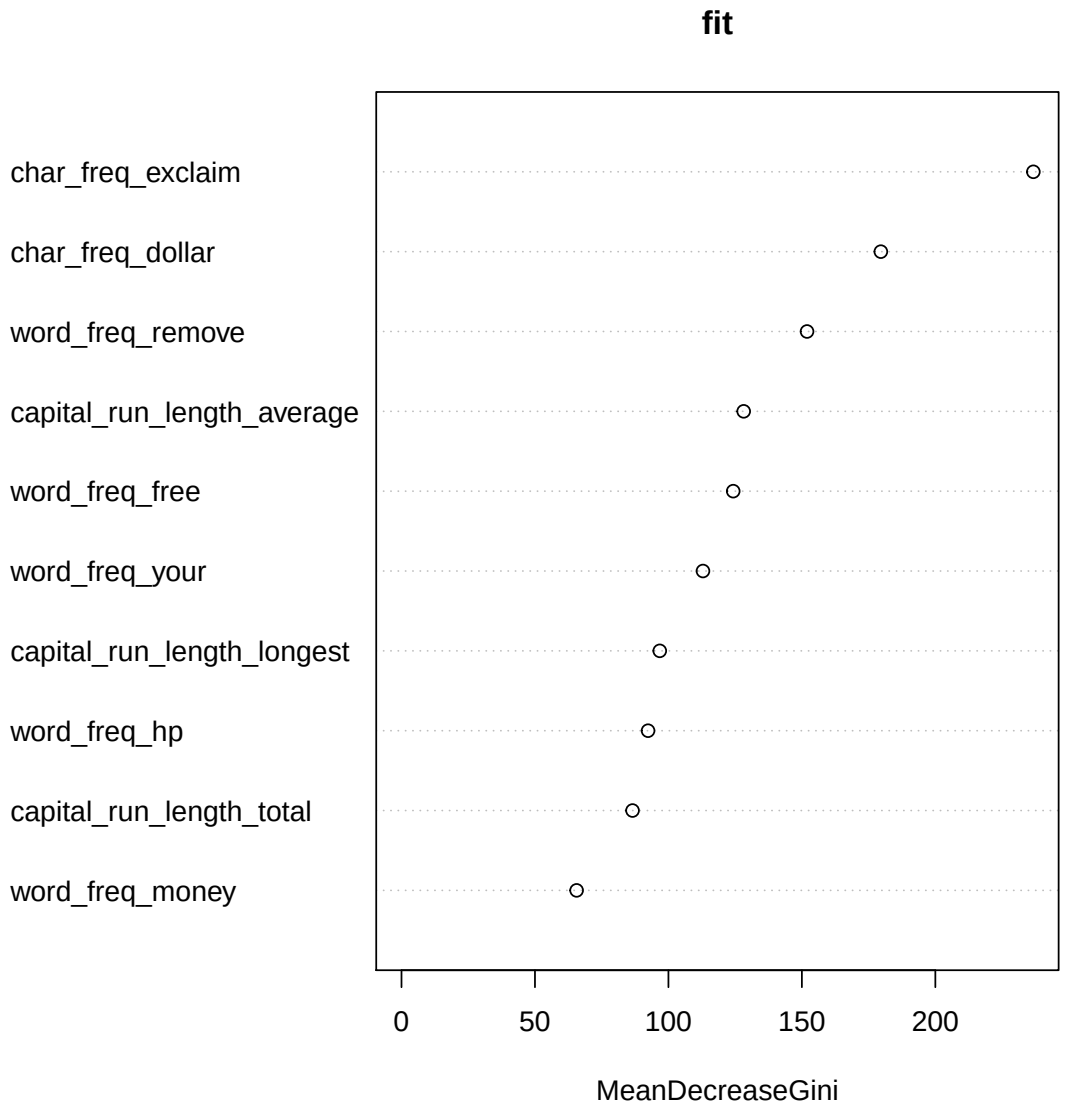
Random Forest Variable Importance

```r
##five most important variables
varImpPlot(fit, sort=TRUE, n.var=10, type=1)
```



Random Forest Variable Importance (2)

```
##five most important variables
varImpPlot(fit, sort=TRUE, n.var=10, type=2)
```



Random Forest Confusion Matrix

```
pred<-predict(fit, newdata = email[validation,], type="class")
table(email[validation,]$spam,pred)
```

```
##            pred
##             important spam
##   important       292   10
##   spam             12  186
```

Random Forest / Bagging Example (2)

```
TRAINING<-data.table(read.csv(file="/Users/thiemo/Dropbox/Research/Matteo and Thiemo/senna/classification-tree
PERSON<-TRAINING[objecttype=="person" & label1!=""]
PERSON$label1 <- factor(PERSON$label1)
PERSON$label1num <- as.numeric(factor(PERSON$label1))
library(tm)
library(RTextTools)
set.seed(30012017)
#a validation set
valid<-sample(1:nrow(PERSON), 500)
PERSON$validation<- 0
PERSON[valid]$validation<-1
PERSON<-PERSON[order(validation)]
DOC<-create_matrix(c(PERSON[,paste(objectcleanpp,sep=" ")]),language="english",
                   removeNumbers=TRUE,stemWords=TRUE,removePunctuation=TRUE,removeSparseTerms=0.999)
DOCCONT<-create_container(DOC,PERSON$label1num, trainSize=1:1200,
                          testSize=1201:nrow(PERSON), virgin=TRUE)
MOD <- train_models(DOCCONT, algorithms=c("MAXENT","TREE","BOOSTING","BAGGING","RF"))
RES <- classify_models(DOCCONT, MOD)
```

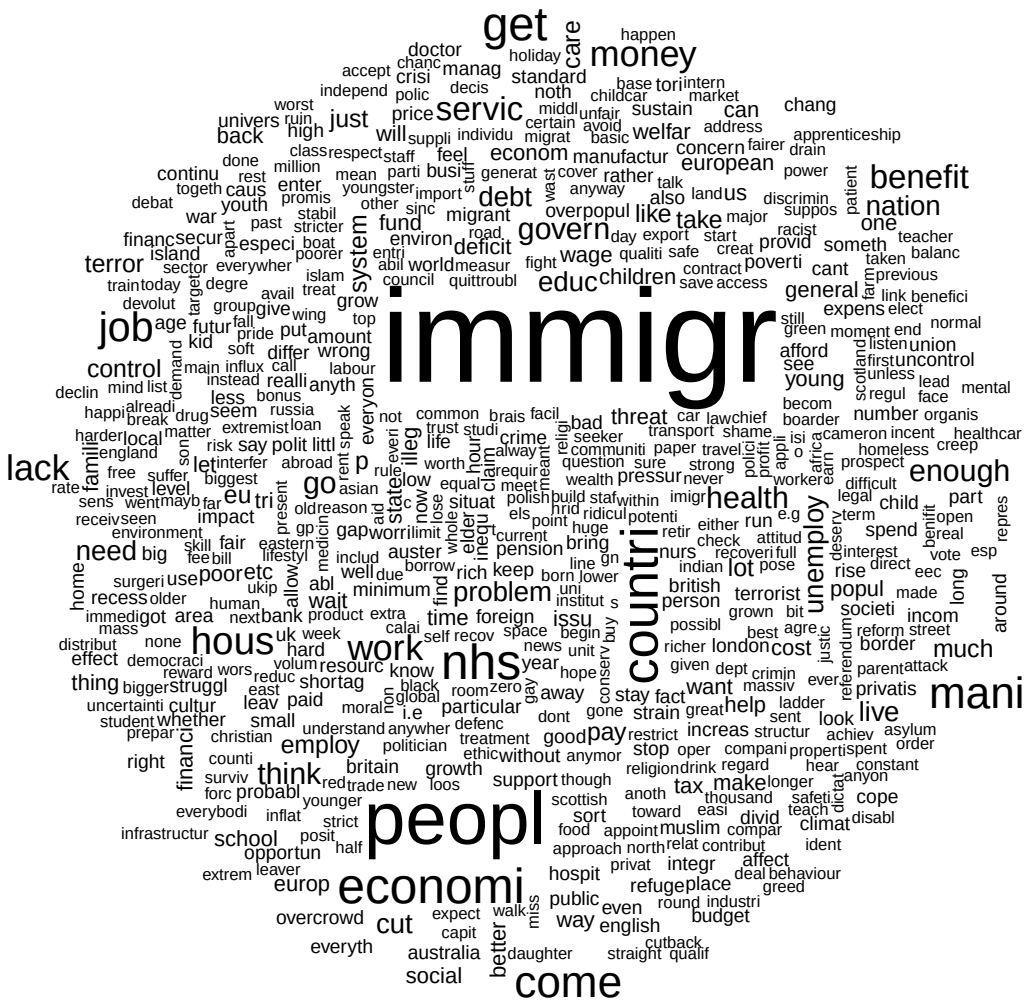
Random Forest / Bagging Example (2)

```
analytics <- create_analytics(DOCCONT, RES)
res<-data.table(analytics@document_summary)
VALID<-cbind(PERSON[validation==1],res)

#confusion matrix
sum(diag(3) *table(VALID$MAXENTROPY_LABEL,VALID$label1))/500
## [1] 0.896
sum(diag(3) *table(VALID$TREE_LABEL,VALID$label1))/500
## [1] 0.876
sum(diag(3) *table(VALID$BAGGING_LABEL,VALID$label1))/500
## [1] 0.904
sum(diag(3) *table(VALID$FORESTS_LABEL,VALID$label1))/500
## [1] 0.912
```

Tree's applications for Classification

- We see that trees by themselves may yield relatively poor performance.
- Extensions such as Bagging or Random Forests, produce superior classification performance.
- Yet, many perceive the models as too "black boxy" and generally, tree based methods are said to have high variance.
- Sometimes, simpler models may be preferred to more complex ones.
- In case you can fit multiple models to a training data, it is worth exploring the extent to which they yield similar results.
- For word features, the fact that Trees explicitly model interactions is very useful as even for unigram word features, we can explicitly model negations etc.

Going back to the Brexit example



Going back to the Brexit example

```
##just look at one word feature
library(tm)
library(RTextTools)
set.seed(17052017)
#a validation set
DOC<-create_matrix(DTA$A1,language="english",
                   removeNumbers=TRUE,stemWords=TRUE,removePunctuation=TRUE, removeSparseTerms = 0.999)
DOCCONT<-create_container(DOC,as.numeric(DTA$leaveeu), trainSize=1:1869,
                          testSize=1870:nrow(DTA), virgin=TRUE)
MOD <- train_models(DOCCONT, algorithms=c("TREE","BOOSTING","BAGGING","RF"))
RES <- classify_models(DOCCONT, MOD)

analytics <- create_analytics(DOCCONT, RES)
res<-data.table(analytics@document_summary)
VALID<-cbind(DTA[1870:nrow(DTA)],res)

#confusion matrix
table(VALID$TREE_LABEL,VALID$leaveeu)/200
```

```
##
##     leave remain
##   1 0.225  0.125
##   2 0.240  0.410
```

```
table(VALID$BAGGING_LABEL,VALID$leaveeu)/200
```

```
##
##     leave remain
##   1 0.205  0.130
##   2 0.260  0.405
```

```
table(VALID$FORESTS_LABEL,VALID$leaveeu)/200
```

```
##
##     leave remain
##   1 0.250  0.170
##   2 0.215  0.365
```

Course Projects

- In total, I received 14 submissions covering 29 students out of a class of 31
- Group size so average group size is 2
- I will aim to combine a couple groups due to similar coverage.

Course Projects

```
##just look at one word feature
library(quanteda)
library(readtext)
CORP<-readtext("/Users/thiemo/Dropbox/Teaching/QTA/Homeworks/Projects/txt")
CORP<-corpus(CORP)

summary(CORP)
```

```
## Corpus consisting of 14 documents.
##
##        Text Types Tokens Sentences
##   Team1.txt   289    707        27
##  Team10.txt   329    750        34
##  Team11.txt   415   1143        79
##  Team12.txt   208    474        27
##  Team13.txt   293    802        36
##  Team14.txt   142    286        16
##   Team2.txt   329    733        33
##   Team3.txt   251    637        18
##   Team4.txt   202    367        25
##   Team5.txt   275    571        21
##   Team6.txt   220    392        27
##   Team7.txt   205    395        15
##   Team8.txt   355    693        24
##   Team9.txt   358    883        32
##
## Source:  /Users/thiemo/Dropbox/Teaching/QTA/Lectures/Week 7/* on x86_64 by thiemo
## Created: Thu May 18 15:38:51 2017
## Notes:
```

Course Projects