Class 7: Decision Trees and Random Forests

BUS 696

Prof. Jonathan Hersh

Class 7: Announcements

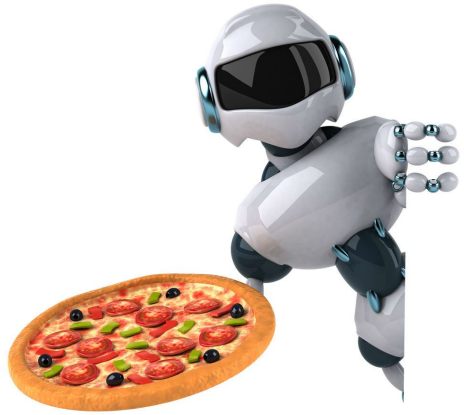
1. Will post Problem Set 3, Due Oct 21

Class 7: Outline

1. AI + Pizza
2. Regression Trees
3. Regression Trees in R
4. Cross-Validating to Determine Optimal Tree Depth
5. Regression Tree Lab
6. Bagging
7. Random Forests
8. Random Forest Lab

Dominos: AI + Pizza

How AI helps Domino's predict when 3 billion pizzas are ready to go



by Zachary Fragoso, Manager, Data Science and AI, Domino's

Zachary Fragoso is a manager of data science and AI at Domino's in Ann Arbor, Michigan.

AI is ready for the enterprise. I know because I'm part of a team at Domino's that's delivering business results with the technology today.

I won't claim it's easy, but I do know success is out there for people ready to do the work needed to master this emerging approach to computing. I hope our experiences can inspire others to experiment with AI in their business, and figure out what works best for them.

Baking algorithms for better predictions

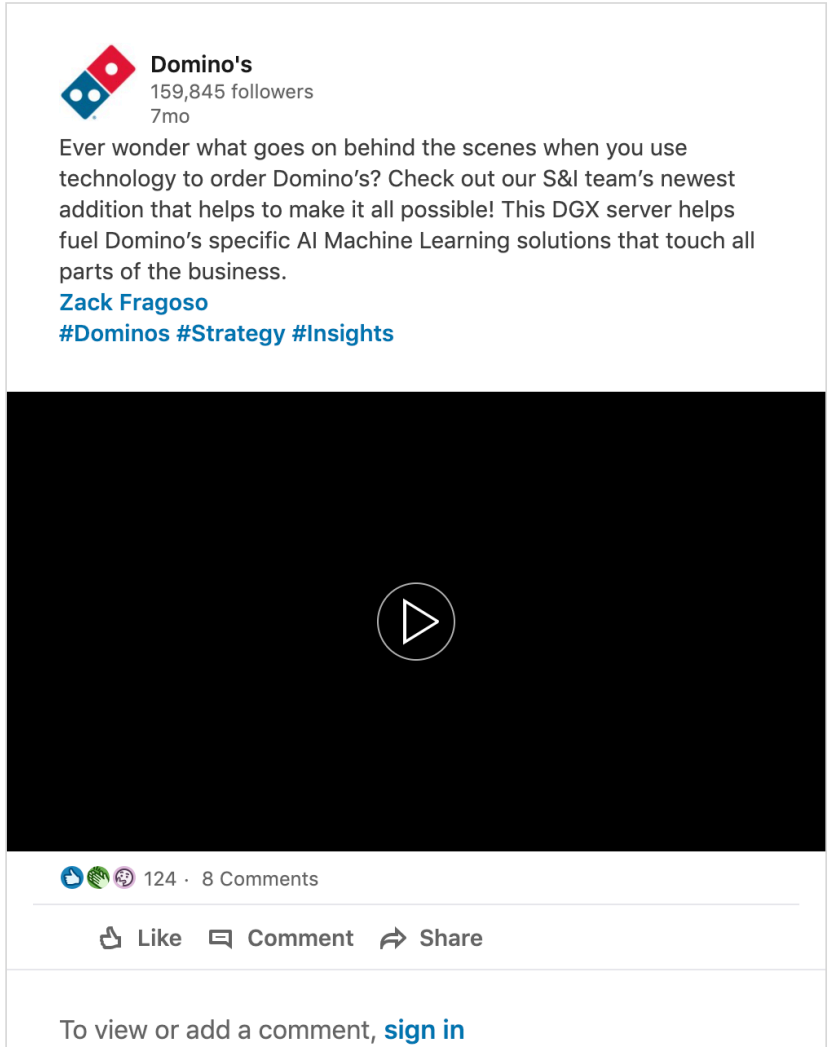
Making quick decisions is important when you need to deliver more than 3 billion pizzas a year — fast. So, Domino's is exploring the use of AI for a host of applications, including more accurately predicting when an order will be ready.

We recently boosted accuracy from 75% to 95% for predictions of an order's readiness. We used what we call a load-time model that factors in labor variables, order complexity and other operational factors.

The improvement has been well received and could be the basis for future ways to advance operator efficiencies and customer experiences, thanks in part to NVIDIA GPUs.

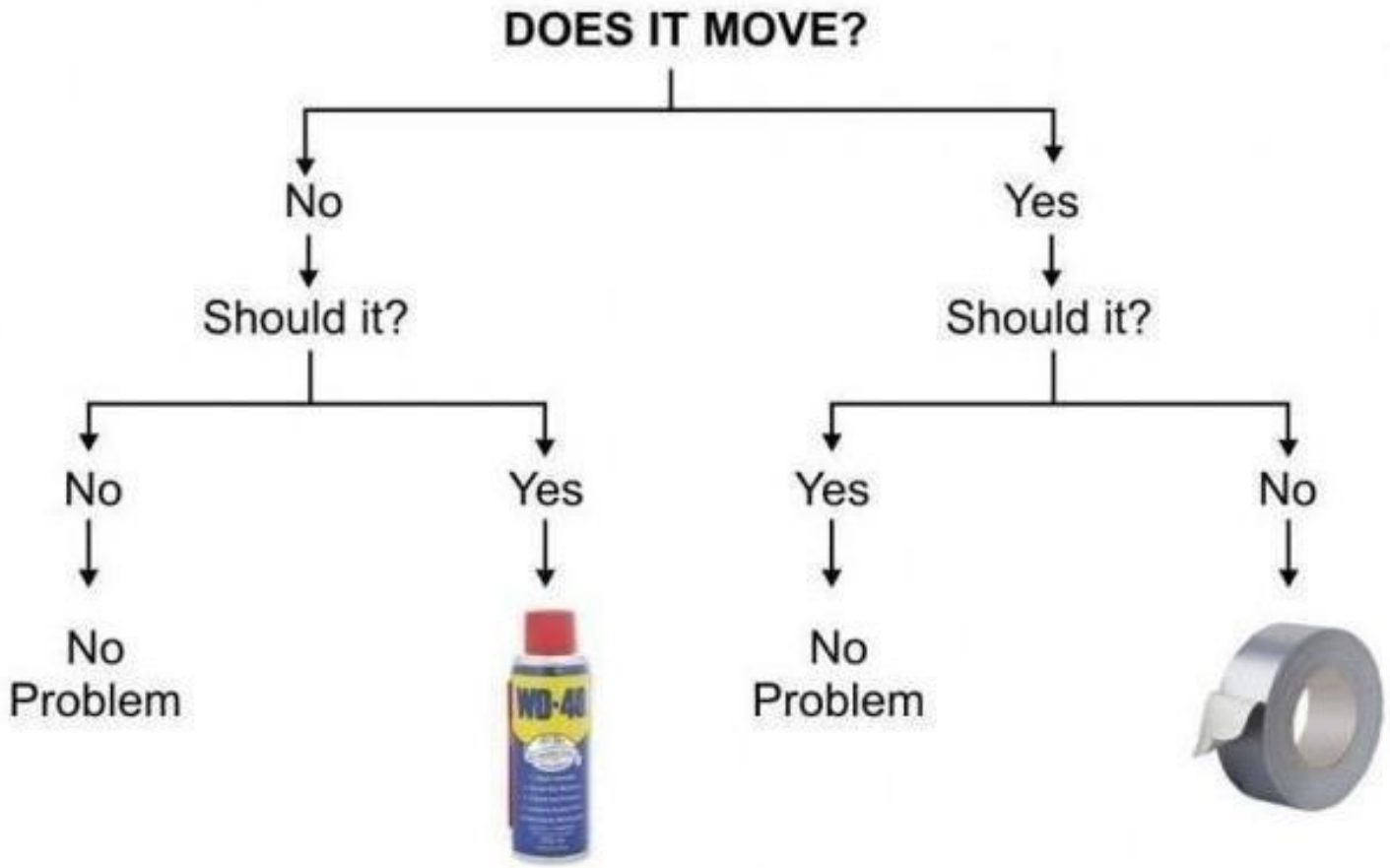
https://techcrunch.com/sponsor/nvidia/how-ai-helps-dominos-predict-when-3-billion-pizzas-are-ready-to-go/

NVIDIA + Dominos Video



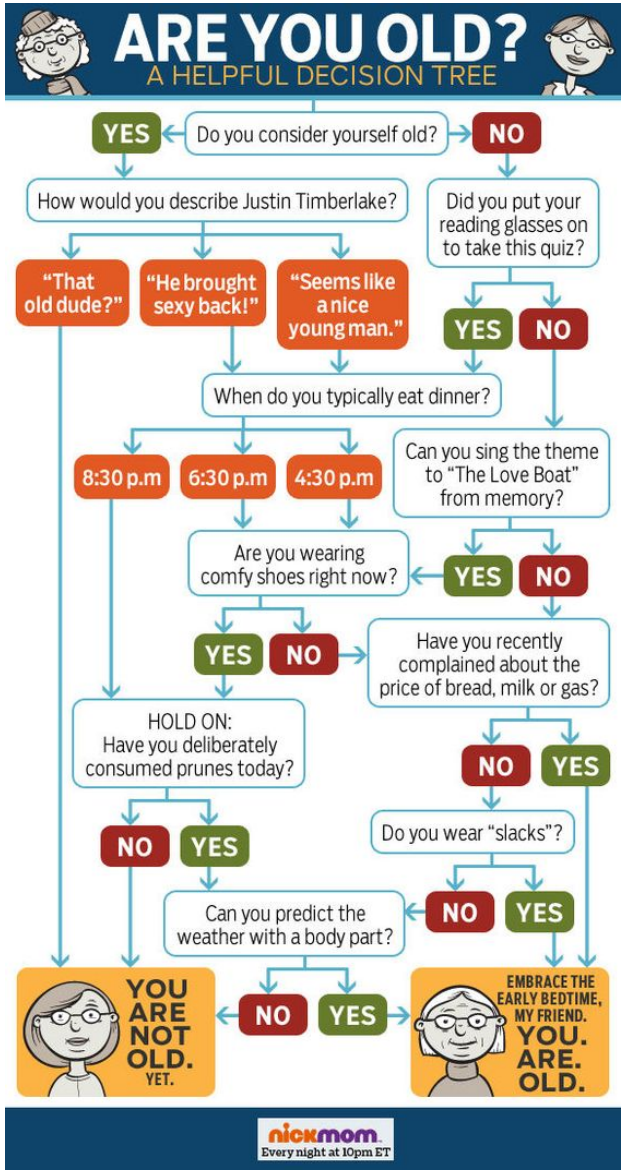
https://news.developer.nvidia.com/dominos-takes-delivery-to-a-whole-new-level-with-nvidia-dgx-1/

What Are Binary Decision Rules?



- Binary decision rules are any rules with only two options!

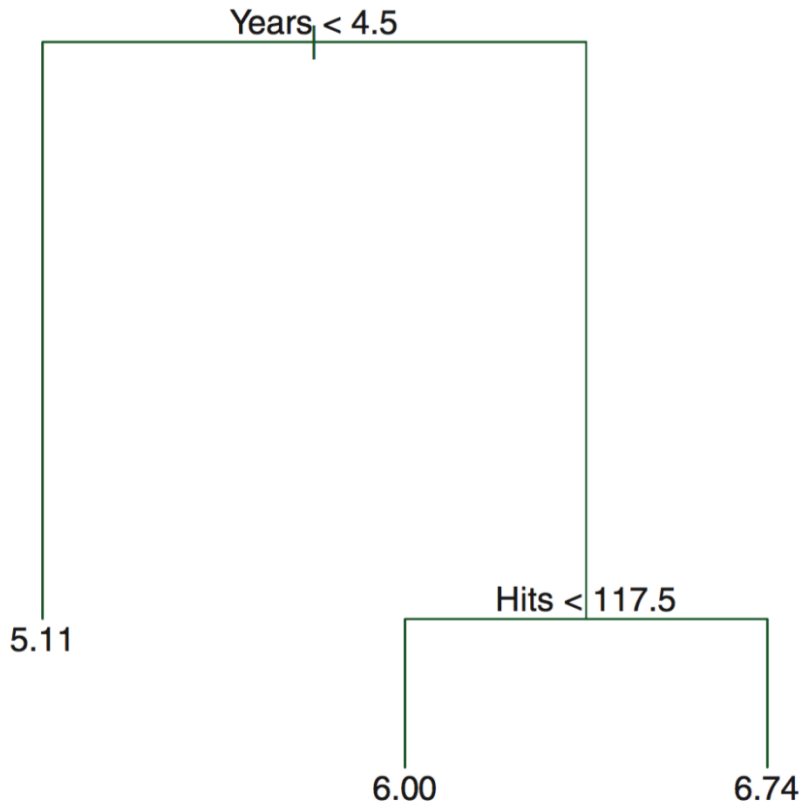
Classification Trees: Series of Binary Decision



- Combining these binary decisions into a "tree" creates a decision tree
- Either classification ($y \in \{0,1\}$) or regression ($y \in [-\infty, \infty]$) problems can be modeled using a decision tree

Regression Trees

- **Tree based methods *stratify* or *segment* the predictor space into different regions**
- Regions are stratified via simple rules
- The splitting rules can be summarized into a tree that is very intuitive



Pros and Cons of Trees

Pros

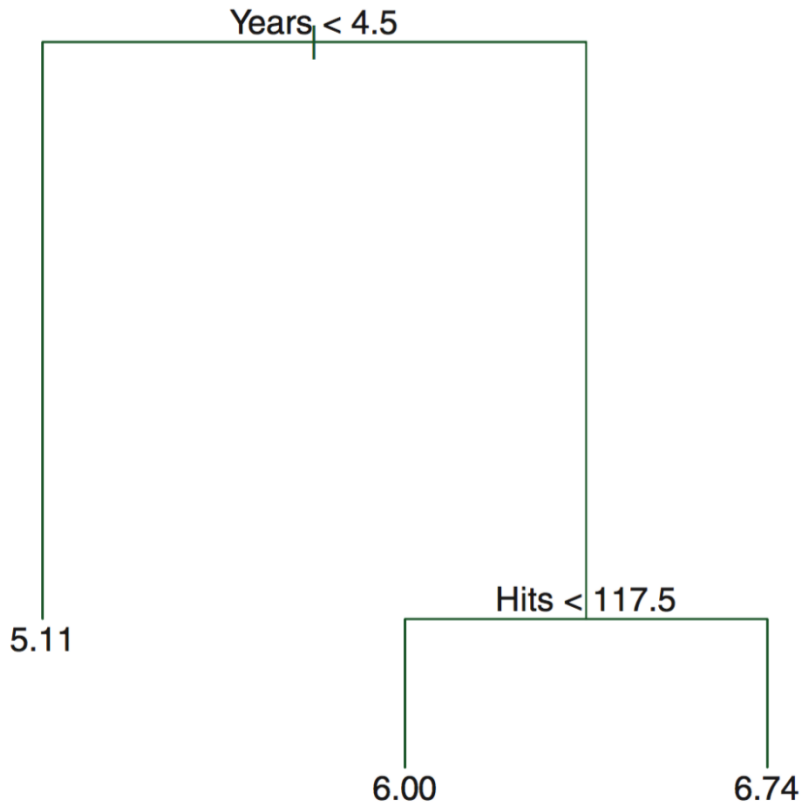
- Simple
- Easy to interpret
- Easy to explain
- Can be displayed graphically!
- Bagging, boosting, and random forests very powerful (combining trees)

Cons

- Slow with large datasets
- Not easy to use “out of the box”
- Choice of split can be unstable

Decision/Regression Trees

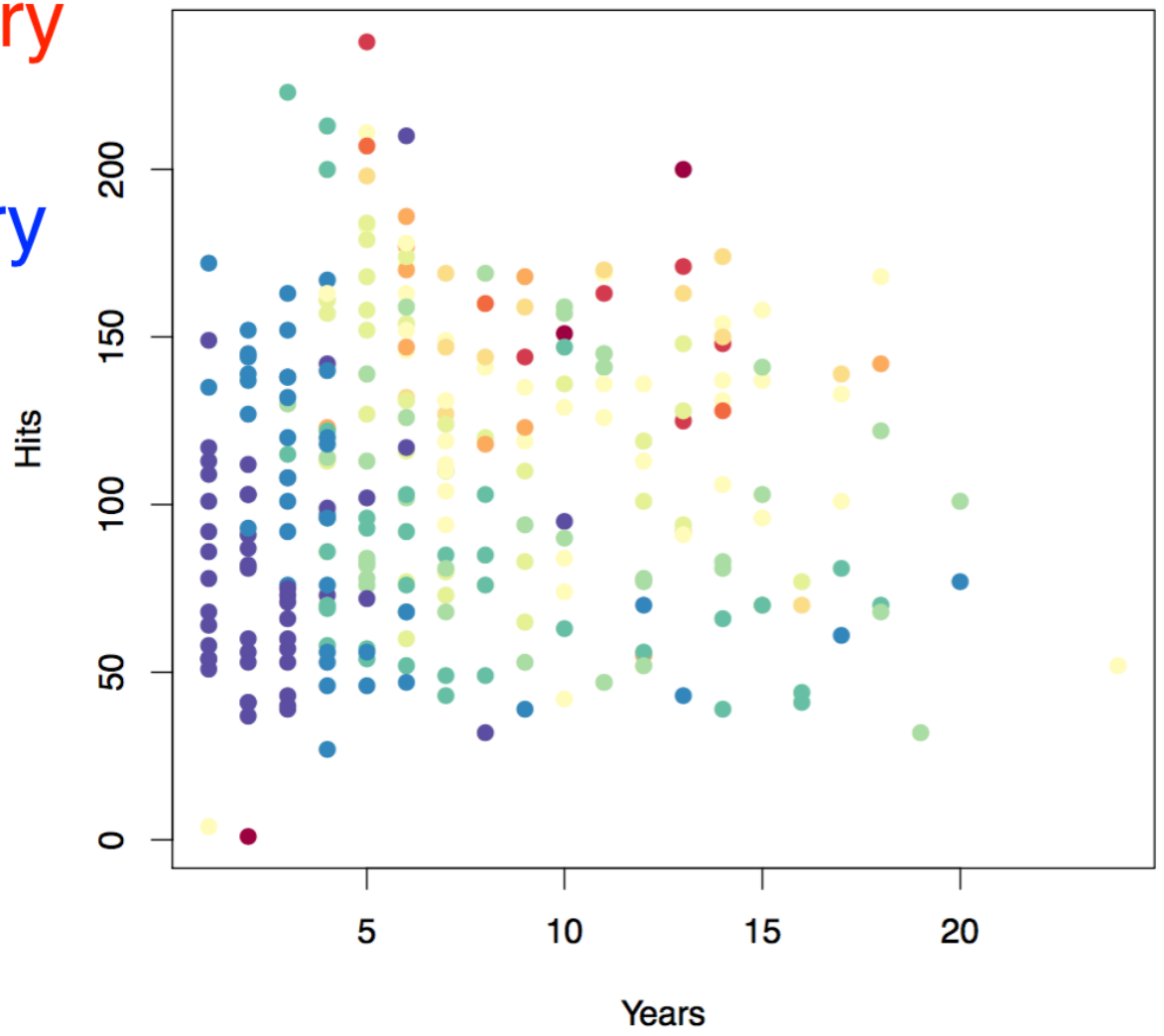
- Decision trees can be applied to both regression problems ($y_i \in R$) and classification problems $y_i \in \{class1, class2, \dots, \}$
- We'll consider both



Baseball salary data: how to partition/stratify?

High salary
red

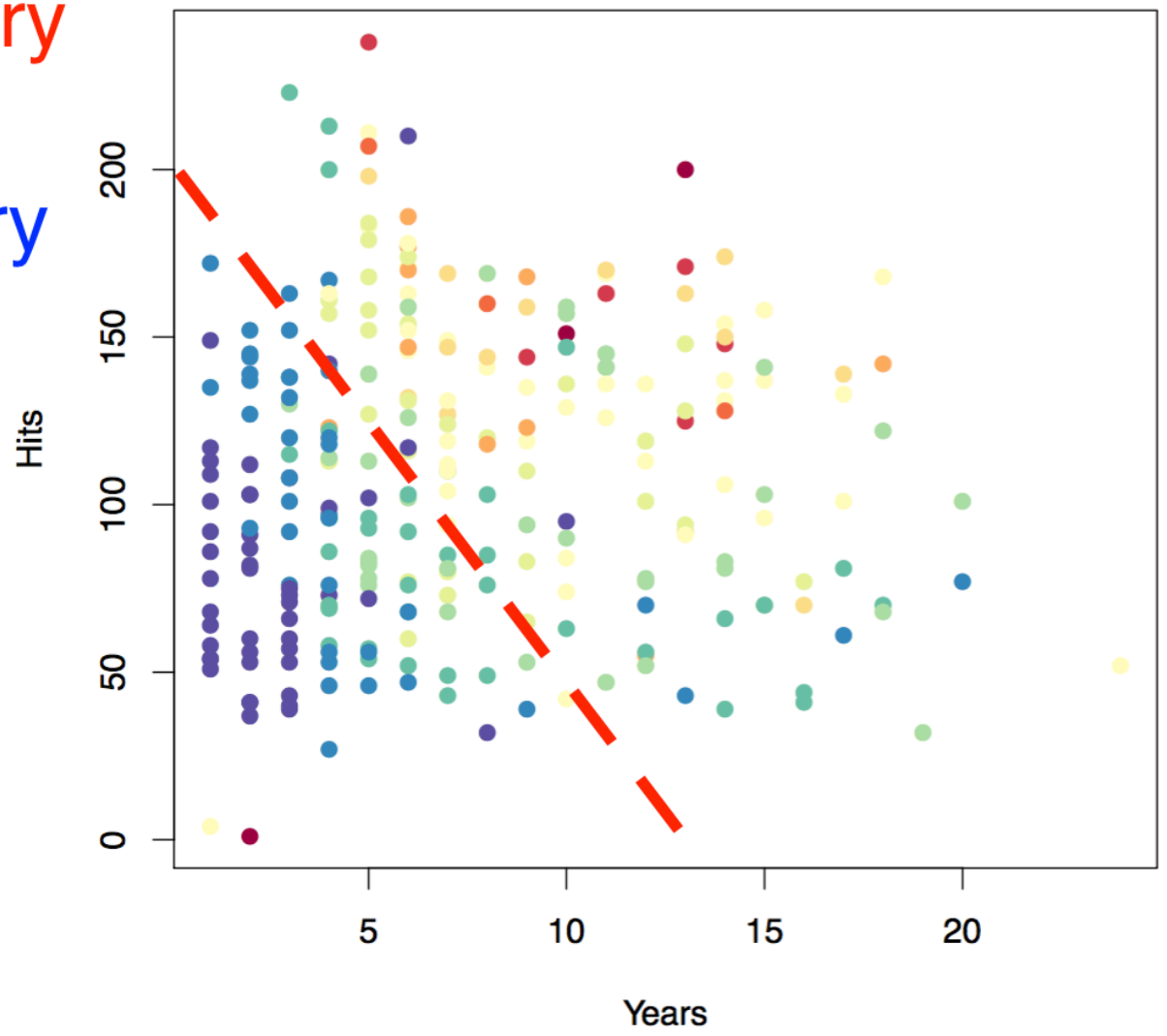
Low salary
blue



Baseball salary data: how to partition/stratify?

High salary
red

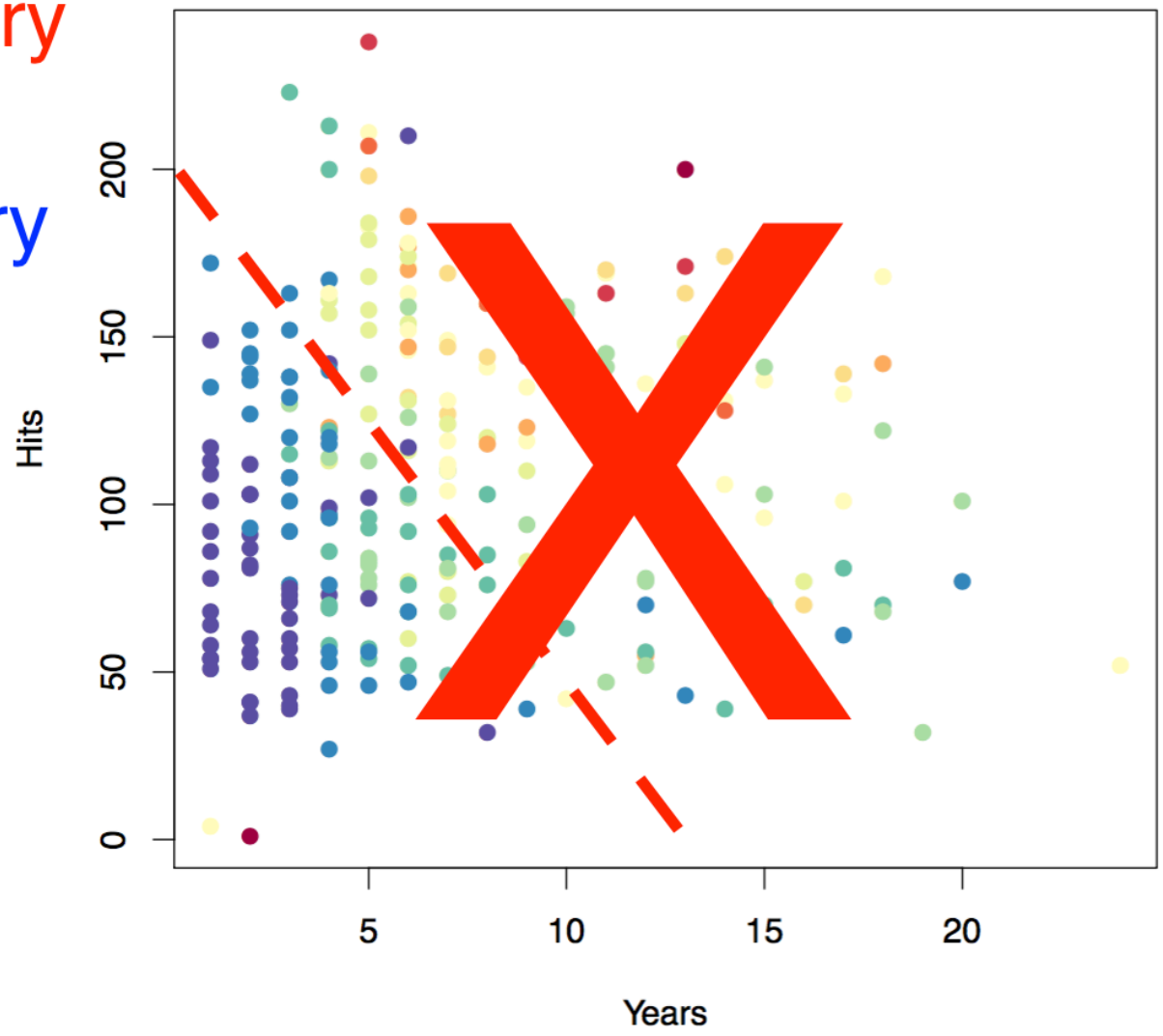
Low salary
blue



Baseball salary data: how to partition/stratify?

High salary
red

Low salary
blue

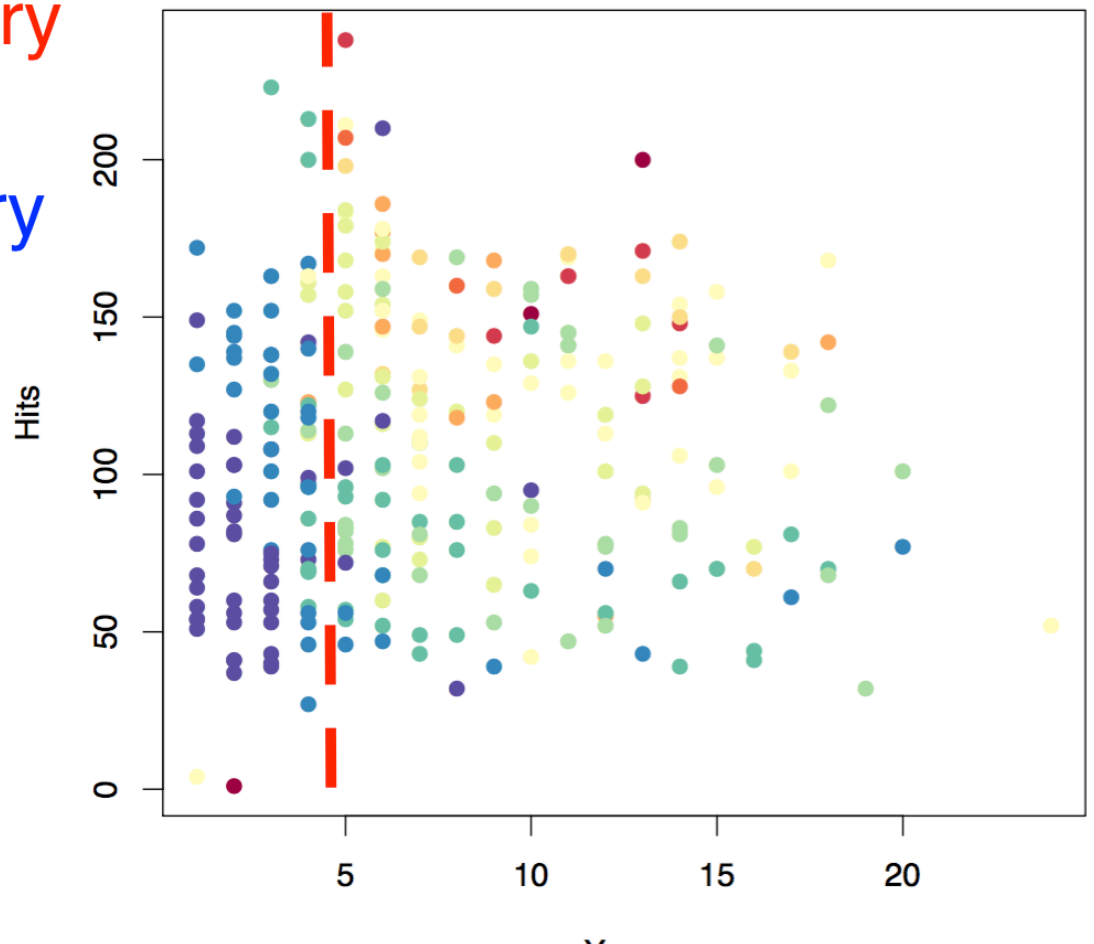


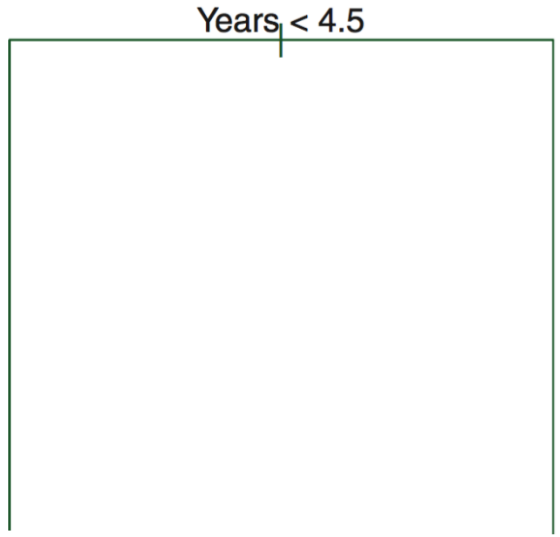
- Only linear classification rules are allowed, e.g. year > 10

How Regression Trees are Constructed

High salary
red

Low salary
blue



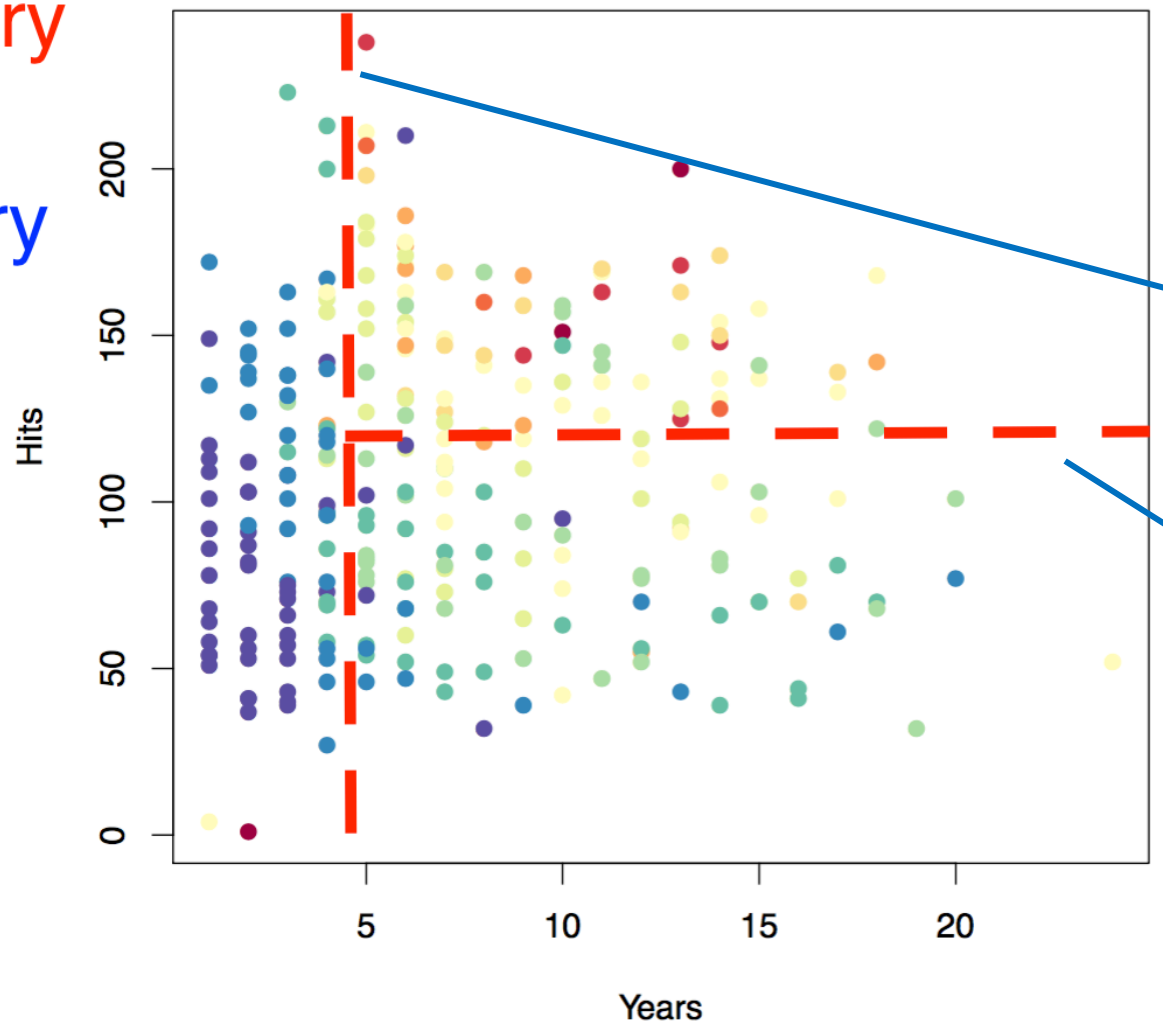


- Every variable and every split is consider.
- Chosen split is one which maximizes separation of high and low salaries:
- Split 1: years > 4.5

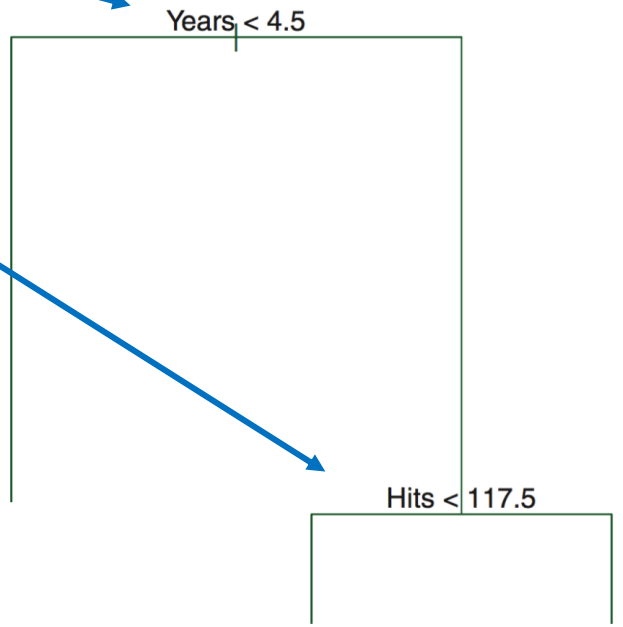
Baseball salary data: split 2

High salary red

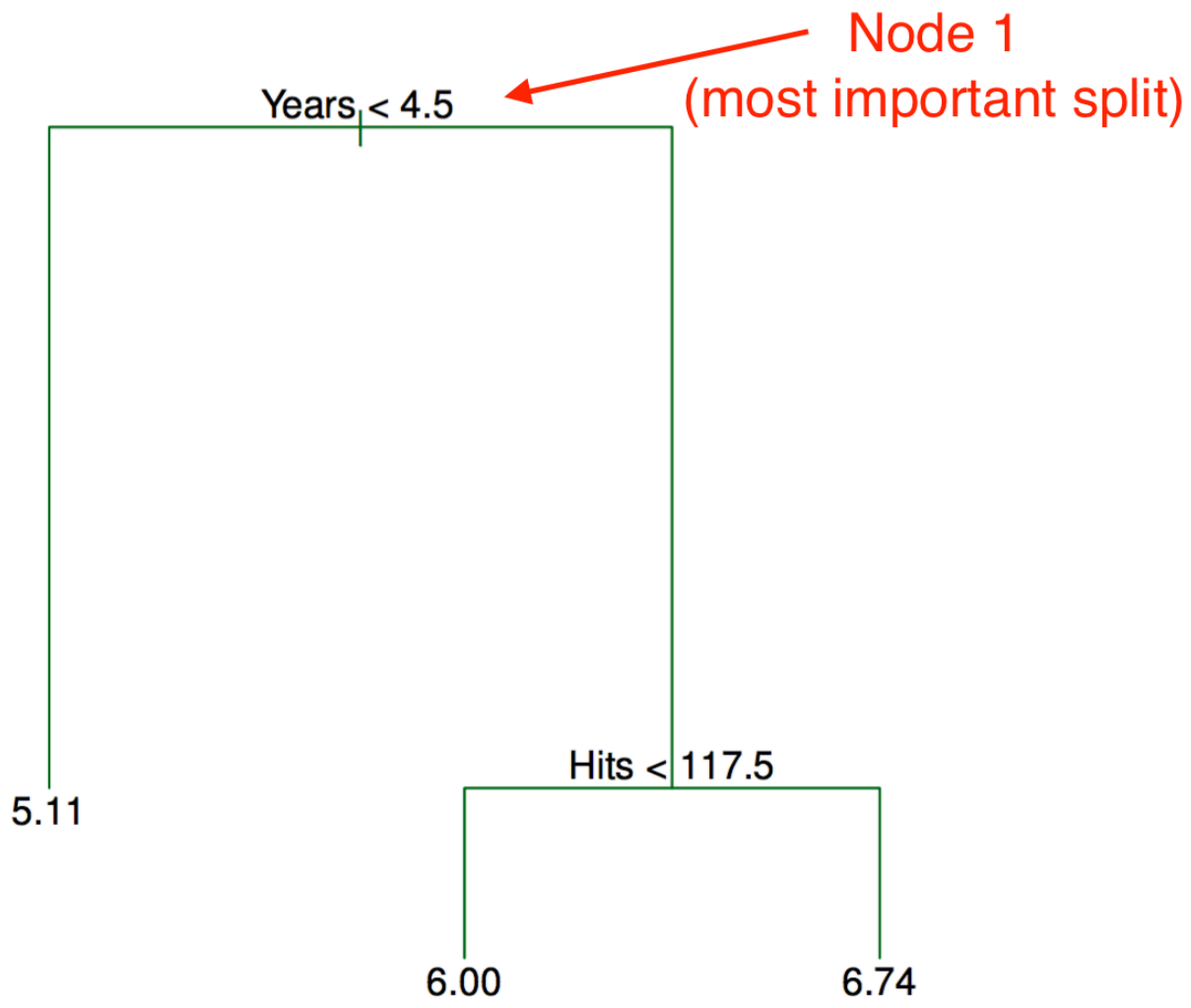
Low salary blue



- Split 1: years > 4.5
- Split 2: hits > 117.5

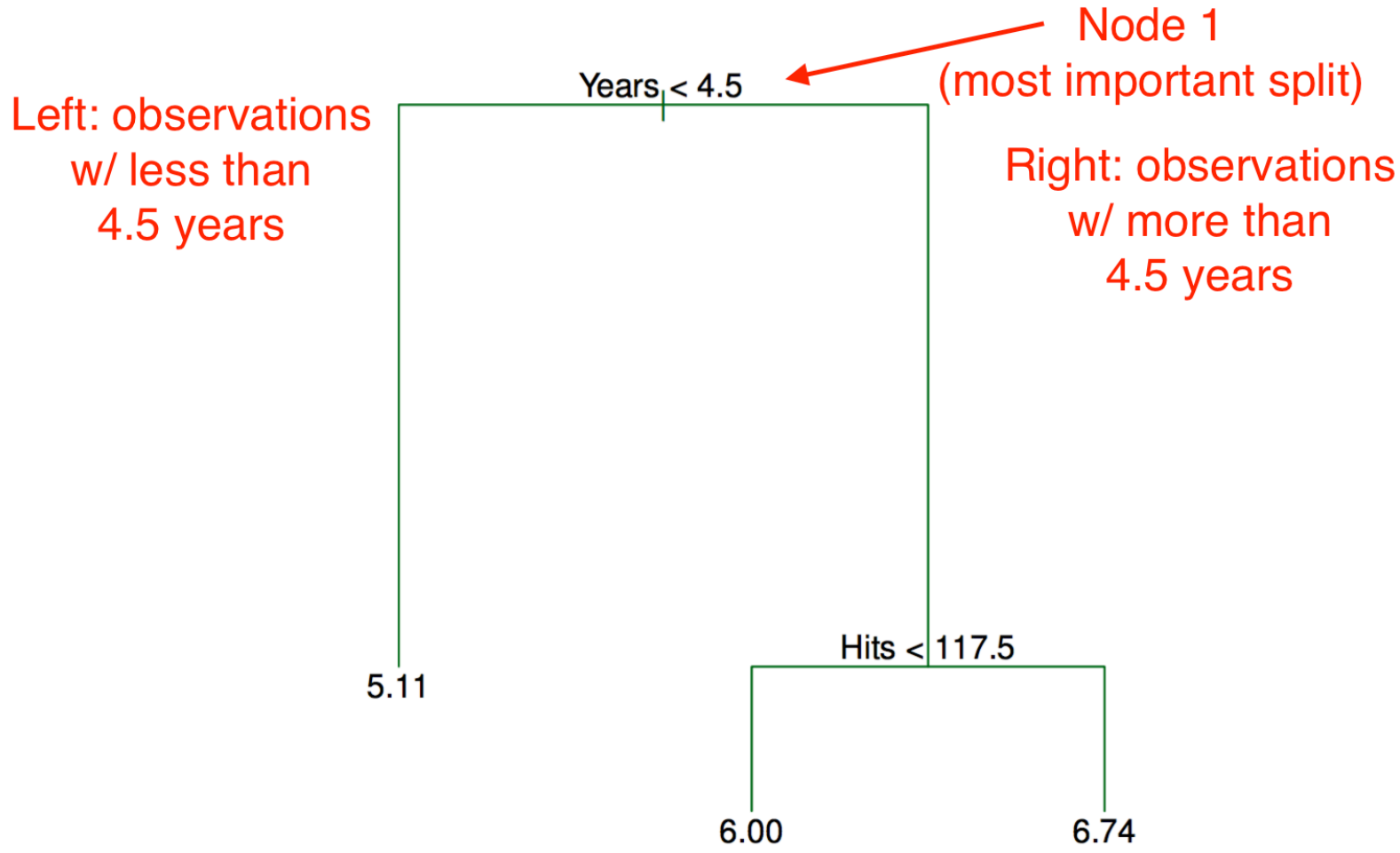


Tree Representation

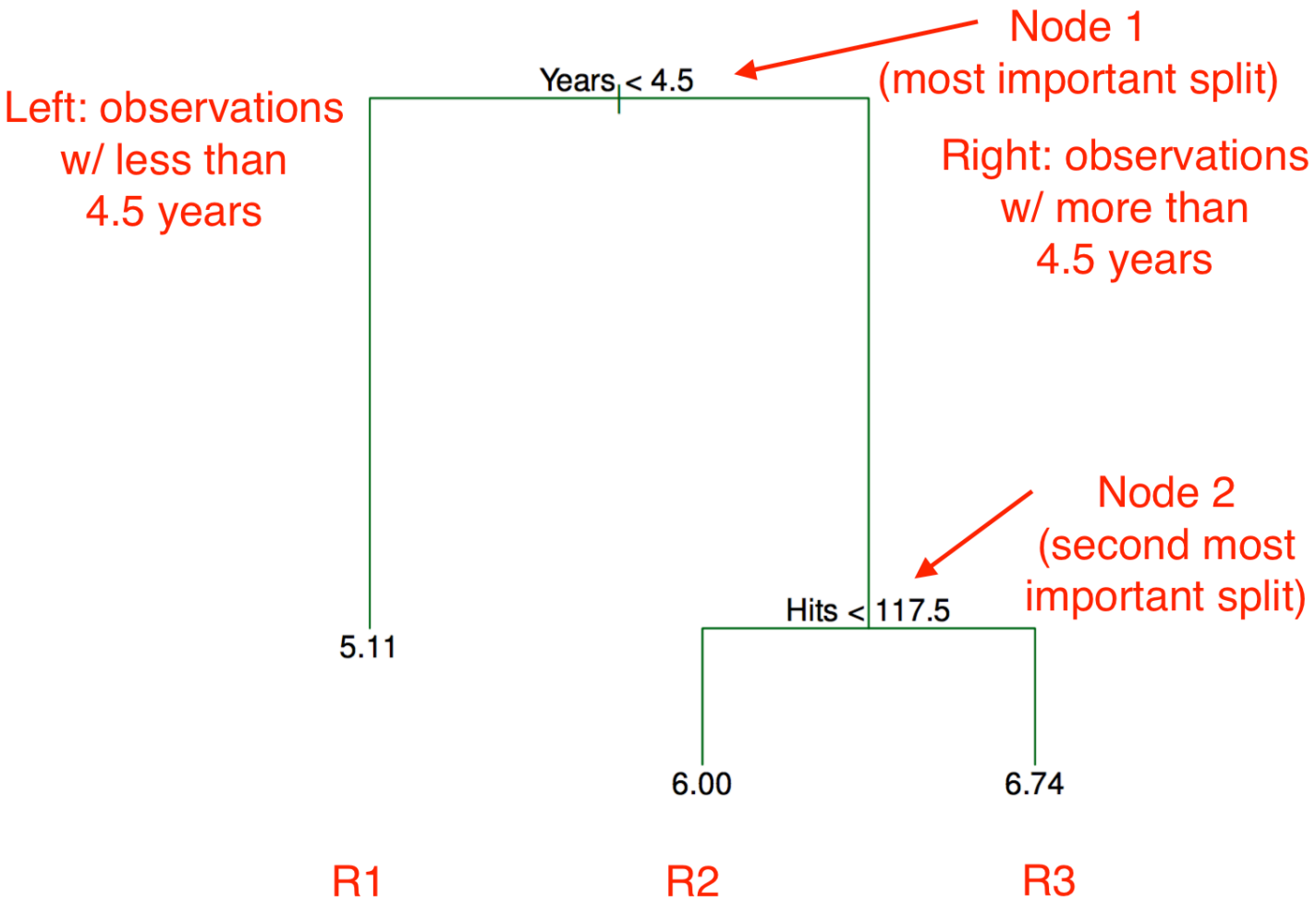


- Trees are read top-down
- Most important split is at top
- Length represents how much within-cluster variance decreases from split
- So Years explains more variance than Hits for this tree

Tree Representation

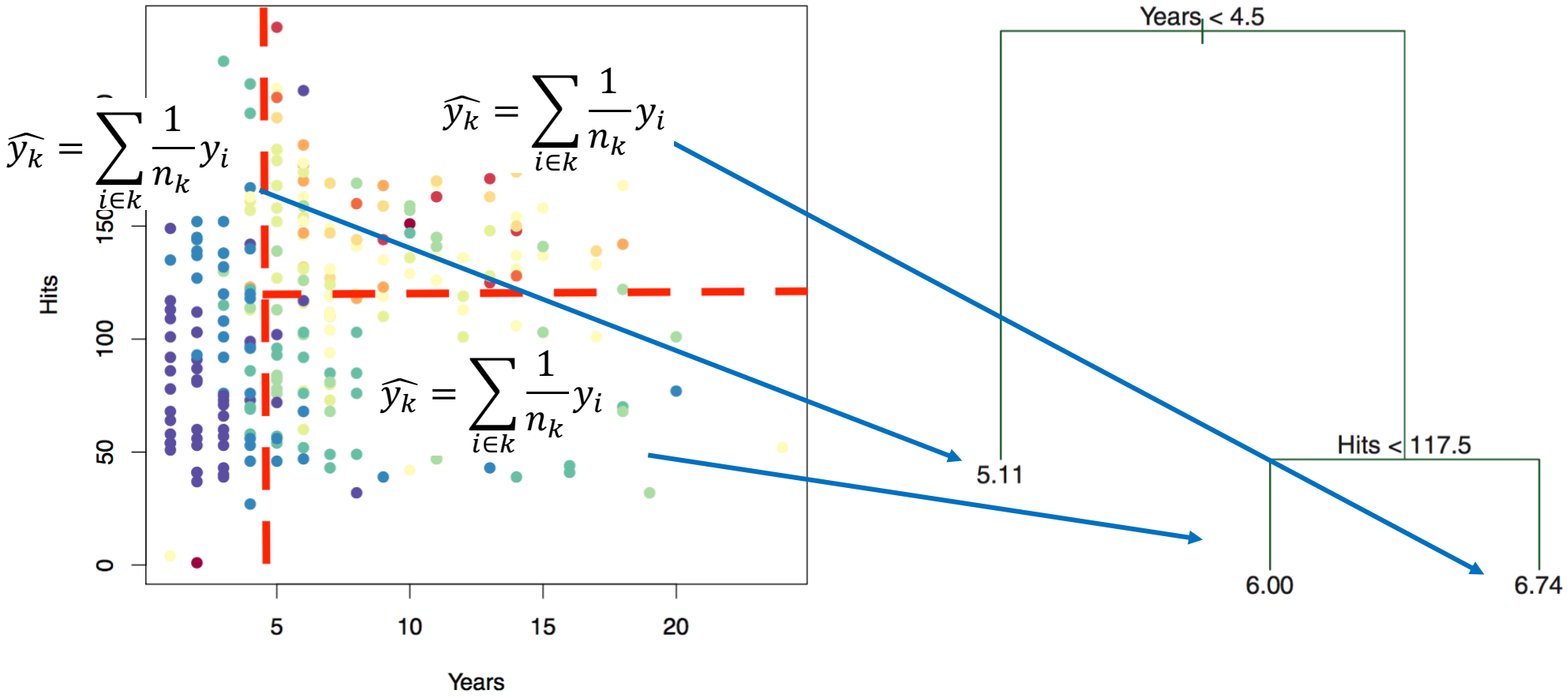


Predictions for Trees? "Leaf" Values



- At the end of the tree are "leafs" (R1, R2, R3)
- How do we predict using a tree? Using the training data we find the average y for all the observations in the leaf $\bar{y}_{leaf} = \frac{1}{n_{leaf}} \sum_{i \in leaf} y_i$
- Any new Xs gets sorted into leafs and assigned the y average for that leaf: $\hat{y}_{i \in leaf} = \bar{y}_{leaf}$

Predictions for Trees? “Leaf” Values



- Leaf predictions are average values in each partition, k

Titanic dataset in ‘titanic’ package

`titanic_train` *Titanic train data.*

Description

Titanic train data.

Usage

```
titanic_train
```

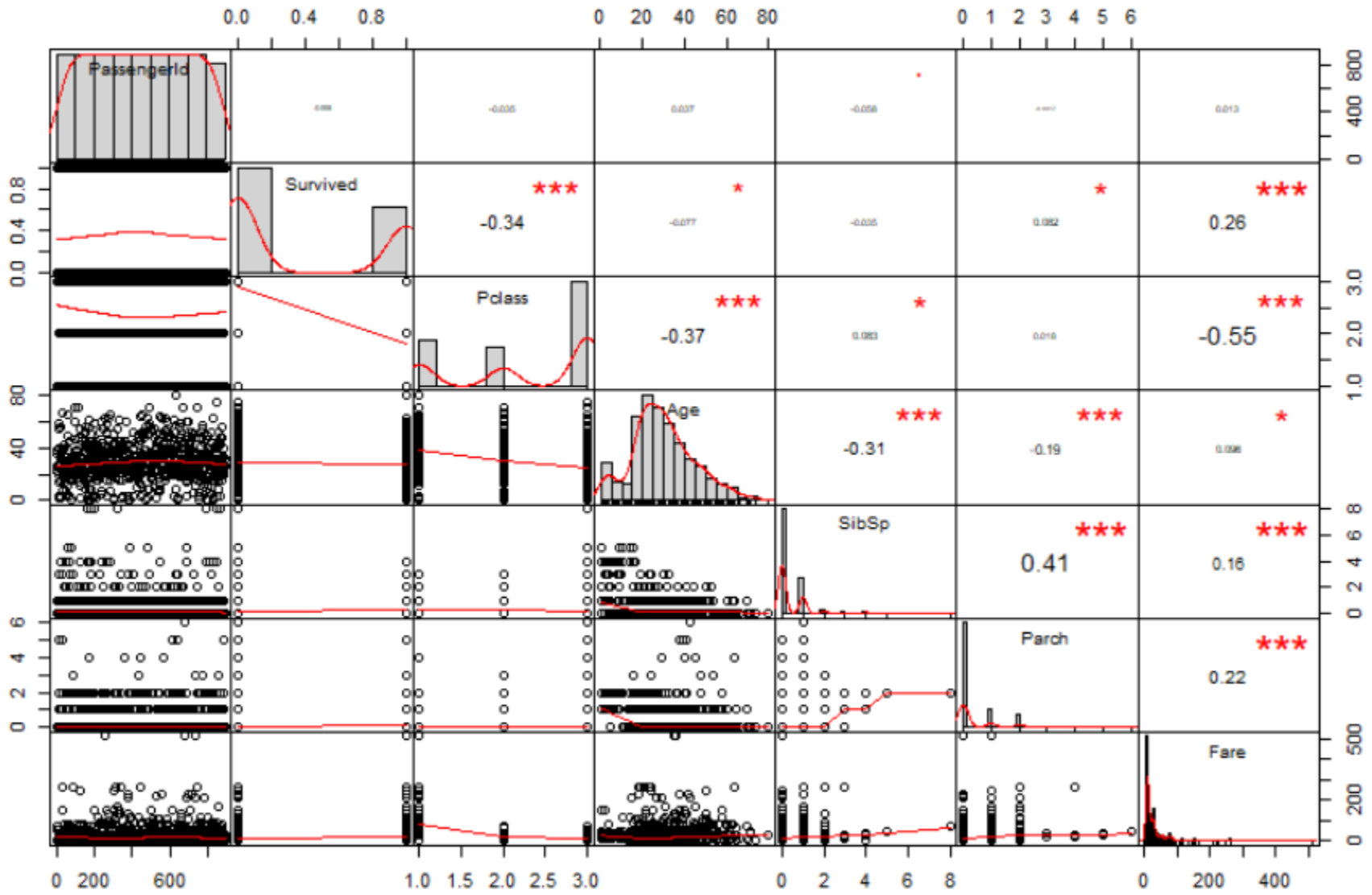
Format

Data frame with columns

PassengerId Passenger ID
Survived Passenger Survival Indicator
Pclass Passenger Class
Name Name
Sex Sex
Age Age
SibSp Number of Siblings/Spouses Aboard
Parch Number of Parents/Children Aboard
Ticket Ticket Number
Fare Passenger Fare
Cabin Cabin
Embarked Port of Embarkation

- To estimate regression trees in R we’re going to use the ‘titanic’ dataset as an example.

Titanic dataset in ‘titanic’ package



ctree() function in package “partykit” to build regression tree

ctree {partykit} R Documentation

Conditional Inference Trees

Description

Recursive partitioning for continuous, censored, ordered, nominal and multivariate response variables in a conditional inference framework.

Usage

```
ctree(formula, data, subset, weights, na.action = na.pass, offset, cluster,
    control = ctree_control(...), ytrafo = NULL,
    converged = NULL, scores = NULL, doFit = TRUE, ...)
```

Arguments

`formula`	a symbolic description of the model to be fit.
`data`	a data frame containing the variables in the model.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`weights`	an optional vector of weights to be used in the fitting process. Only non-negative integer valued weights are allowed.
`offset`	an optional vector of offset values.
`cluster`	an optional factor indicating independent clusters. Highly experimental, use at your own risk.
`na.action`	a function which indicates what should happen when the data contain missing value.
`control`	a list with control parameters, see `ctree_control`.
`ytrafo`	an optional named list of functions to be applied to the response variable(s) before testing their association with the explanatory variables. Note that this transformation is only performed once for the root node and does not take weights into account. Alternatively, `ytrafo` can be a function of `data` and `weights`. In this case, the transformation is computed for every node with corresponding weights. This feature is experimental and the user interface likely to change.
`converged`	an optional function for checking user-defined criteria before splits are implemented. This is not to be used and very likely to change.
`scores`	an optional named list of scores to be attached to ordered factors.
`doFit`	a logical, if `FALSE`, the tree is not fitted.
`...`	arguments passed to `ctree_control`.

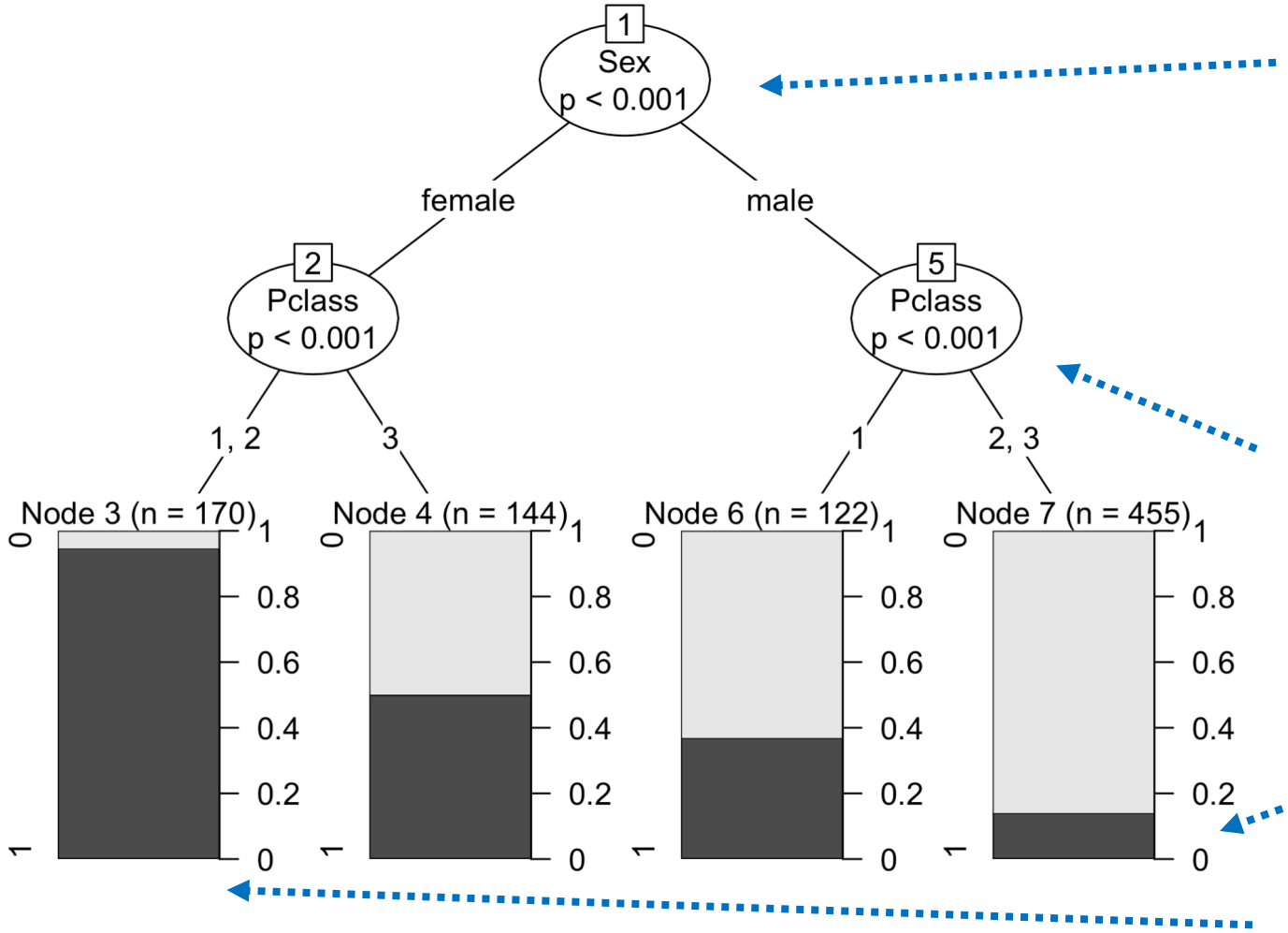
Estimate a tree model to predict titanic survival

```r
# clean data
titanic_df <- titanic_train %>% as_tibble() %>%
  mutate(Suvived = if_else(Survived == 1,
                           "Survived", "Dead"),
         Survived = as.factor(Survived),
         Sex = as.factor(Sex),
         Pclass = as.factor(Pclass)) %>%
         mutate_if(is.character, as.factor)
```

```r
# ctree to estimate model
titanic_tree <- ctree(Survived ~ Sex + Pclass,
                      data = titanic_df)
titanic_tree
```

```
> titanic_tree

Model formula:
Survived ~ Sex + Pclass

Fitted party:
[1] root
|   [2] Sex in female
|   |   [3] Pclass in 1, 2: 1 (n = 170, err = 5.3%)
|   |   [4] Pclass in 3: 0 (n = 144, err = 50.0%)
|   [5] Sex in male
|   |   [6] Pclass in 1: 0 (n = 122, err = 36.9%)
|   |   [7] Pclass in 2, 3: 0 (n = 455, err = 14.1%)

Number of inner nodes:    3
Number of terminal nodes: 4
```

Plot Fitted Regression Tree



- First split is shown at top with p-value with null hypothesis that split doesn't increase class "separation". Sex of passenger is the most important variable, and p-value shows it improves model fit
- Second split is given by successive node. For males, passenger class is second most important variable
- Survival rates are shown in the leaves summaries at the bottom. There are 455 males with passenger class 2 or 3, and about 15% of them survived.
- For females in classes 1 or 2, nearly all survive. Class 3 females have a survival rate of 50%.

Pruning trees

- How do we know when to stop splitting the data?
- **Trees with many splits can overfit the data**
- Solution is to grow a large tree T_0, then prune it to obtain a smaller sub-tree

Baseball example, unpruned tree

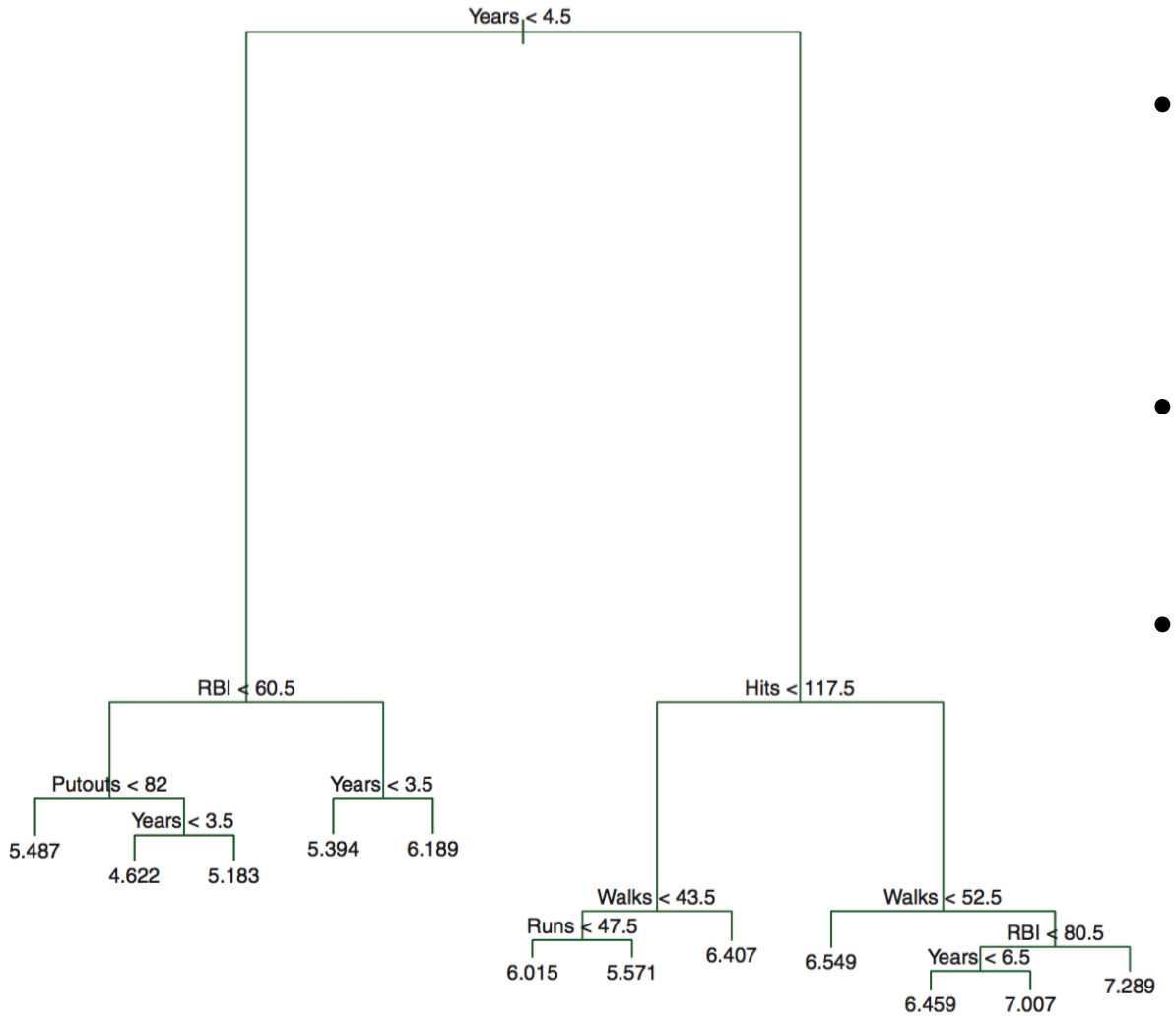
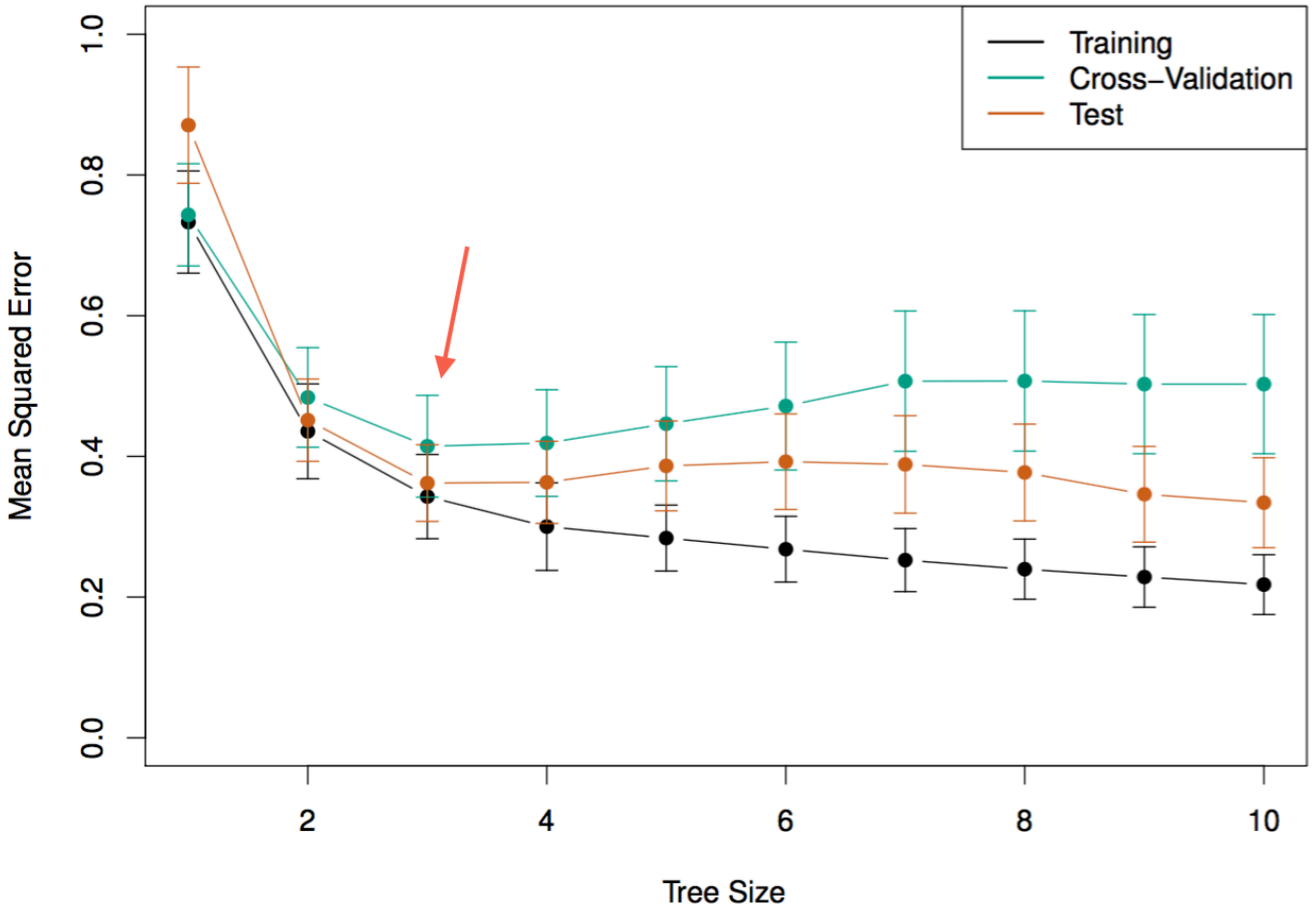


FIGURE 8.4. *Regression tree analysis for the* `Hitters` *data. The unpruned tree that results from top-down greedy splitting on the training data is shown.*

- This tree has too many splits at the bottom and will likely overfit any other data set
- Therefore we must adjust the "depth" of the tree.
- As in prevent the tree from having too many successive splits that only explain the training data and not the test

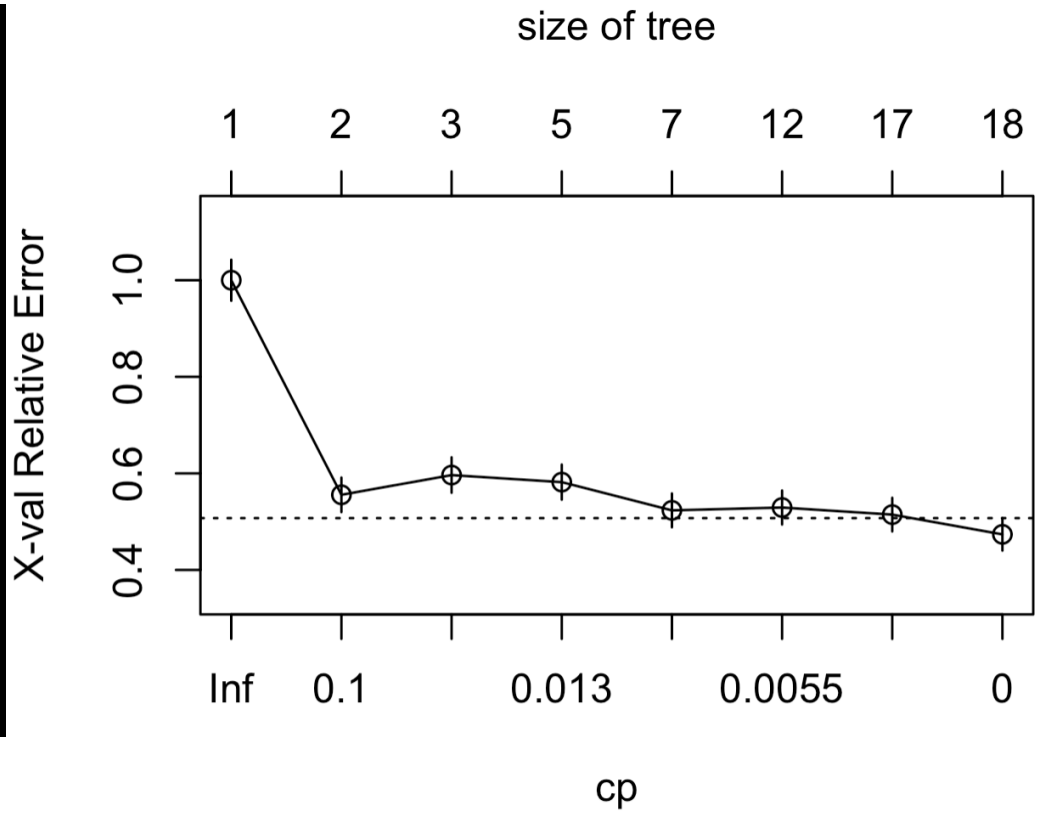
Cross-validate to find alpha



- Complexity in decision trees is controlled by the tree size or depth
- We must cross-validate to select the optimal tree depth, which is here a tree size of depth 3
- Note training error declines with tree depth! (Make sure you know why.)

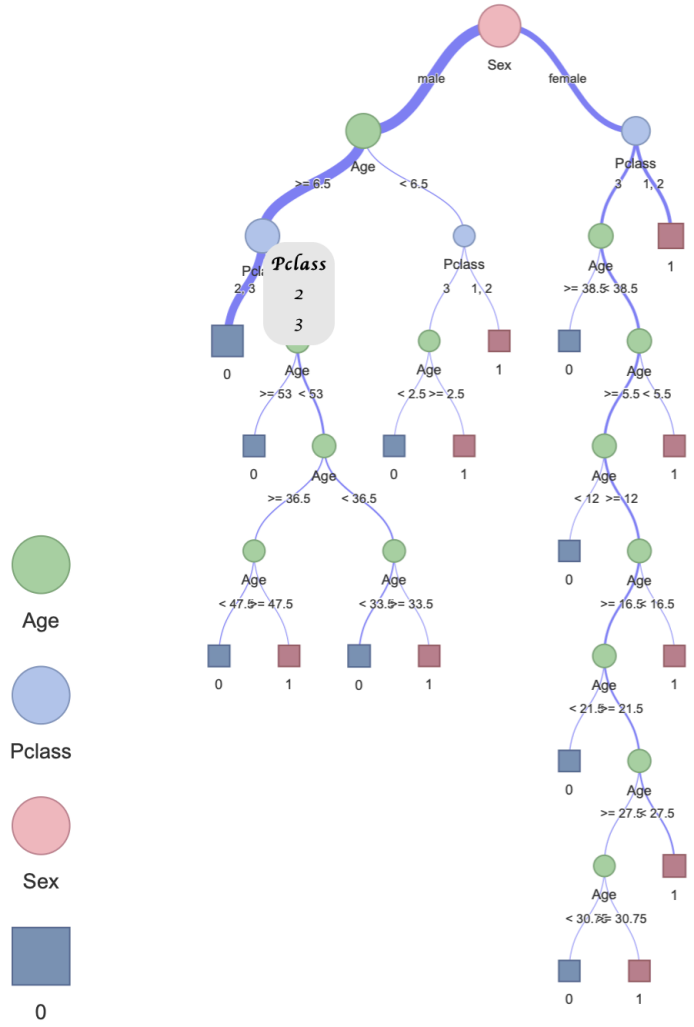
Cross –Validate Our Titanic Data to Determine Optimal Tree Depth

```
# rpart function to select optimal depth of tree
# read the help() file for rpart.control to learn about
#  the different function options
# max depth = 6 ensures the final tree only has this
#  many splits
# min split means minimum observations in a node before
#  a split can be attempted
# cp is the complexity parameter, overall Rsq must
#  increase by cp at each step
titanic_mod_rpart <- rpart(Survived ~ Sex + Pclass + Age,
                           data = titanic_df,
                           method = "class",
                           control = list(cp = 0,
                                          minsplit = 10,
                                          maxdepth = 15))
```



Visualize the Fitted Tree Interactively Using vizTree in VizNetworks Package

```
# fancy, interactive tree visual
install.packages('visNetwork')
install.packages('sparkline')
visNetwork::visTree(titanic_mod_rpart,
                    nodesPopSize = TRUE,
                    edgesFontSize = 18,
                    nodesFontSize = 20,
                    width = "100%",
                    height = "1200px")
```



https://datastorm-open.github.io/visNetwork/

Regression Tree Lab

```
#-------------------------------------------------------------
# Regression Tree Exercises
#-------------------------------------------------------------
# 1. Estimate a regression tree model predicting survival as a function
#    of Sex, Pclass, Age, SibSp and Fare paid using the ctree package.
#    Store this model as titanic_tree_mod2
# 2. Use the print function against the fitted model to view the text
#    Descriptions of the model fit
# 3. Use the plot function on the fitted object to produce the tree plot
#    (you can use the option "gp = gpar(fontsize = 6)")
#    to change the text font size.
# 4. Who has the best chance of survival? Who has the worst?
# 5. If you have time, use the rpart package to cross-validate
#    and select the optimal depth of the model with more predictor variables
#    (Sex, Pclass, Age, SibSp, and Fare). Call this model titanic_mod_rpart_mod2
# 6. Call the plotcp function against Wthe titanic_mod_rpart.
#    What is the optimal depth according to the plotcp.
```

Class 7: Outline

1. AI + Pizza
2. Regression Trees
3. Regression Trees in R
4. Cross-Validating to Determine Optimal Tree Depth
5. Regression Tree Lab
6. Bagging
7. Random Forests
8. Random Forest Lab

ENTERTAINMENT

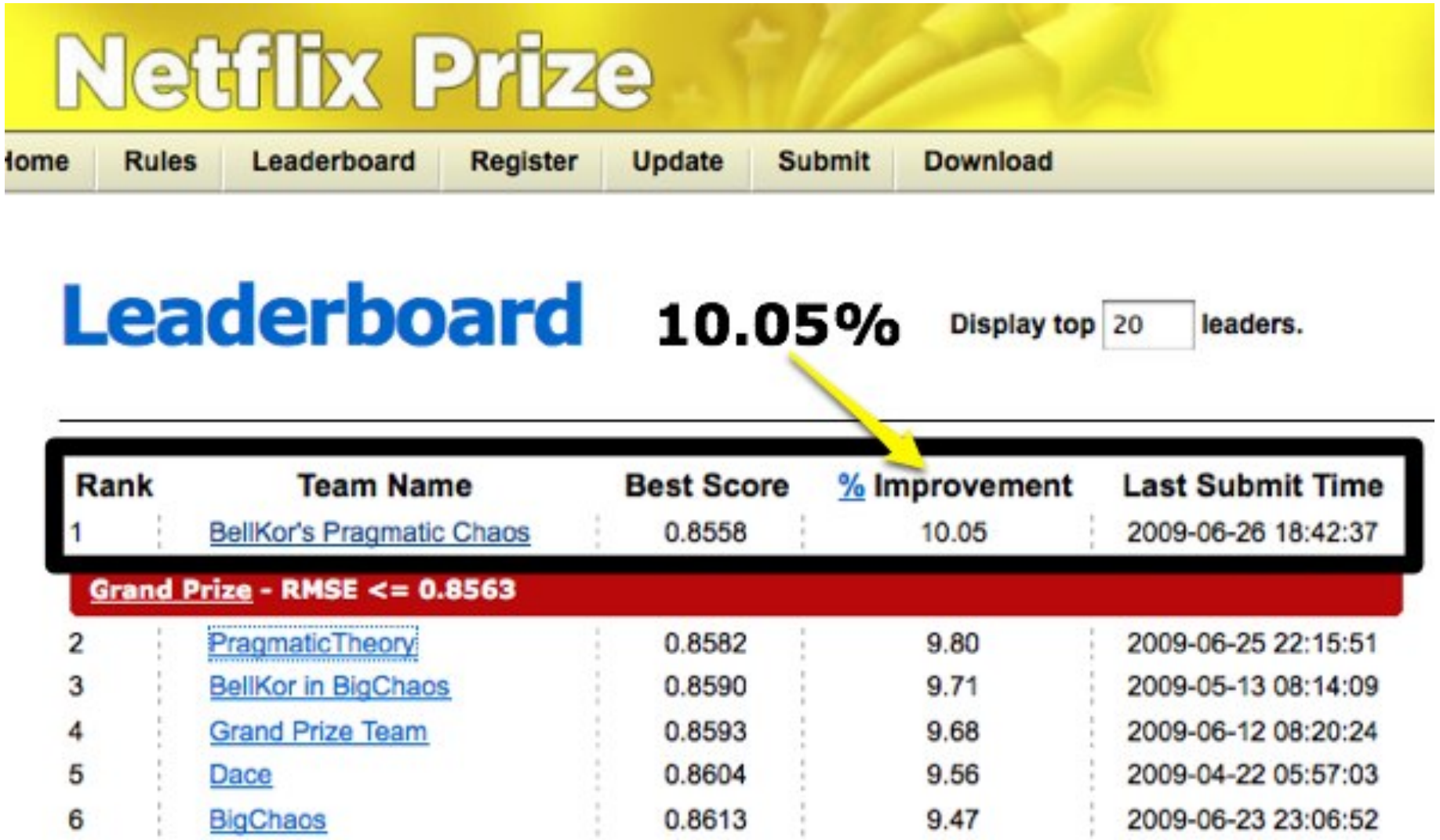
The Netflix Prize: How a $1 Million Contest Changed Binge-Watching Forever

By DAN JACKSON
Published On 07/07/2017
@danielvjackson

In October 2006, Netflix, then a service peddling discs of every movie and TV show under the sun, announced "The Netflix Prize," a competition that lured Mackey and his contemporaries for the computer programmer equivalent of the *Cannonball Run*. The mission: Make the company's recommendation engine 10% more accurate -- or die coding. Word of the competition immediately spread like a virus through comp-sci circles, tech blogs, research communities, and even the mainstream media. ("And if You Liked the Movie, a Netflix Contest May Reward You Handsomely" read the *New York Times* headline.) And while a million dollars created attention, it was the data set -- over 100 million ratings of 17,770 movies from 480,189 customers -- that had number-crunching nuts salivating. There was nothing like it at the time. There hasn't been anything quite like it since.

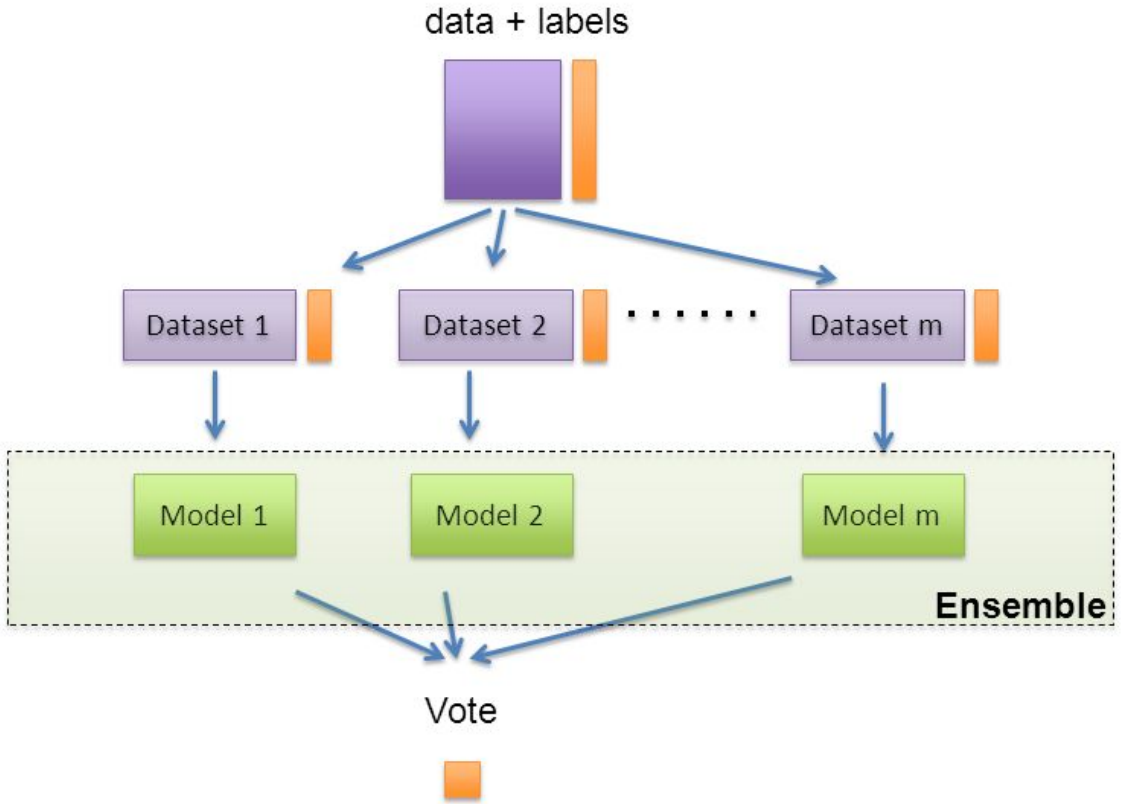
https://www.thrillist.com/entertainment/nation/the-netflix-prize

Netflix prize



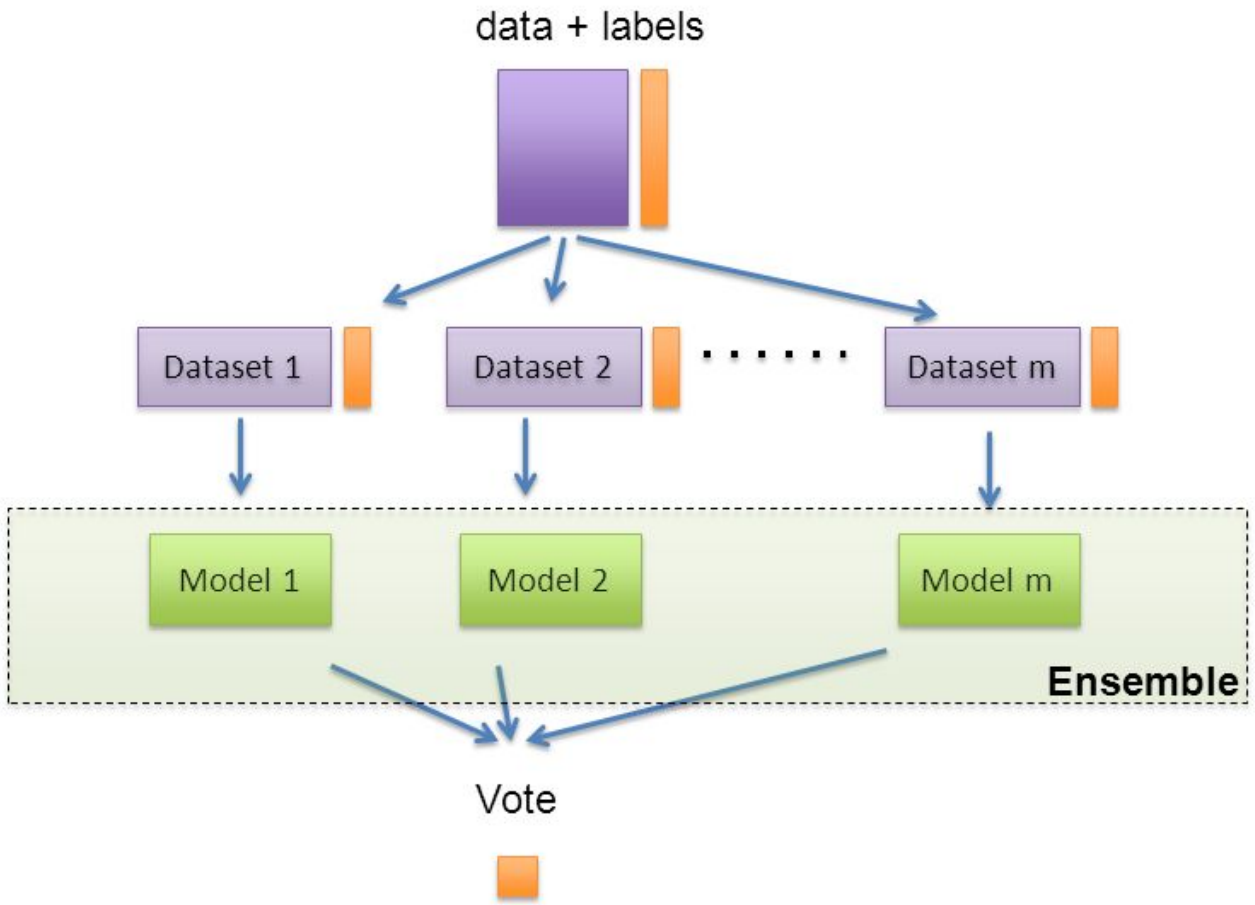
Rank	Team Name	Best Score	% Improvement	Last Submit Time
1	BellKor's Pragmatic Chaos	0.8558	10.05	2009-06-26 18:42:37
Grand Prize - RMSE <= 0.8563				
2	PragmaticTheory	0.8582	9.80	2009-06-25 22:15:51
3	BellKor in BigChaos	0.8590	9.71	2009-05-13 08:14:09
4	Grand Prize Team	0.8593	9.68	2009-06-12 08:20:24
5	Dace	0.8604	9.56	2009-04-22 05:57:03
6	BigChaos	0.8613	9.47	2009-06-23 23:06:52

Netflix prize winners



- Punchline: simple models beat out one very deep model

Netflix prize conclusion: ensemble of simple methods beats one complex method



Bagging

- Bagging is short for bootstrap aggregation
- **It's a general purpose method for reducing variance in any machine learning method**

- With n independent observations, $z_1, z_2, \ldots, z_n$ each with variance σ^2 , the variance of the mean ($\bar{z}$) is given by $\sigma^2/\bar{z}$
- We usually cannot do this because we don't have multiple training datasets

Bagging (Bootstrap Aggregation)

Bagging algorithm

1. Generate B bootstrap training datasets
2. Train method on the b-th bootstrapped set to obtain $\hat{f}^*(x)$ the prediction for data point x
3. Average all predictions to obtain average prediction over the bootstrapped samples

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

- Netflix prize intuition: average of many models will perform better than a single model

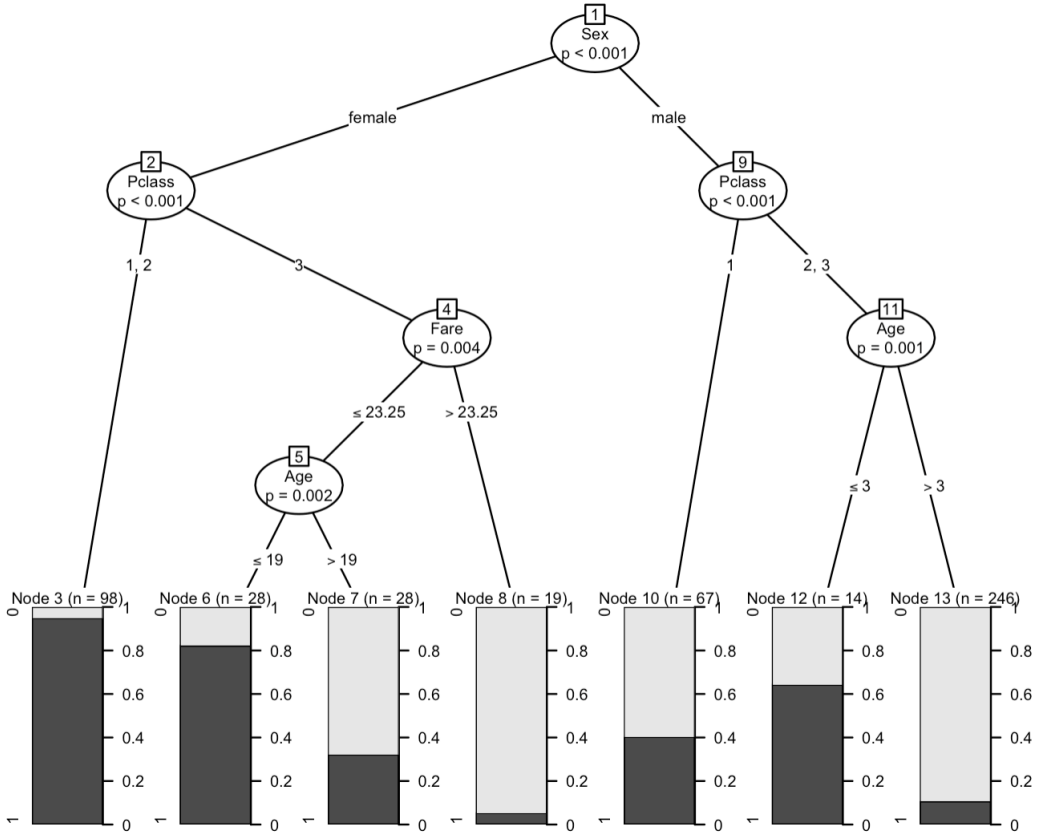
Bagging Model of Titanic Data

```r
# store rownames as columns
titanic_boot_preds <- titanic_df %>% rownames_to_column() %>%
  mutate(rowname = as.numeric(rowname))

B <- 100      # number of bootstrap samples
num_b <- 500  # sample size of each bootstrap
boot_mods <- list() # store our bagging models
for(i in 1:B){
  boot_idx <- sample(1:nrow(titanic_df),
                     size = num_b,
                     replace = FALSE)
  # fit a tree on each bootstrap sample
  boot_tree <- ctree(Survived ~ Pclass + Sex + Age + SibSp + Fare,
                     data = titanic_df %>%
                       slice(boot_idx))
  # store bootstraped model
  boot_mods[[i]] <- boot_tree
  # generate predictions for that bootstrap model
  preds_boot <- data.frame(
    preds_boot = predict(boot_tree),
    rowname = boot_idx
  )
  # rename prediction to indicate which boot iteration it came from
  names(preds_boot)[1] <- paste("preds_boot",i,sep = "")
  # merge predictions to dataset
  titanic_boot_preds <- left_join(x = titanic_boot_preds, y = preds_boot,
                                  by = "rowname")
}
```

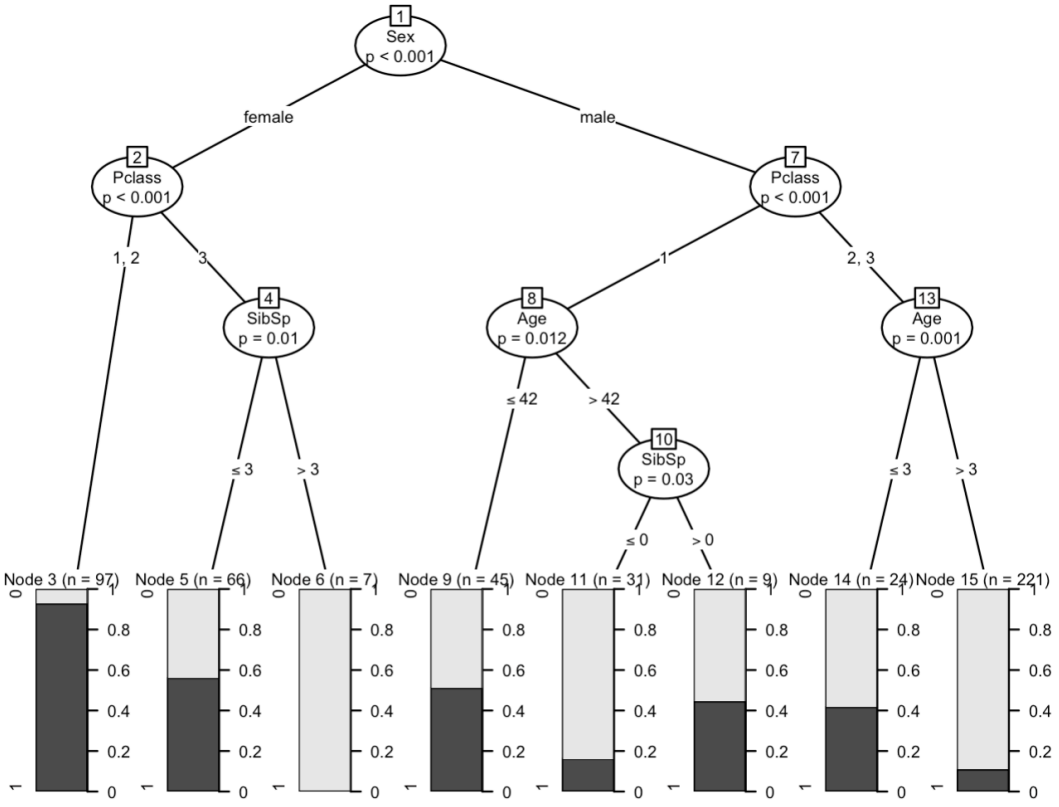
- Do not need to understand code here!
- The important part is you understand conceptually how bagging works

Examine Individual Tree Models In Bagging Aggregated Model

```
plot(boot_mods[[10]])
```

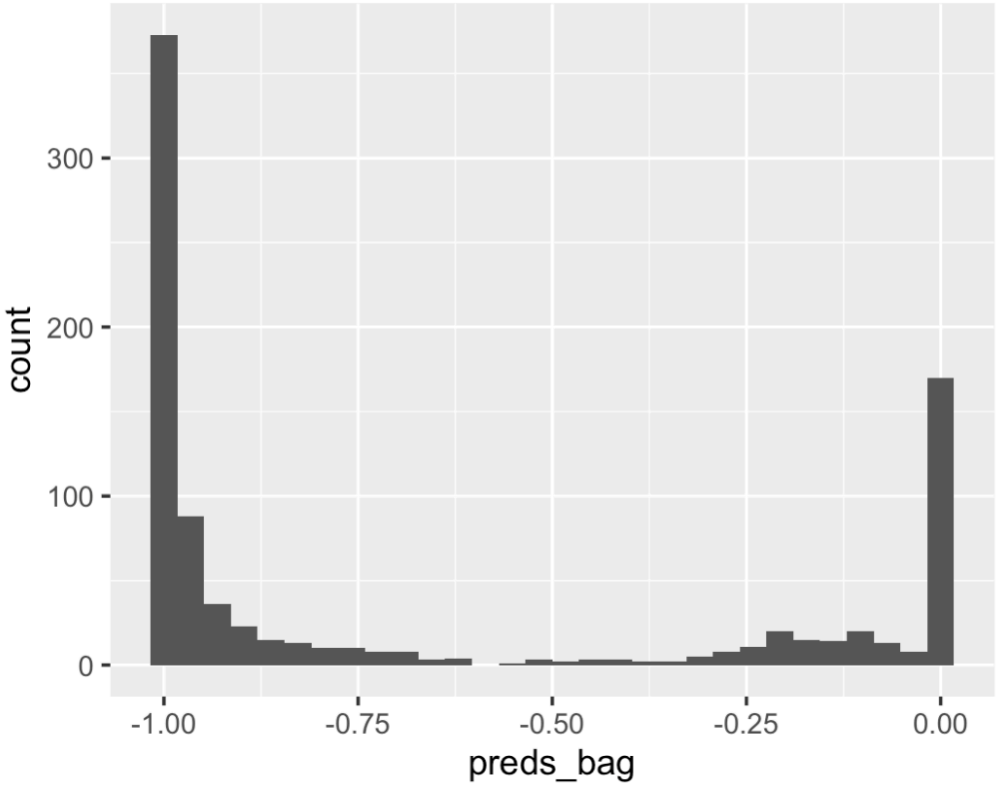



```
plot(boot_mods[[20]])
```



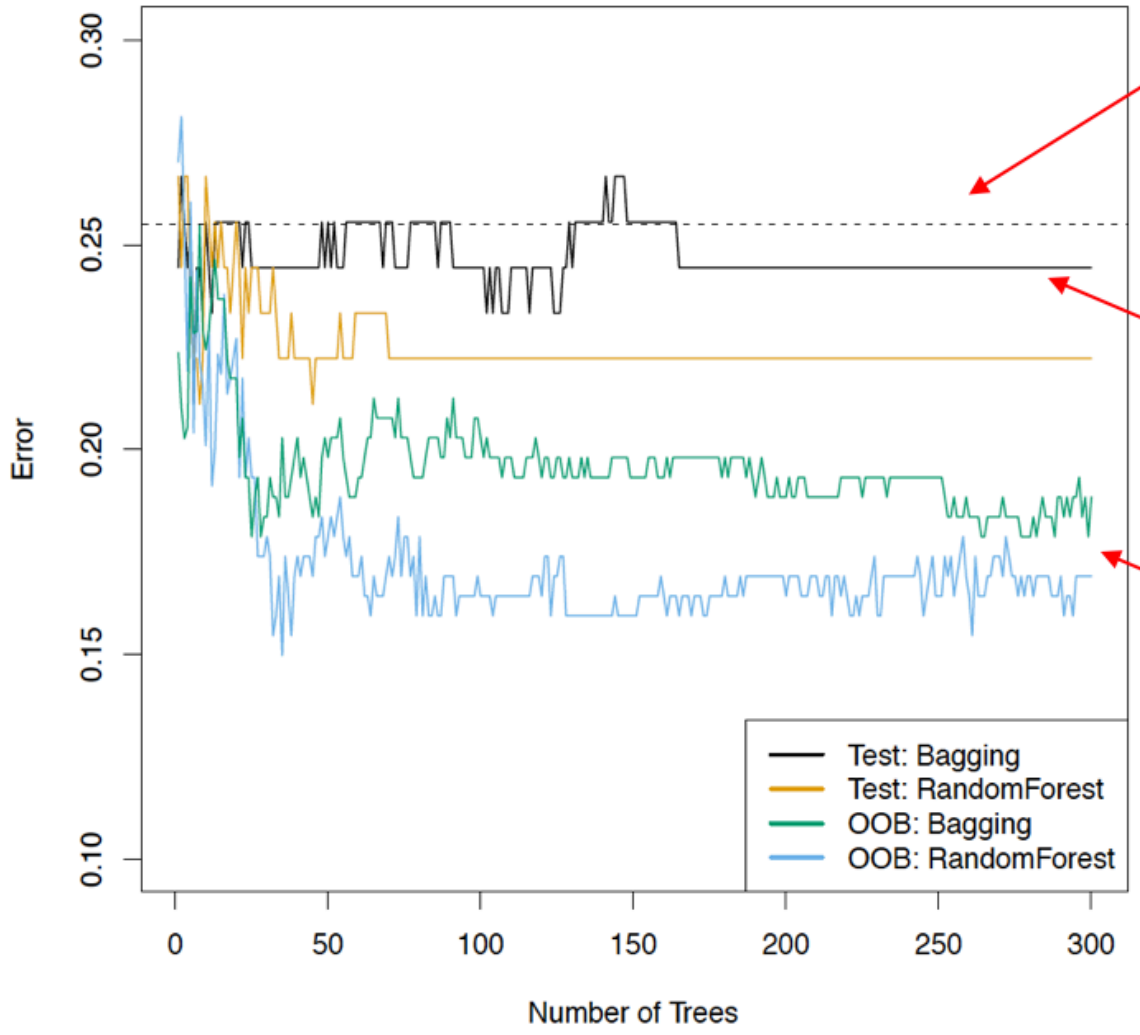
Row Means Over Columns to Average Predictions

```
# must convert factor into numeric, note that class "0" = 1,
# and class "1" = 2, so we need to subtract 1 from every column
titanic_boot_preds %<>% mutate_if(is.factor, as.numeric) %>%
  mutate_all(function(x){x - 1})

# calculate mean over all the bootstrap predictions
titanic_boot_preds <- titanic_boot_preds %>%
  mutate(preds_bag =
  select(., preds_boot1:preds_boot100) %>%
  rowMeans(na.rm = TRUE))

# congratulations! You have bagged your first model!
ggplot(titanic_boot_preds, aes(x = preds_bag)) +
  geom_histogram()
```



- Note: you don't need to worry about replicating this code. It is just here in case you are curious. But do make sure you understand the *concept* about bagging.

Bagging Results As We Vary Number of Trees



Test error single tree

Test error Bagging

"out of bag" i.e. Training error on Observations not Used in bootstrapped sample

Out-of-bag error

- Recall that for each bootstrapped sample *b* is composed of a subset of the total training data
- For each sample, the data not used to fit the model is referred to as **out-of-bag (OOB) observations**
- We can better approximate out of sample error by only using out-of-bag observations for model validation

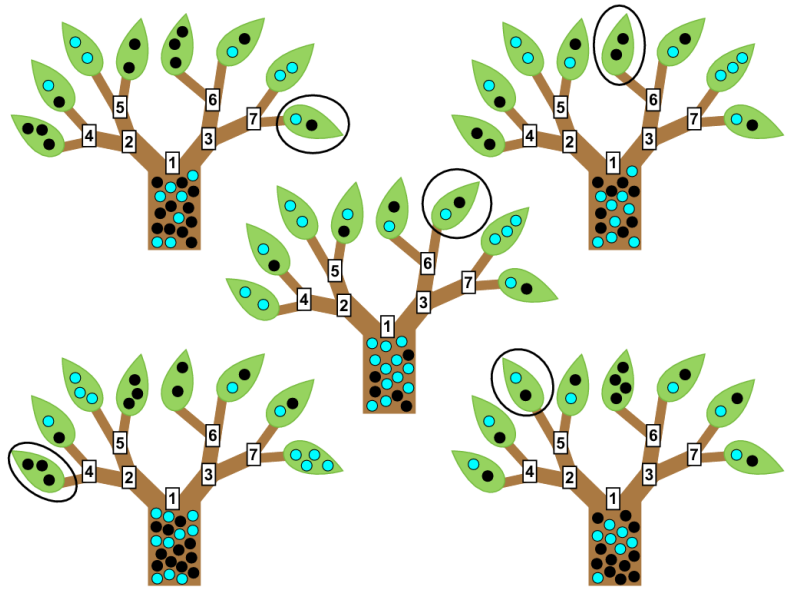
preds_boot1	preds_boot2	preds_boot3
0	0	0
0	0	0
NA	0	1
0	*NA*	0
NA	1	0

Class 7: Outline

1. AI + Pizza
2. Regression Trees
3. Regression Trees in R
4. Cross-Validating to Determine Optimal Tree Depth
5. Regression Tree Lab
6. Bagging
7. **Random Forests**
8. Random Forest Lab

Random Forests

- Random forests are a slight trick to bagging that highly improves predictive power
- **Many trees do poorly because the stepwise greedy algorithm doesn't fully explore variable and parameter space**
- Random forests is like bagging, only each time a split in a tree is considered, a random selection of m predictors is chosen as split candidate
- A fresh set of m predictors is taken at each split.



Random Forest Model: Many Decision Trees

B bootstrap sampes of data (~ 1000)

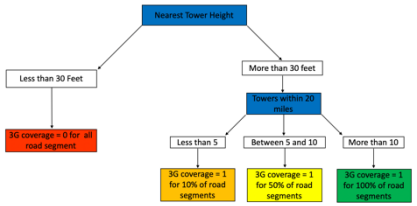
...

Bootstrap 1

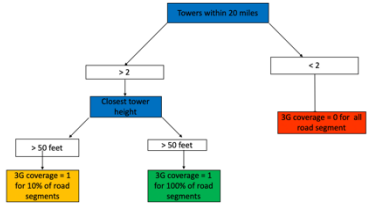
Bootstrap b

Bootstrap 1000

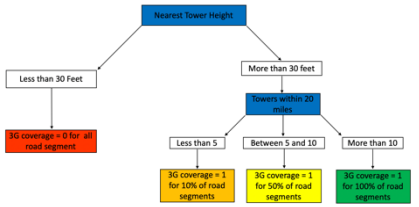
Tree 1



Tree b



Tree 1000



Vote road i has 3G access = 0 or 1

Vote road i has 3G access = 0 or 1

Vote road i has 3G access = 0 or 1

Estimating Random Forest Models Using “randomForest”

randomForest {randomForest} R Documentat

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 an 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,
             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, ...)
## S3 method for class 'randomForest'
print(x, ...)
```

- Key parameters in the randomForest package:
  - Mtry: number of variables to randomly select at each candidate node split
  - Ntree: number of regression or classification trees used to fit total random forest model

Estimating Random Forest Model Using randomForest() package

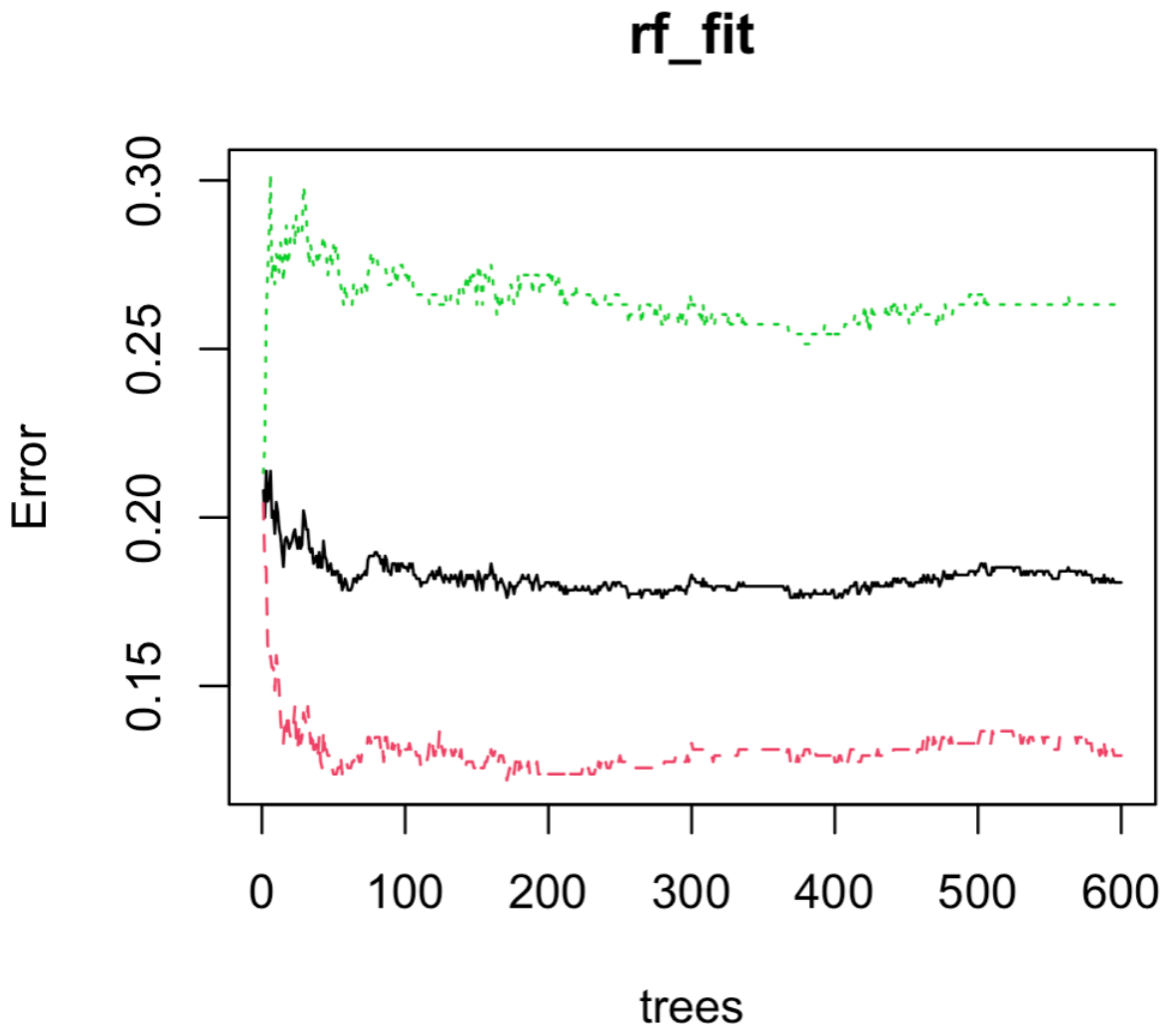
```
#-----------------------------------------------------------
# Random Forest
#-----------------------------------------------------------
library('randomForest')

rf_fit <- randomForest(Survived ~
                         Pclass + Sex + Age + SibSp + Fare,
                       data = titanic_df,
                       type = classification,
                       mtry = 3,
                       na.action = na.roughfix,
                       ntree = 600)

rf_fit
```

- Must specify formula and dataset to use per usual
- Additionally must specify “mtry” the number of variables to sample (randomly) for each node
- Missing values will cause an error and this rough fix replaces them with median values
- Ntree specifices the number of trees in the random forest

Random Forest Model Object

```
> print(rf_fit)

Call:
 randomForest(formula = Survived ~ Pclass + Sex + Age + SibSp +
tion = na.roughfix)
               Type of random forest: classification
                     Number of trees: 600
No. of variables tried at each split: 5

        OOB estimate of  error rate: 18.07%
Confusion matrix:
    0   1 class.error
0 478  71   0.1293260
1  90 252   0.2631579
```

plot() function against rf object – 600 trees, mtry = 3



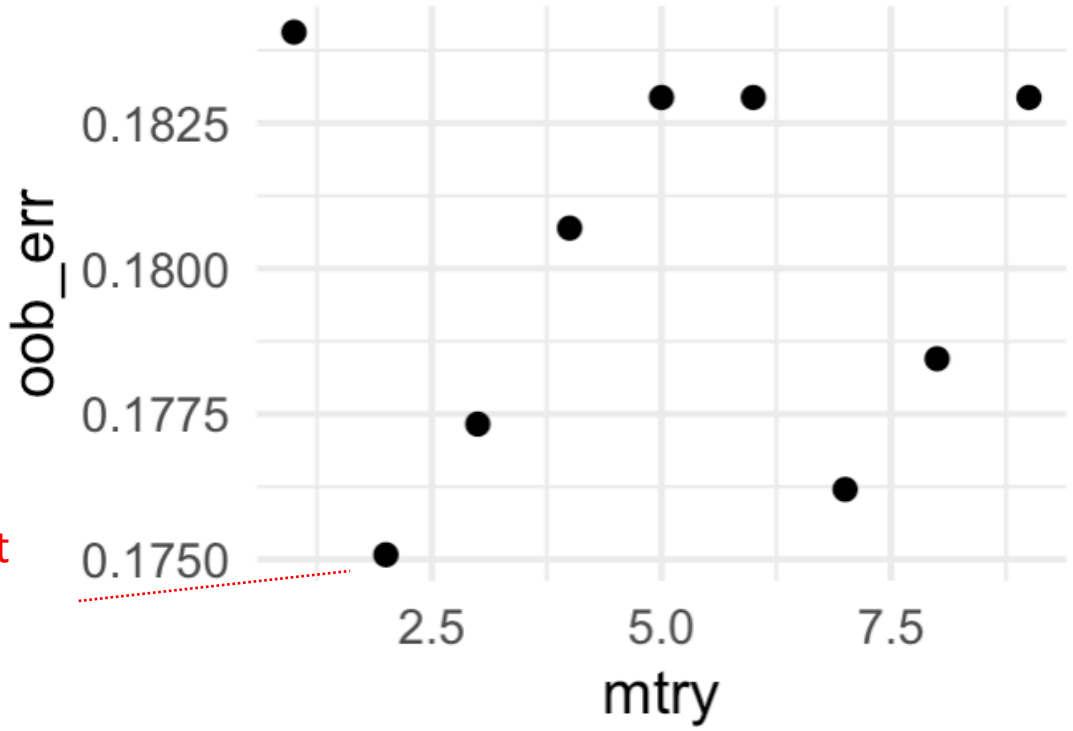
- Green is error rate for positive class, red is error rate for negative class (didn't survive), and black is overall error rate (all out of bag error)
- Error seems to stabilize at 100-200 trees, so we only need around 200 trees for this prediction problem

Cross-validate to Tune/Select Optimal "mtry"

```
#-------------------------------------------------------------
# Tuning Random Forests To Determine
#  Optimal Parameters (mtry)
#-------------------------------------------------------------
rf_mods <- list()
oob_err <- NULL
test_err <- NULL
for(mtry in 1:9){
  rf_fit <- randomForest(Survived ~
                           Pclass + Sex + Age + SibSp + Fare,
                         data = titanic_df,
                         mtry = mtry,
                         na.action = na.roughfix,
                         ntree = 600)
  oob_err[mtry] <- rf_fit$err.rate[600]

  cat(mtry," ")
}
```

- In the previous, we just guessed that mtry = 3 is the right parameter.
- To get a better model fit, we can estimate a model for every value of mtry and choose the parameter that maximizes cross-validated (out of bag) fit
- Most packages will cross-validate mtry for us, but in principle it is the same as this code

Cross-validate to select “mtry”



Mtry value With lowest error

- If we plot the OOB (out of bag) error against mtry, we find mtry = 2 gives us the lowest OOB error.
- Therefore the “tuned” model sets mtry = 2, and ntree = 200
- To see true out of sample fit, we use these tuned parameters to predict out of sample fit on the test set

```
results_DF <- data.frame(mtry = 1:9, oob_err)
ggplot(results_DF, aes(x = mtry, y = oob_err)) + geom_point() + theme_minimal()
```

Random Forest Exercises (Time Permitting)

```
#-------------------------------------------------------------
# Random Forest Exercises
#-------------------------------------------------------------
# 1. Estimate a random forest model predicting survival using the predictors
#    Pclass, Sex, Age, SibSp and Fare. First set mtry = 5, and select ntree = 400.
#    Call this model rf_fit
# 2. Call the print function against rf_fit
# 3. call the plot() function against the fitted model and describe the plot.
# 4. How many trees should we use to estimate the model?
```

Class 7 Summary

- Regression trees use binary decisions of the X variables to predict the outcome
- Regression trees can be used for either regression or classification problems
- We must control the depth and/or complexity of a tree or else it is prone to overfitting
- Bagging, or bootstrap aggregation, creates an ensemble of models that performs better than a single model
- Random Forests are specific bagging applied to regression trees, where we randomize/sample which variables we use to build the tree
- The randomness allows the model to fully explore the predictor space (Xs) and not use the single one or two most important variables as the first node of every model
- Random Forests have parameters that must be "tuned" for optimal performance.
- Most packages do this tuning automatically for us but it is an important step!