6. Regression Trees and Random Forests

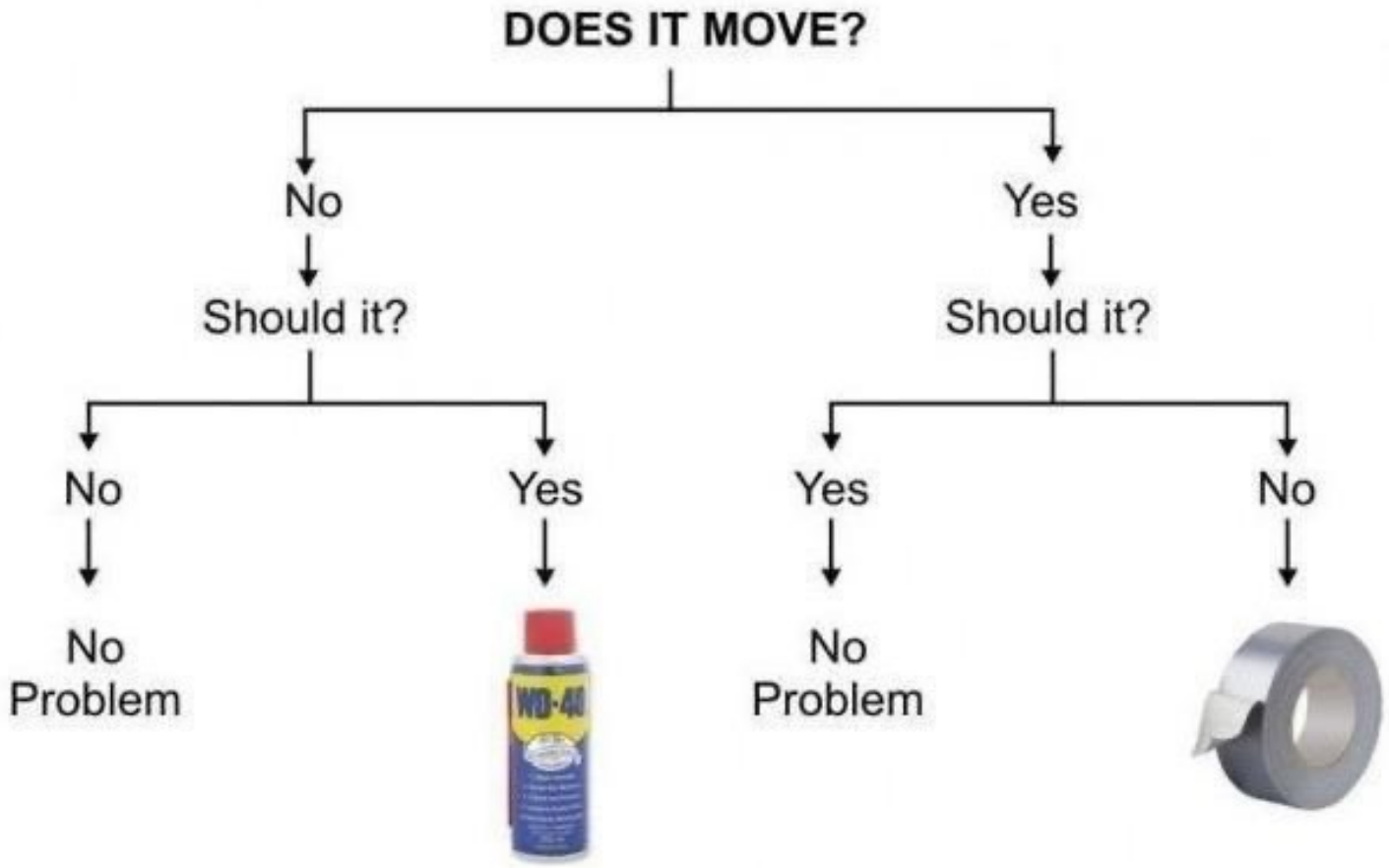
Jonathan Hersh (Chapman University Argyros School of Business)

Outline

1. Regression Trees
2. Bagging and Ensemble Methods
3. Random Forests

Regression Trees

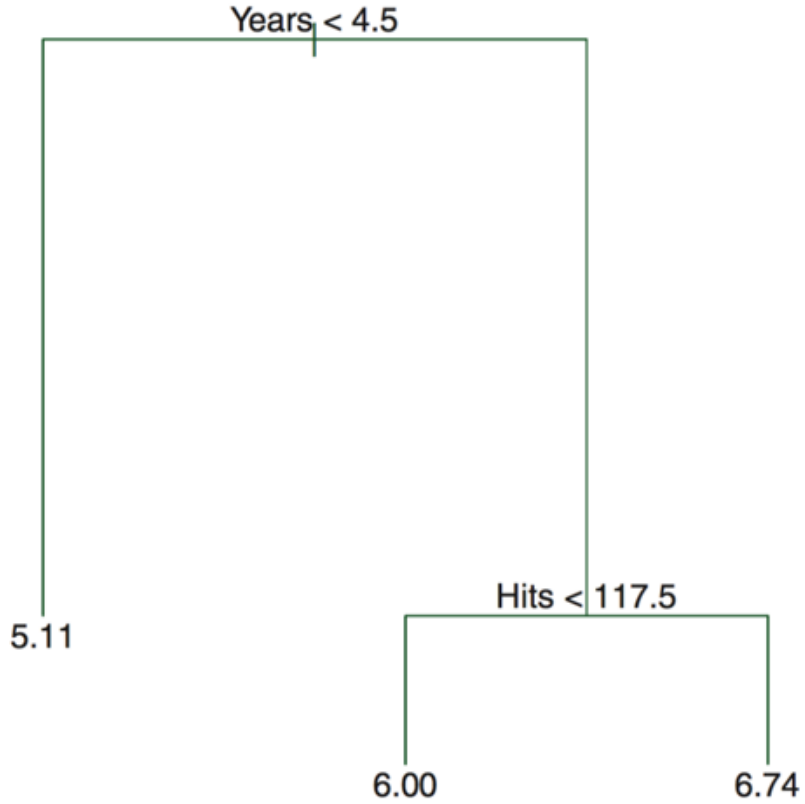
What Are Binary Decision Rules?



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

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

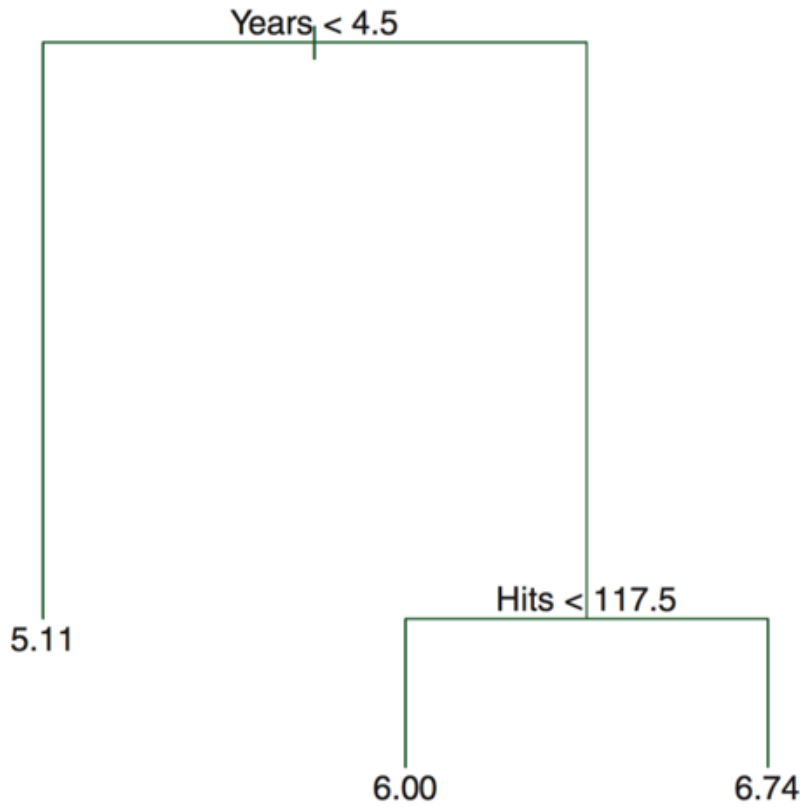
- Simple
- Easy to interpret
- Easy to explain
- Can be displayed graphically!
- Bagging, boosting, and random forests very powerful (combining trees)

Cons

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

Decision/Regression Trees

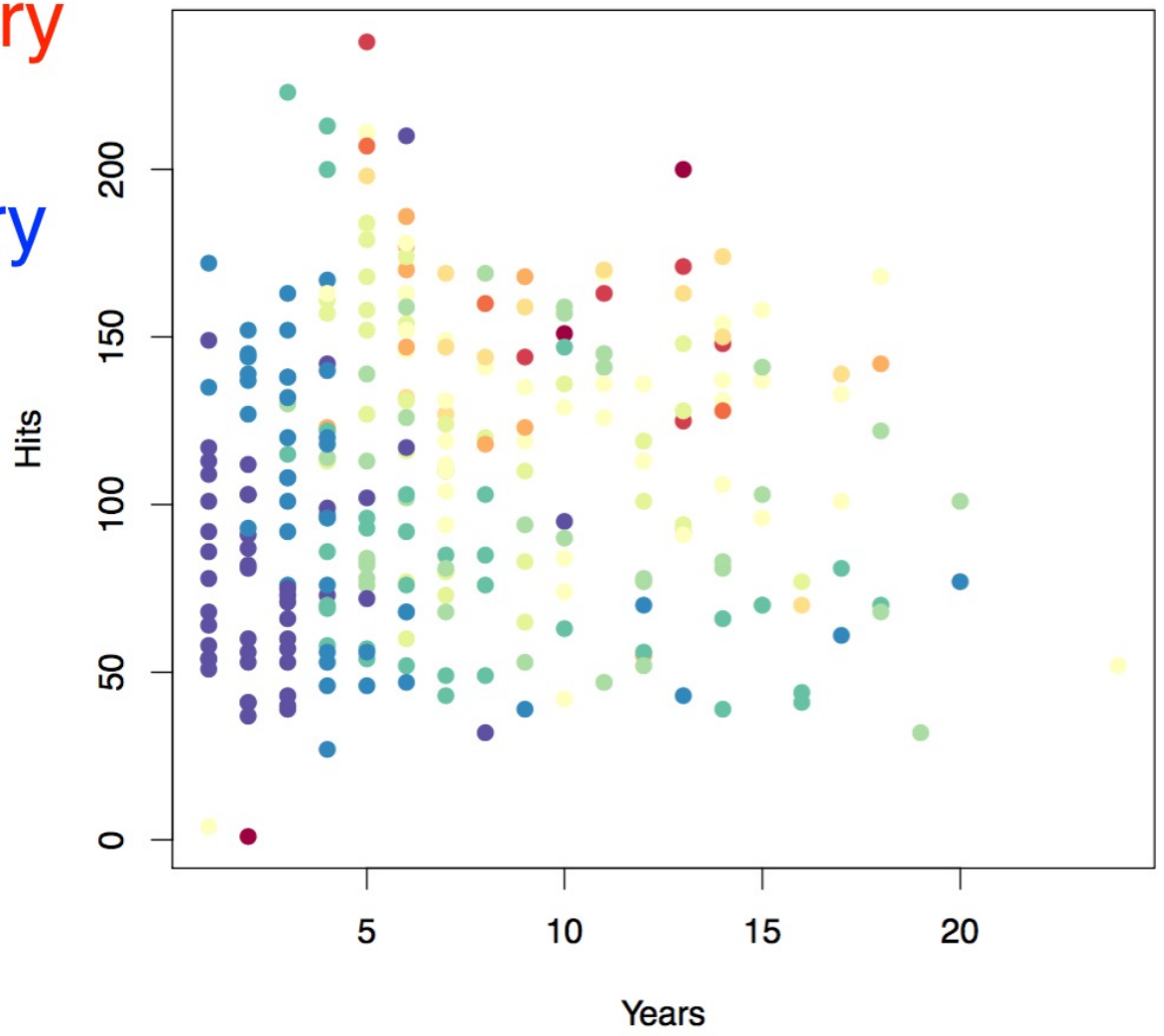
- Decision trees can be applied to both regression problems ($y_i \in R$) and classification problems $y_i \in \{class1, class2, \dots, \}$
- We'll consider both



Baseball salary data: how to partition/stratify?

High salary
red

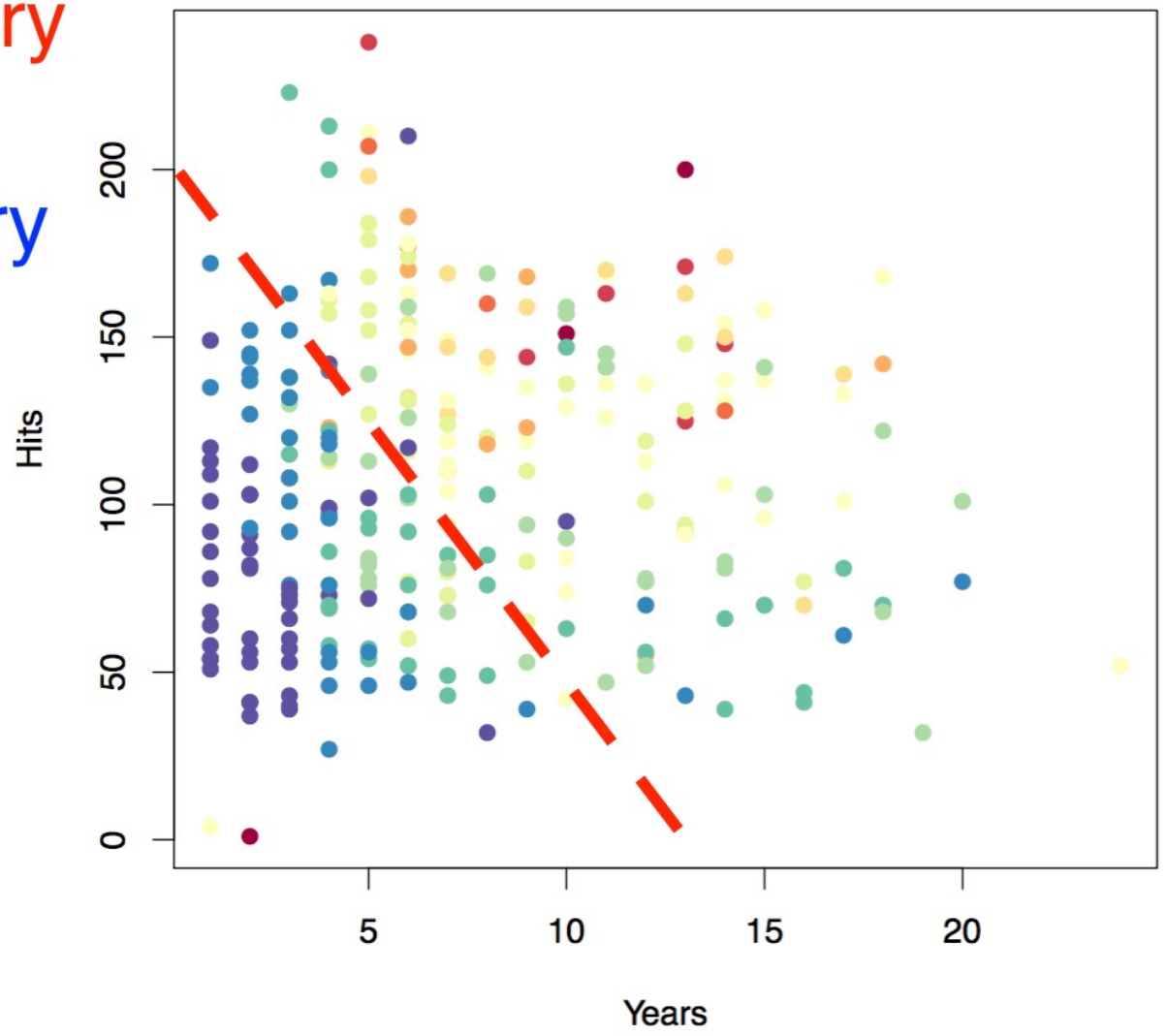
Low salary
blue



Baseball salary data: how to partition/stratify?

High salary
red

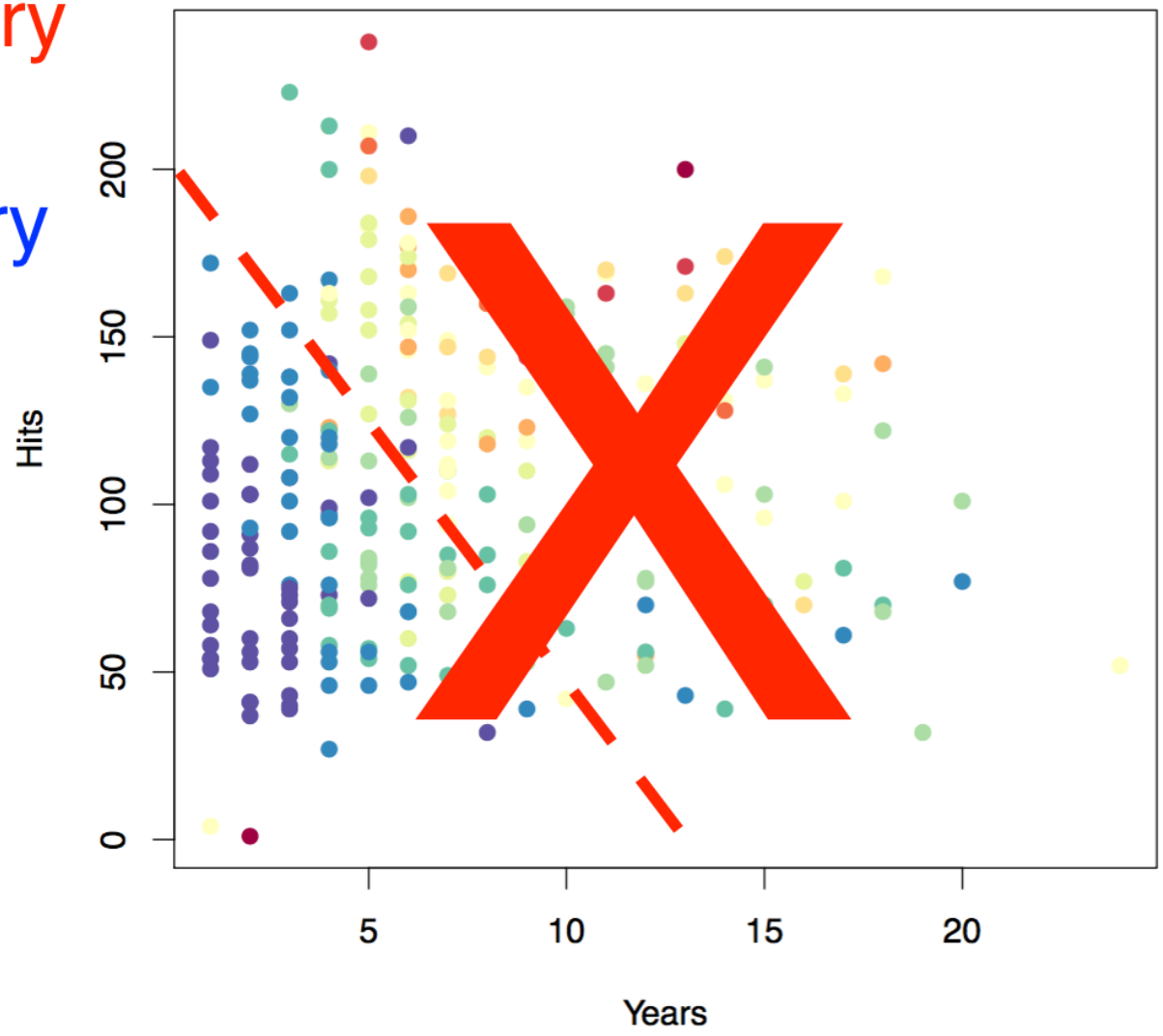
Low salary
blue



Baseball salary data: how to partition/stratify?

High salary
red

Low salary
blue

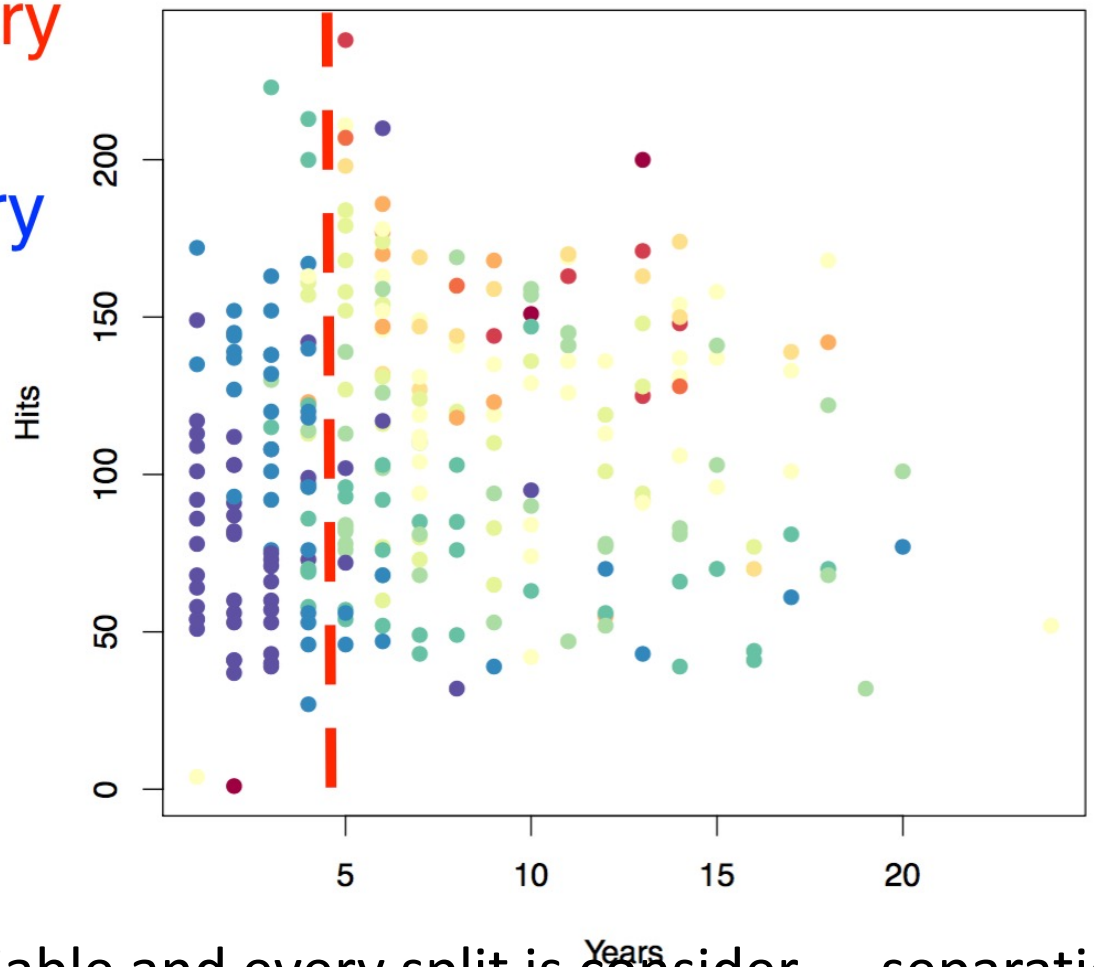


- Only linear classification rules are allowed, e.g. year > 10

How Regression Trees are Constructed

High salary
red

Low salary
blue

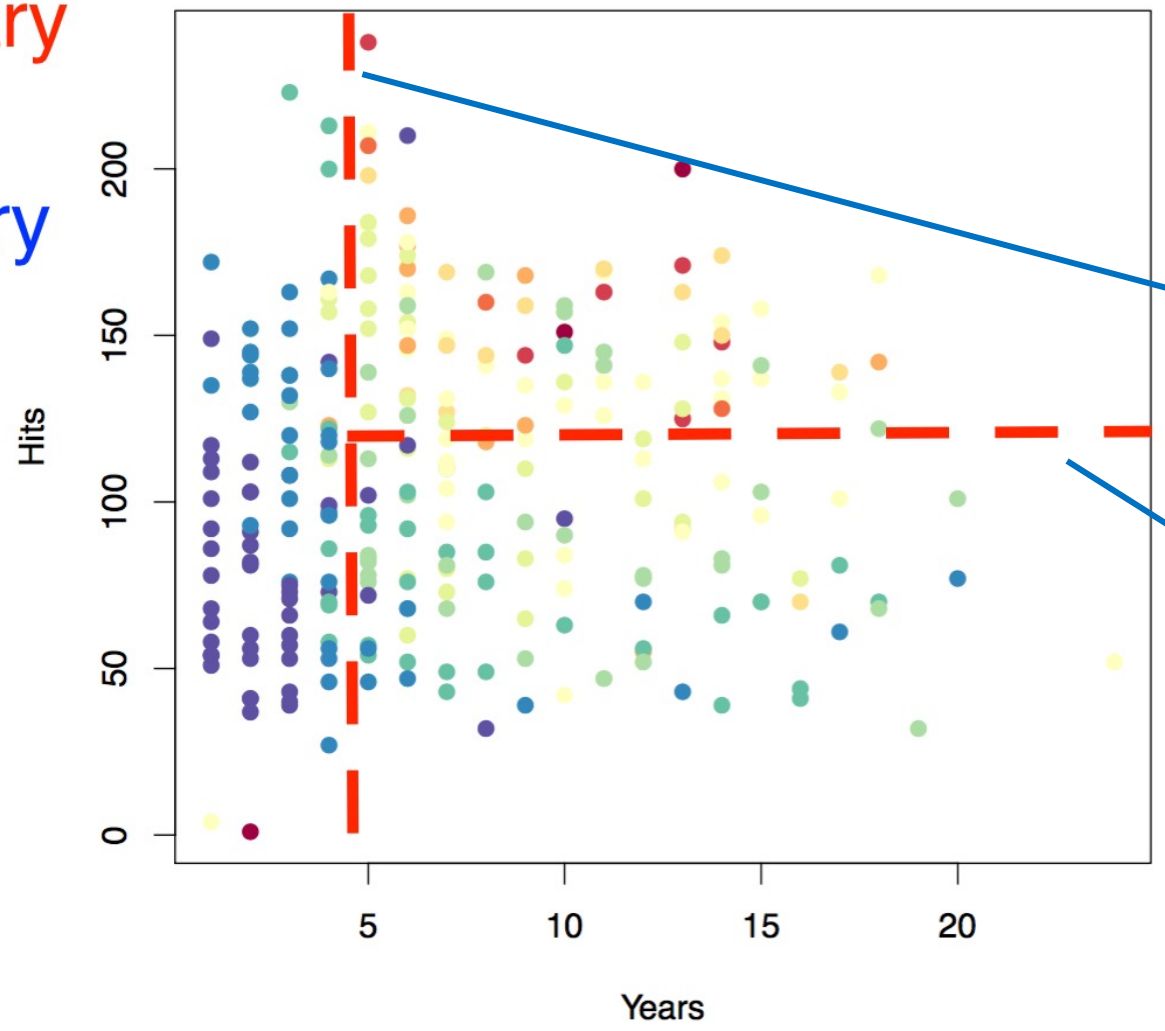


- Every variable and every split is consider.
- Chosen split is one which maximizes separation of high and low salaries:
- Split 1: years > 4.5

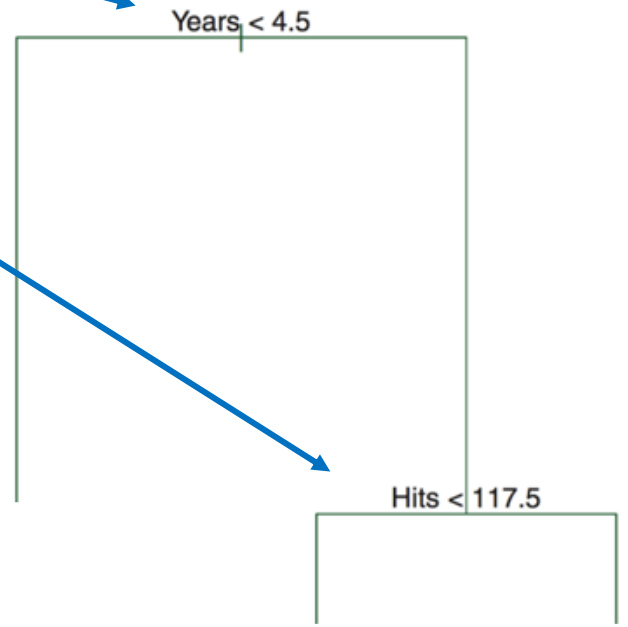
Baseball salary data: split 2

High salary red

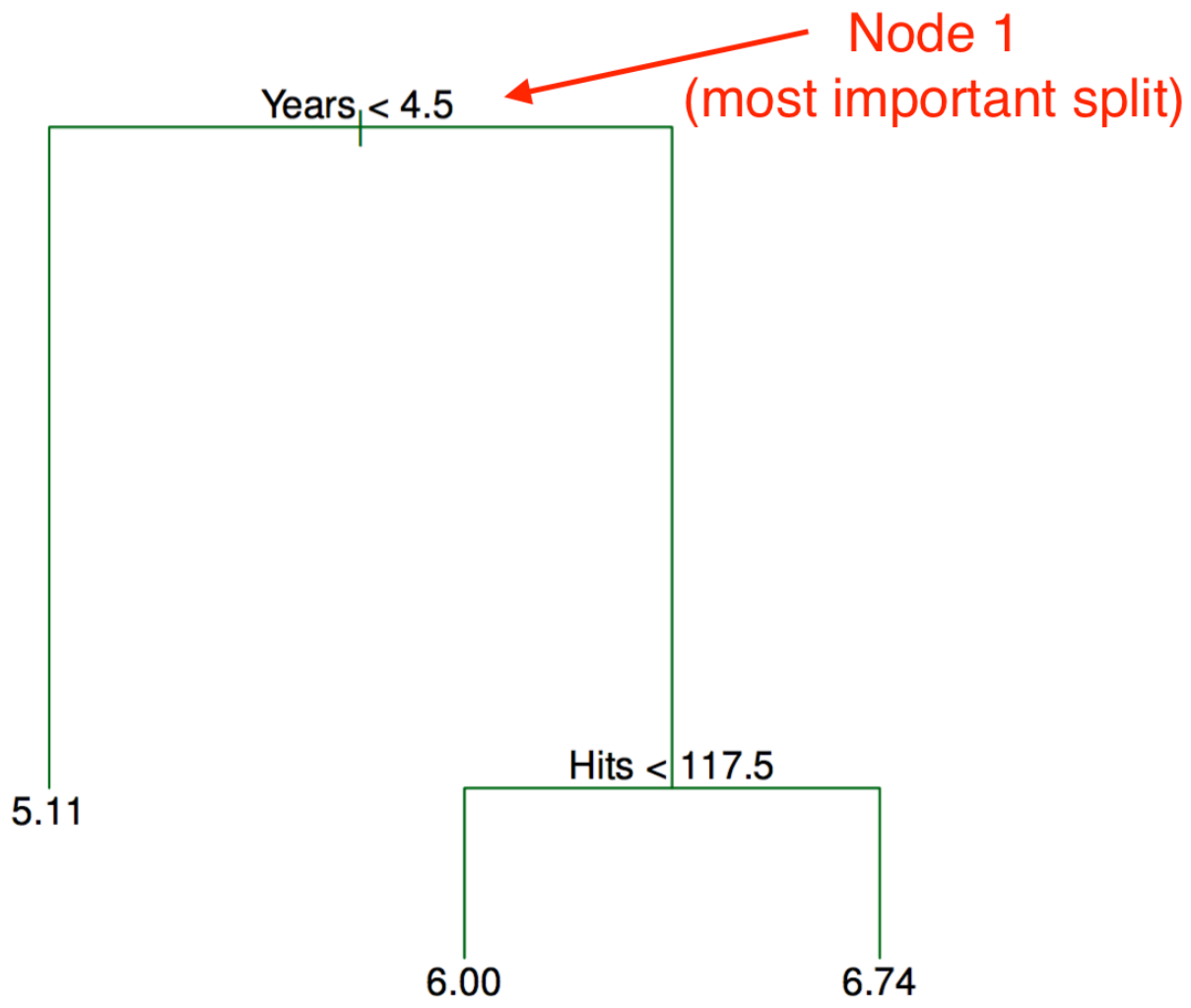
Low salary blue



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

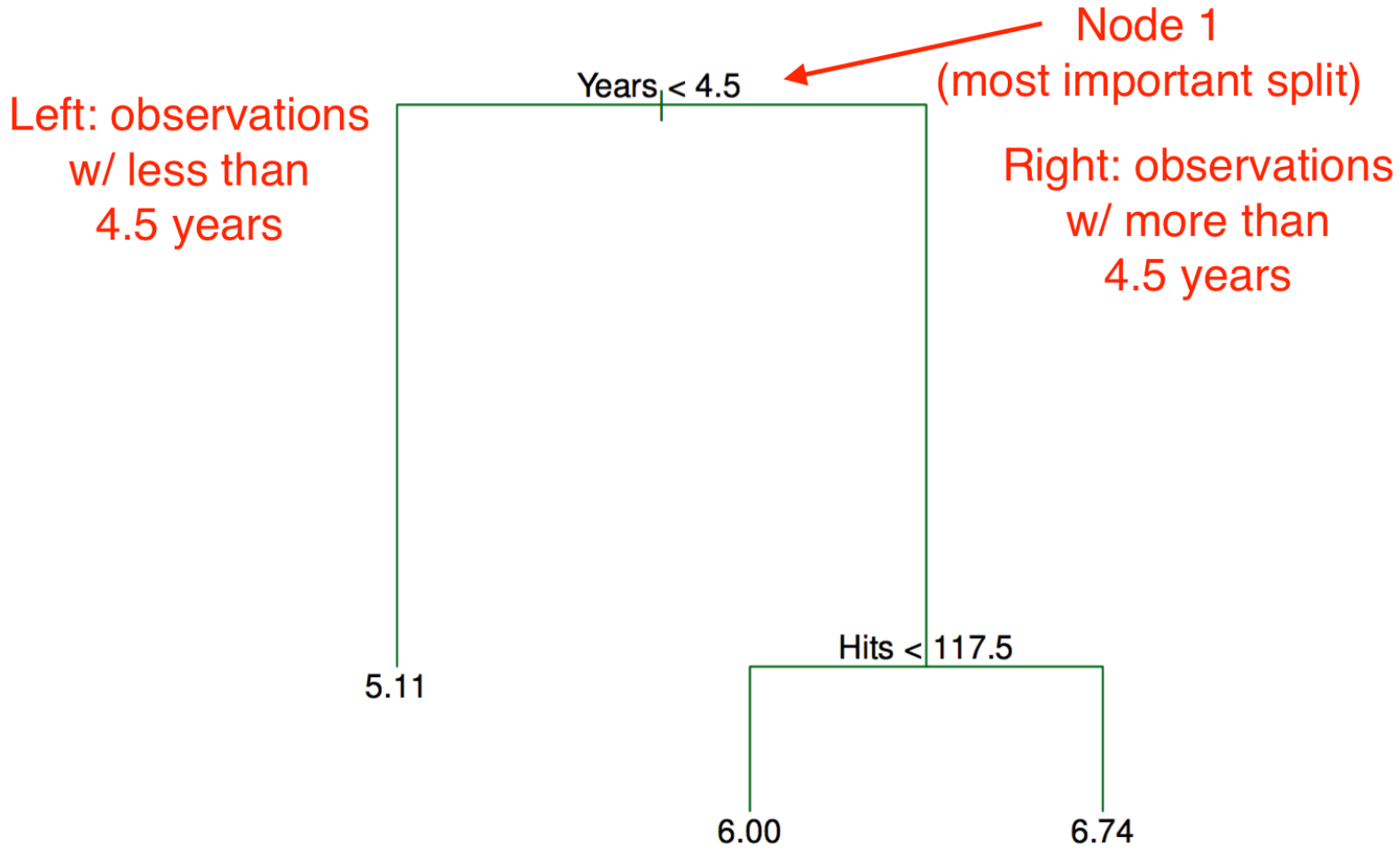


Tree Representation

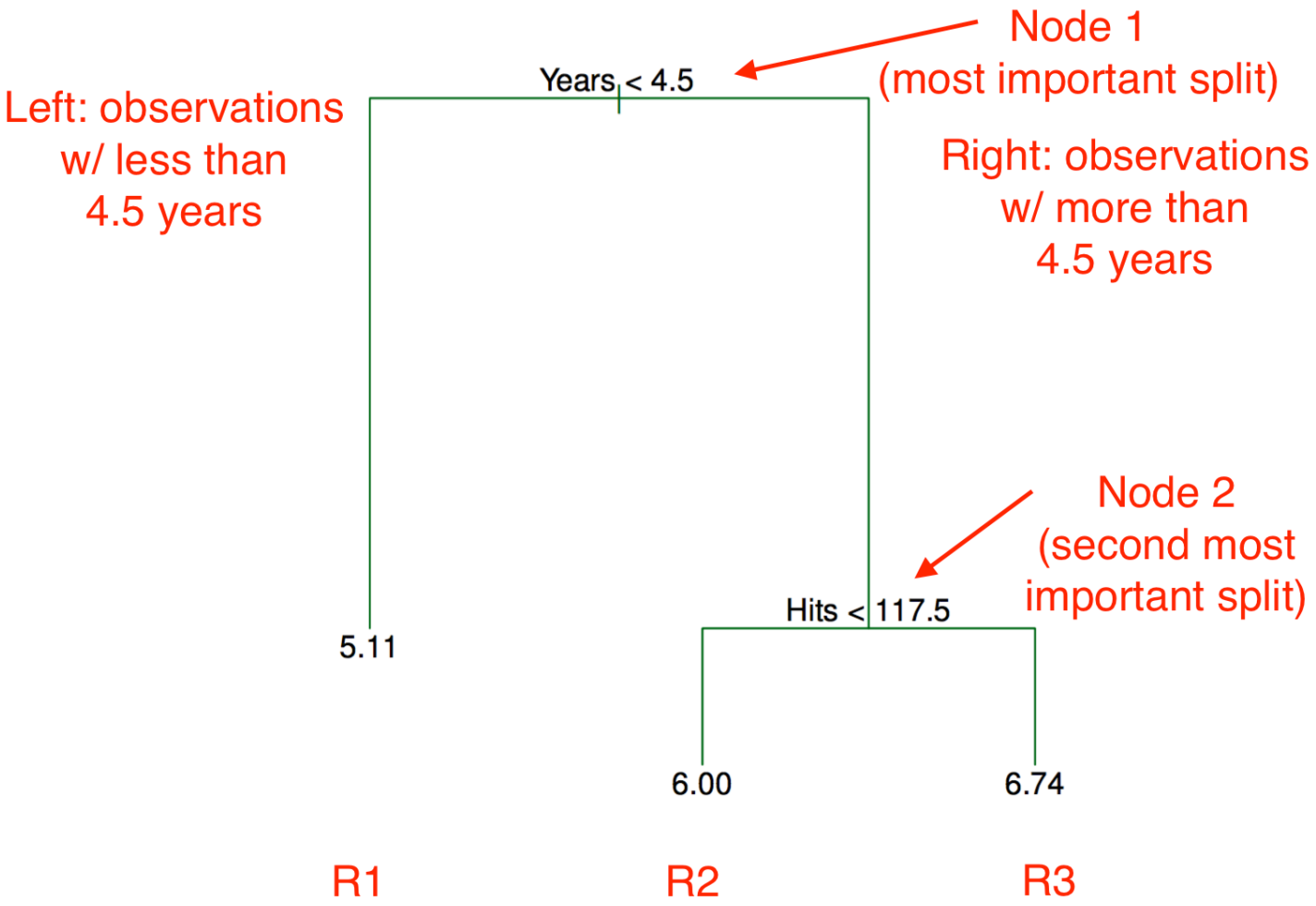


- Trees are read top-down
- Most important split is at top
- Length represents how much within-cluster variance decreases from split
- So Years explains more variance than Hits for this tree

Tree Representation

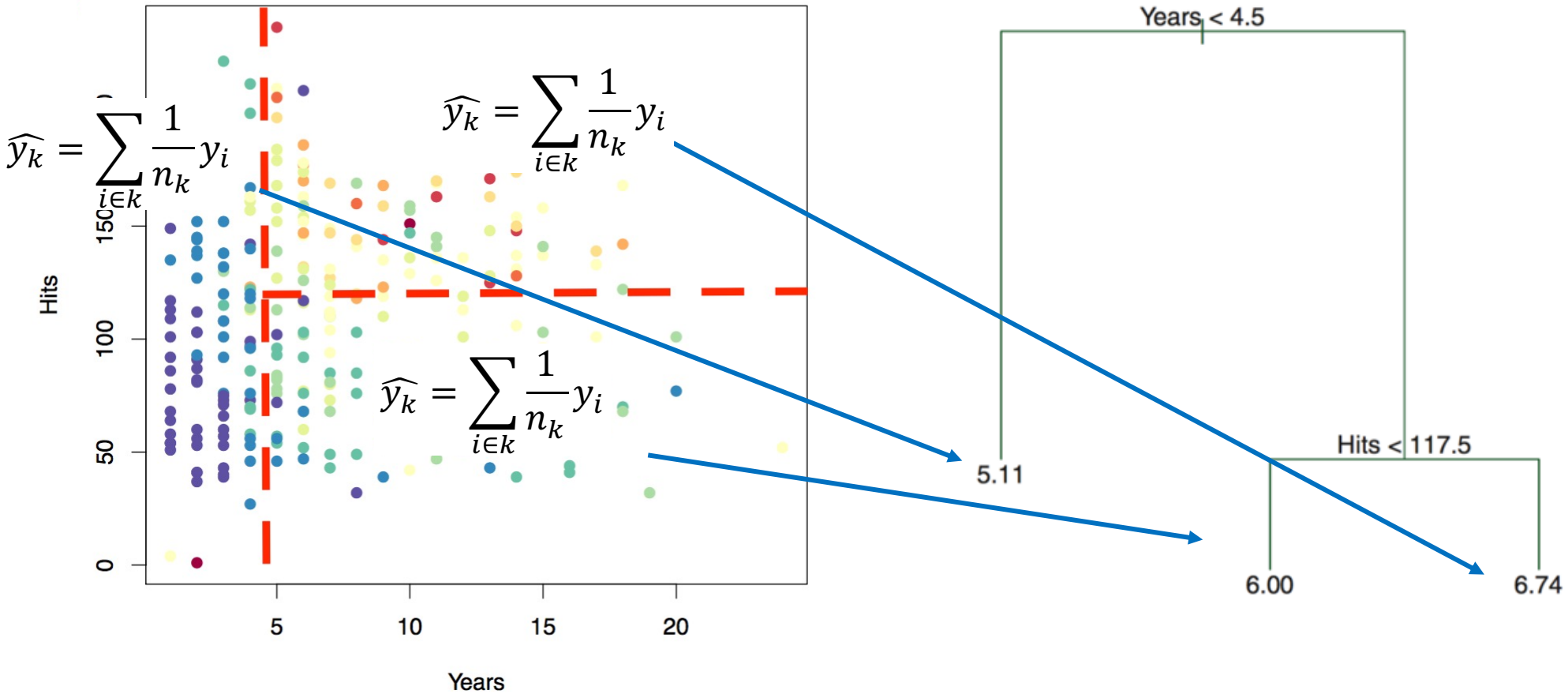


Predictions for Trees? "Leaf" Values



- 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

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 bank account

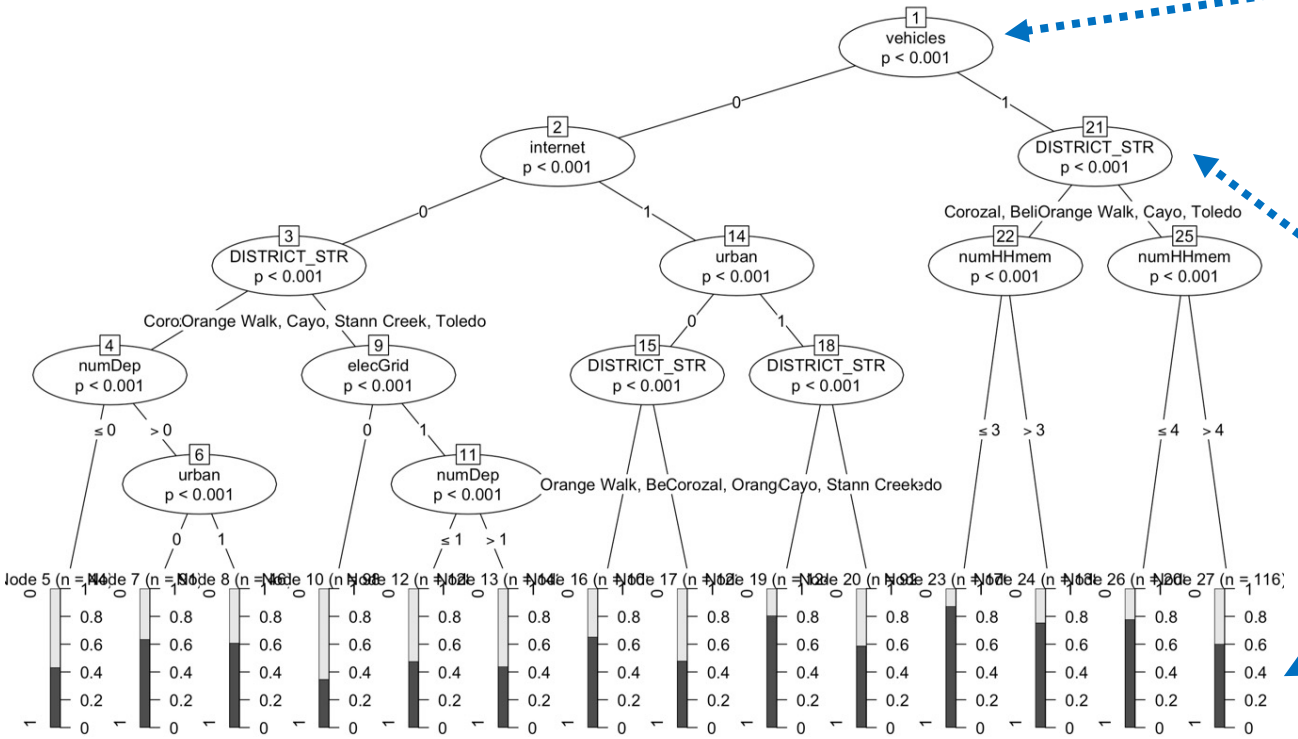
```
bank_tree <- ctree(any_bank_account ~ urban  + toiletPoor + numDep
                   + elecGrid + internet + vehicles + numHHmem + DISTRICT_STR,
                   control = partykit::ctree_control(alpha=0.001,
                                                     minbucket = 3000),
                    data = LFS_train, weights = Weight)
```

- ctree() function estimates a regression tree
- We need: formula (Y and Xs)

```
> print(bank_tree)

Model formula:
any_bank_account ~ urban + toiletPoor + numDep + elecGrid + internet +
    vehicles + numHHmem + DISTRICT_STR

Fitted party:
[1] root
|   [2] vehicles in 0
|   |   [3] internet in 0
|   |   |   [4] DISTRICT_STR in Corozal, Belize
|   |   |   |   [5] numDep <= 0: 0 (w = 3213.3, err = 41.6%)
|   |   |   |   [6] numDep > 0
|   |   |   |   |   [7] urban in 0: 1 (w = 3984.2, err = 33.1%)
|   |   |   |   |   [8] urban in 1: 1 (w = 3343.6, err = 44.3%)
|   |   |   [9] DISTRICT_STR in Orange Walk, Cayo, Stann Creek, Toledo
|   |   |   |   [10] elecGrid in 0: 0 (w = 3020.9, err = 36.7%)
|   |   |   |   [11] elecGrid in 1
|   |   |   |   |   [12] numDep <= 1: 0 (w = 4125.4, err = 50.0%)
|   |   |   |   |   [13] numDep > 1: 0 (w = 5106.5, err = 44.1%)
|   |   [14] internet in 1
|   |   |   [15] urban in 0
|   |   |   |   [16] DISTRICT_STR in Corozal, Stann Creek: 1 (w = 4246.4, err = 35.8%)
|   |   |   |   [17] DISTRICT_STR in Orange Walk, Belize, Cayo, Toledo: 0 (w = 4873.0, err = 48.3%)
|   |   |   [18] urban in 1
|   |   |   |   [19] DISTRICT_STR in Corozal, Orange Walk, Belize, Toledo: 1 (w = 6043.5, err = 22.0%)
|   |   |   |   [20] DISTRICT_STR in Cayo, Stann Creek: 1 (w = 5014.8, err = 41.1%)
|   [21] vehicles in 1
|   |   [22] DISTRICT_STR in Corozal, Belize, Stann Creek
|   |   |   [23] numHHmem <= 3: 1 (w = 9371.0, err = 11.4%)
|   |   |   [24] numHHmem > 3: 1 (w = 7836.4, err = 23.0%)
|   |   [25] DISTRICT_STR in Orange Walk, Cayo, Toledo
|   |   |   [26] numHHmem <= 4: 1 (w = 8056.8, err = 21.9%)
|   |   |   [27] numHHmem > 4: 1 (w = 4288.6, err = 39.9%)

Number of inner nodes:    13
Number of terminal nodes: 14
```

- Raw model output isn't the prettiest to read (but does show fitted model.)

Plot Fitted Regression Tree



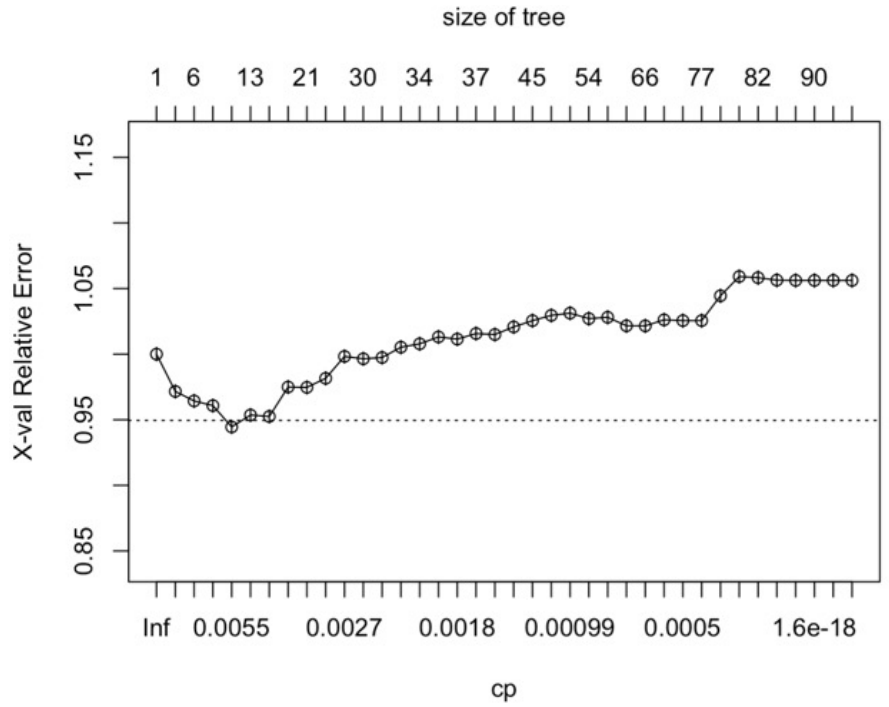
- First split is shown at top with p-value with null hypothesis that split doesn't increase class "separation". Number of vehicles is the most important variable, and p-value shows it improves model fit
- Second split is given by successive node. District access is the second most important
- For each "leaf" at the bottom the distribution plots for the outcome

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

Cross –Validate Our Titanic Data to Determine Optimal Tree Depth

```
library('rpart')
bank_rpart <- rpart(any_bank_account ~ urban  + toiletPoor + numDep
                    + elecGrid + internet + vehicles + numHHmem + DISTRICT_STR,
                    data = LFS_train, weights = Weight,
                    method = "class",control = list(cp = 0,
                                minsplit = 10,
                                maxdepth = 10))
```



Bagging

ENTERTAINMENT

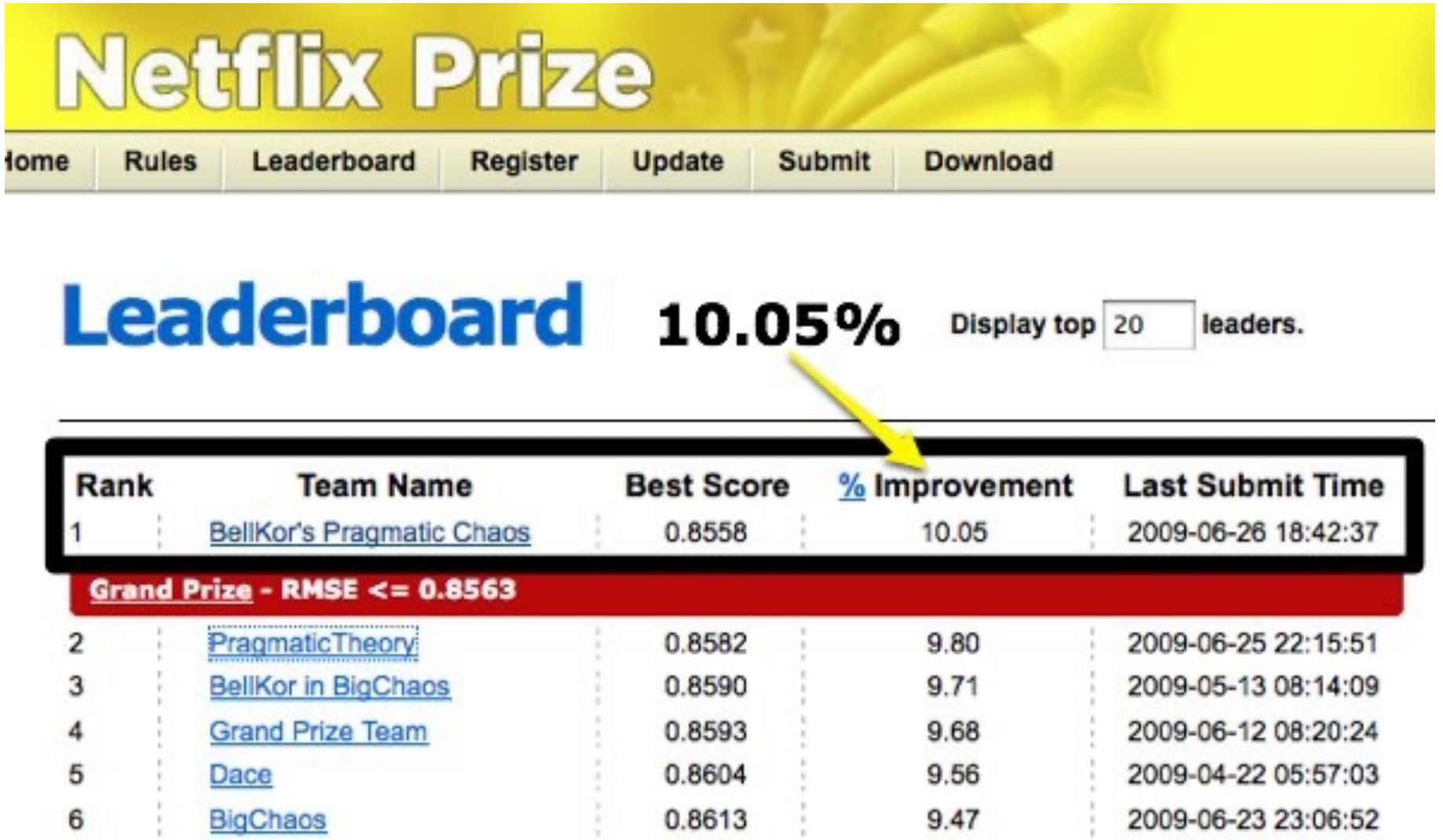
The Netflix Prize: How a $1 Million Contest Changed Binge-Watching Forever

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.

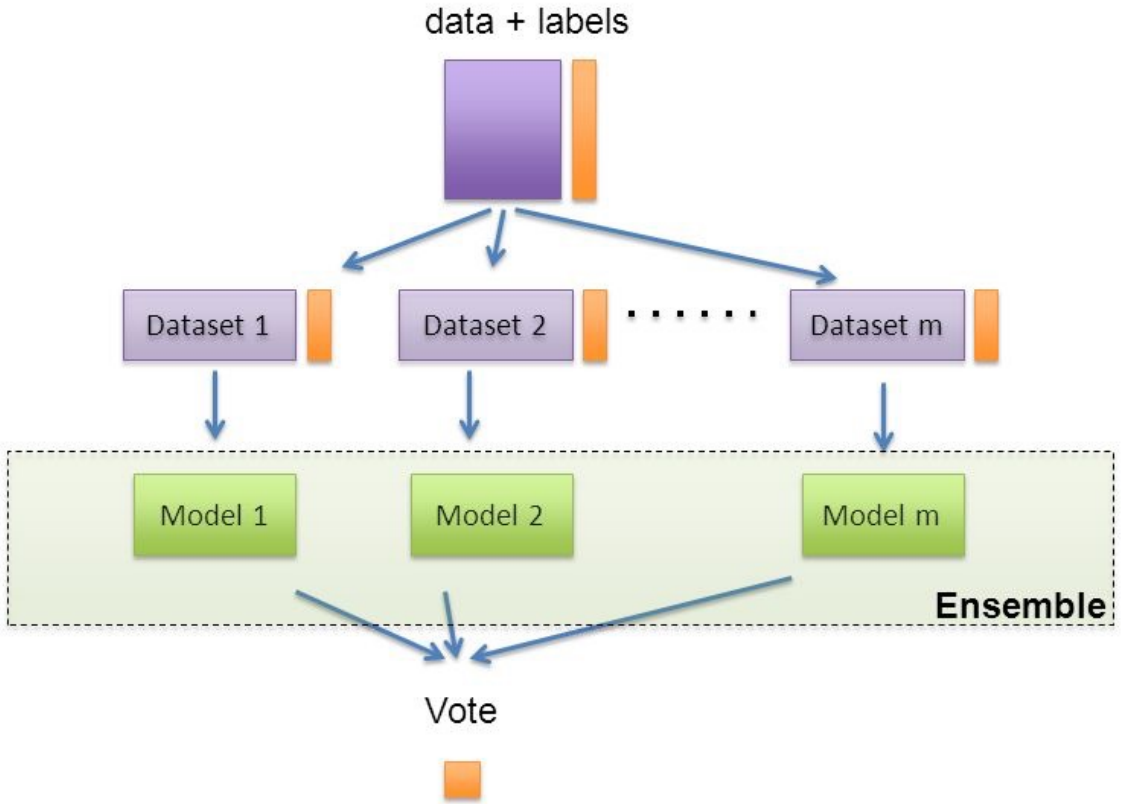
https://www.thrillist.com/entertainment/nation/the-netflix-prize

Netflix prize



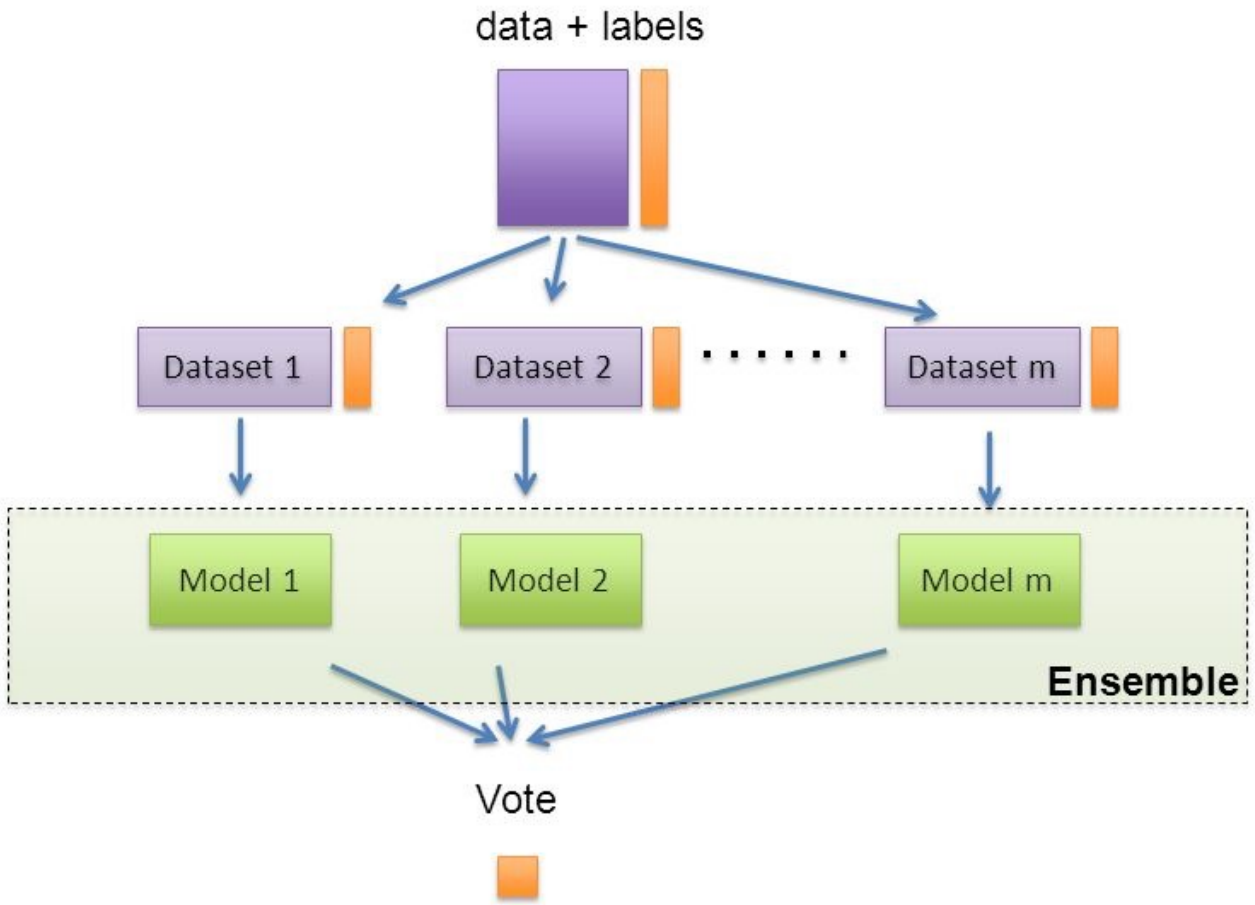
Rank	Team Name	Best Score	% Improvement	Last Submit Time
1	BellKor's Pragmatic Chaos	0.8558	10.05	2009-06-26 18:42:37
Grand Prize - RMSE <= 0.8563				
2	PragmaticTheory	0.8582	9.80	2009-06-25 22:15:51
3	BellKor in BigChaos	0.8590	9.71	2009-05-13 08:14:09
4	Grand Prize Team	0.8593	9.68	2009-06-12 08:20:24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

1. Generate B bootstrap training datasets
2. Train method on the b-th bootstrapped data set to obtain $\hat{f}^*(x)$ the prediction model built using bootstrap sample b
3. Repeat for every bootstrap sample, resulting in b models, and b sets of predictions
4. Once all bootstrap samples have models, 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

Out-of-bag error

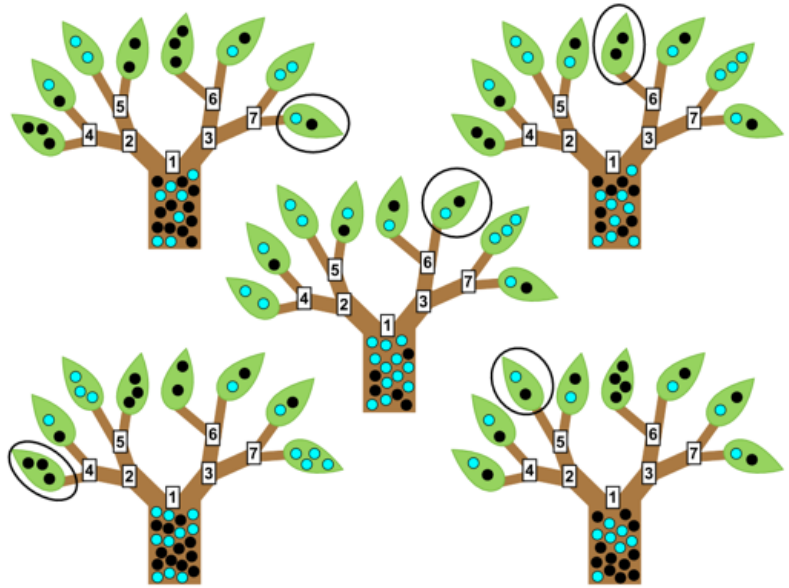
- Recall that for each bootstrapped sample *b* is composed of a subset of the total training data
- For each sample, the data not used to fit the model is referred to as **out-of-bag (OOB) observations**
- We can better approximate out of sample error by only using out-of-bag observations for model validation

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

Random Forests

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

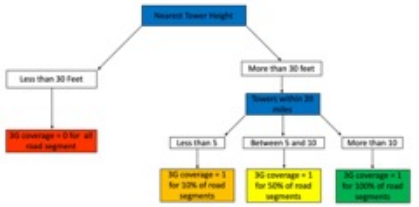
Bootstrap b

Bootstrap 1000

Tree 1

Tree b

Tree 1000



Vote if household is poor

Vote if household is poor

Vote if household is poor

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

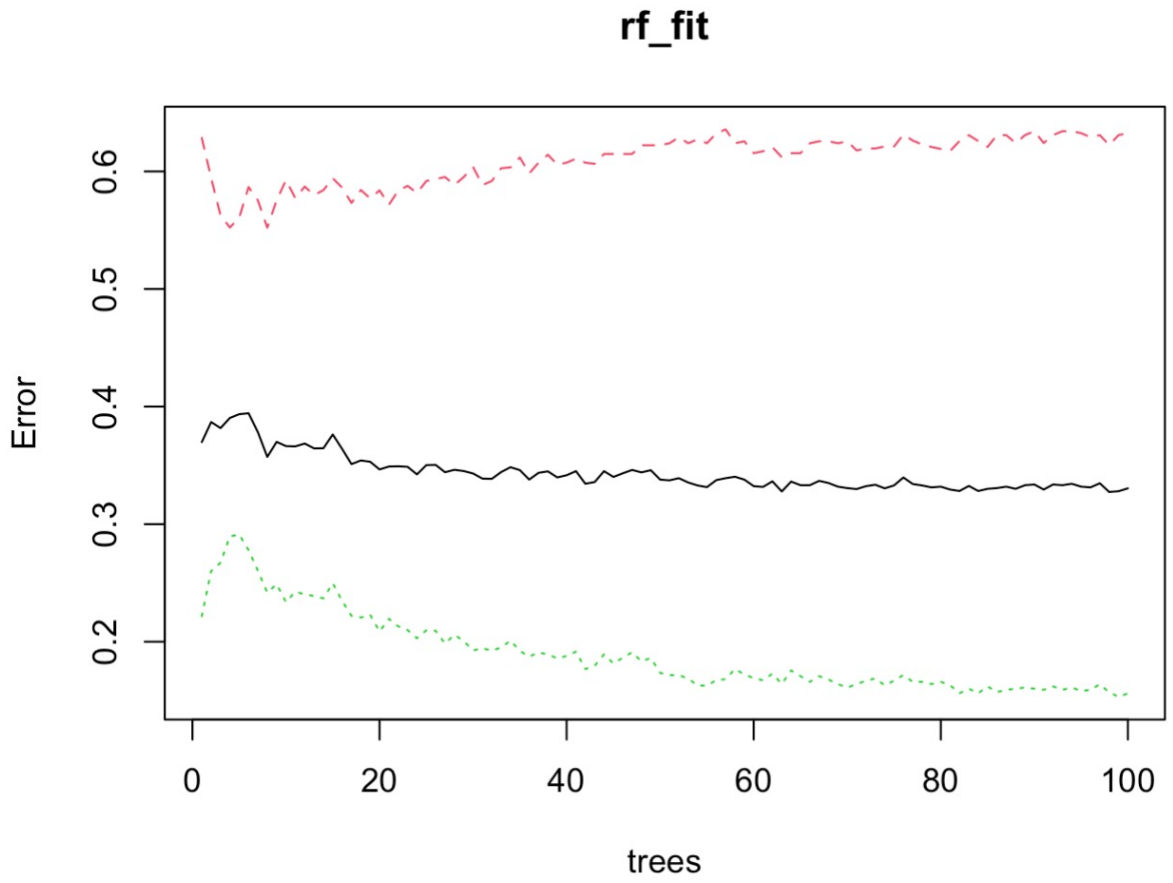
```
rf_fit <- randomForest(any_bank_account ~ urban + tenureTypeOwn + outerWallsPoor
                       + toiletPoor + elecGrid + bedrooms + aircon + fridges
                       + micros + washers + stereos + DVDplayers + TVs +
                         cellphones + computers + vehicles + cable +
                         internet + numHHmem + numDep + numChildren,
                       type = classification,
                       data = LFS_train,
                       mtry = 3,
                       weights = LFS_train$Weight,
                       na.action = na.roughfix,
                       ntree = 100,
                       importance = TRUE)
```

- 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

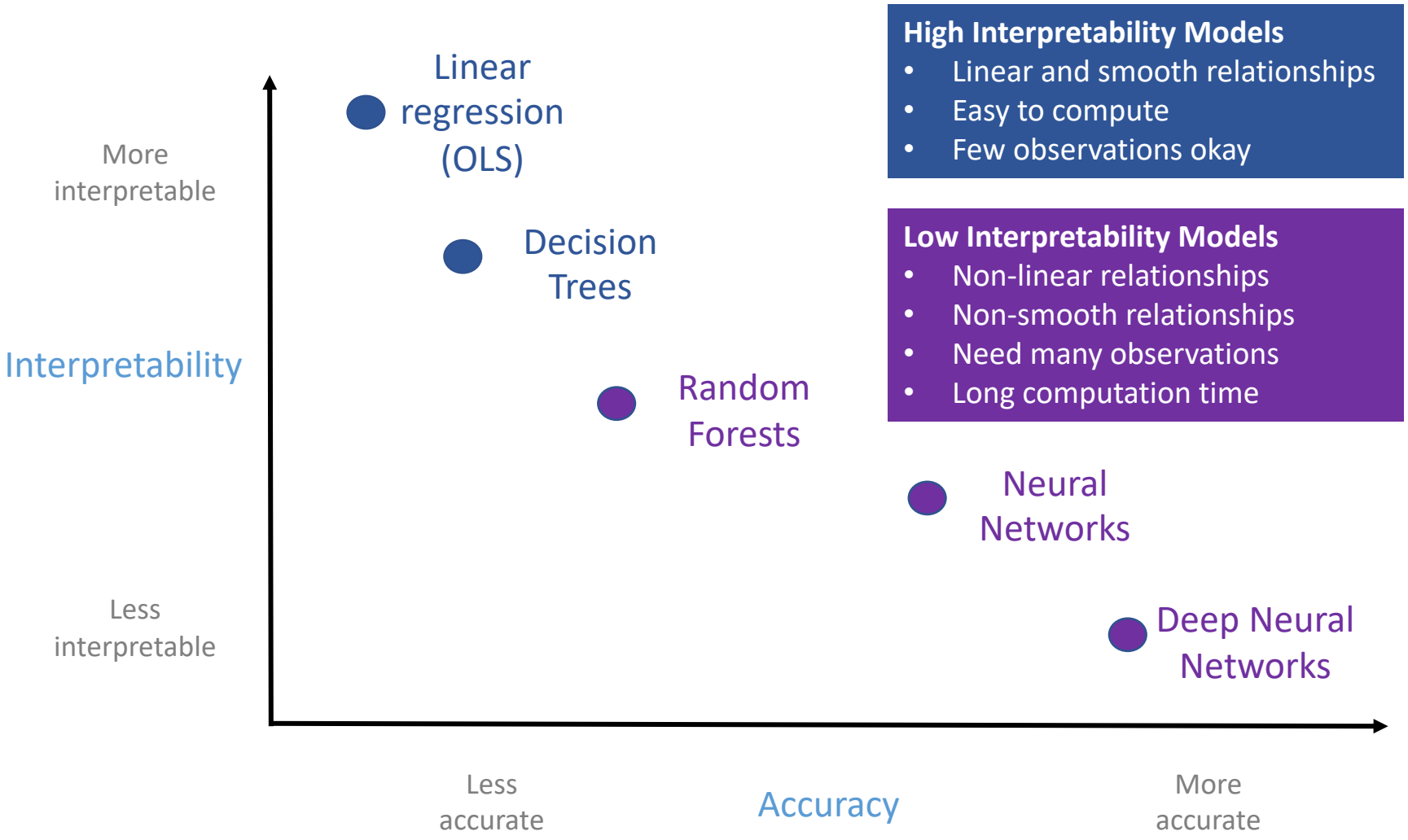
```
> print(rf_fit)

Call:
 randomForest(formula = any_bank_account ~ urban + tenureTypeOwn +      outerWallsPoor + toiletPoor + e
lecGrid + bedrooms + aircon +      fridges + micros + washers + stereos + DVDplayers + TVs +      cellp
hones + computers + vehicles + cable + internet + numHHmem +      numDep + numChildren, data = LFS_trai
n, type = classification,      mtry = 3, weights = LFS_train$Weight, ntree = 100, importance = TRUE,
   na.action = na.roughfix)
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 3

        OOB estimate of  error rate: 33.05%
Confusion matrix:
    0   1 class.error
0 219 377   0.6325503
1 161 871   0.1560078
```

plot() function against rf object – 100 trees, mtry = 3



- Green is error rate for positive class, red is error rate for negative class (not poor), and black is overall error rate (all out of bag error)
- Error seems to stabilize at 50-100 trees, so we only need around 100 trees for this prediction problem

What Is Model Interpretability?



- **Model interpretability:**
  - "the degree to which a human can understand the cause of a decision" (Miller, 2017)
- The higher the interpretability, the easier it is for someone to comprehend why a decision has been made

Why Do We Care About Explainable AI?

1. Consequential decisions may need human validation
2. Explaining the model may help us build better models
3. Explainable models might engender more trust

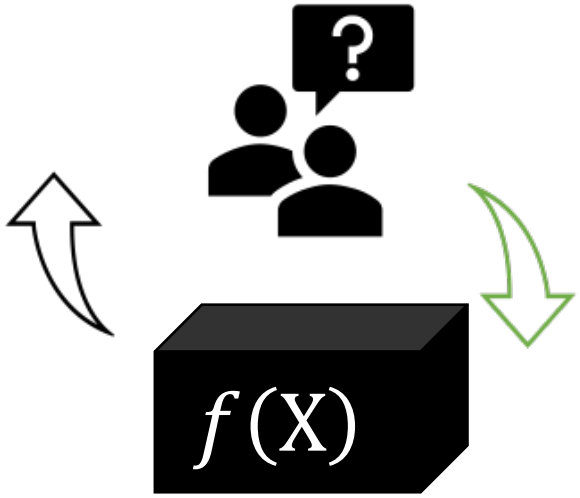
Opinion

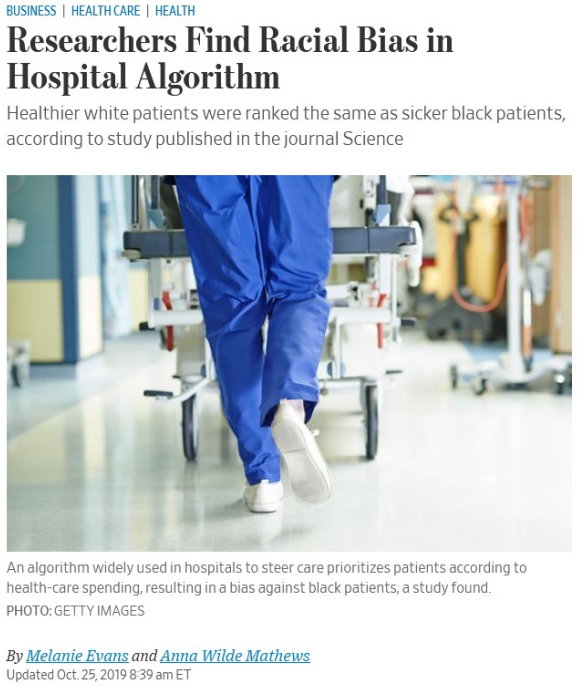
OP-ED CONTRIBUTOR

When a Computer Program Keeps You in Jail

By Rebecca Wexler

June 13, 2017





BUSINESS | HEALTH CARE | HEALTH

Researchers Find Racial Bias in Hospital Algorithm

Healthier white patients were ranked the same as sicker black patients, according to study published in the journal Science

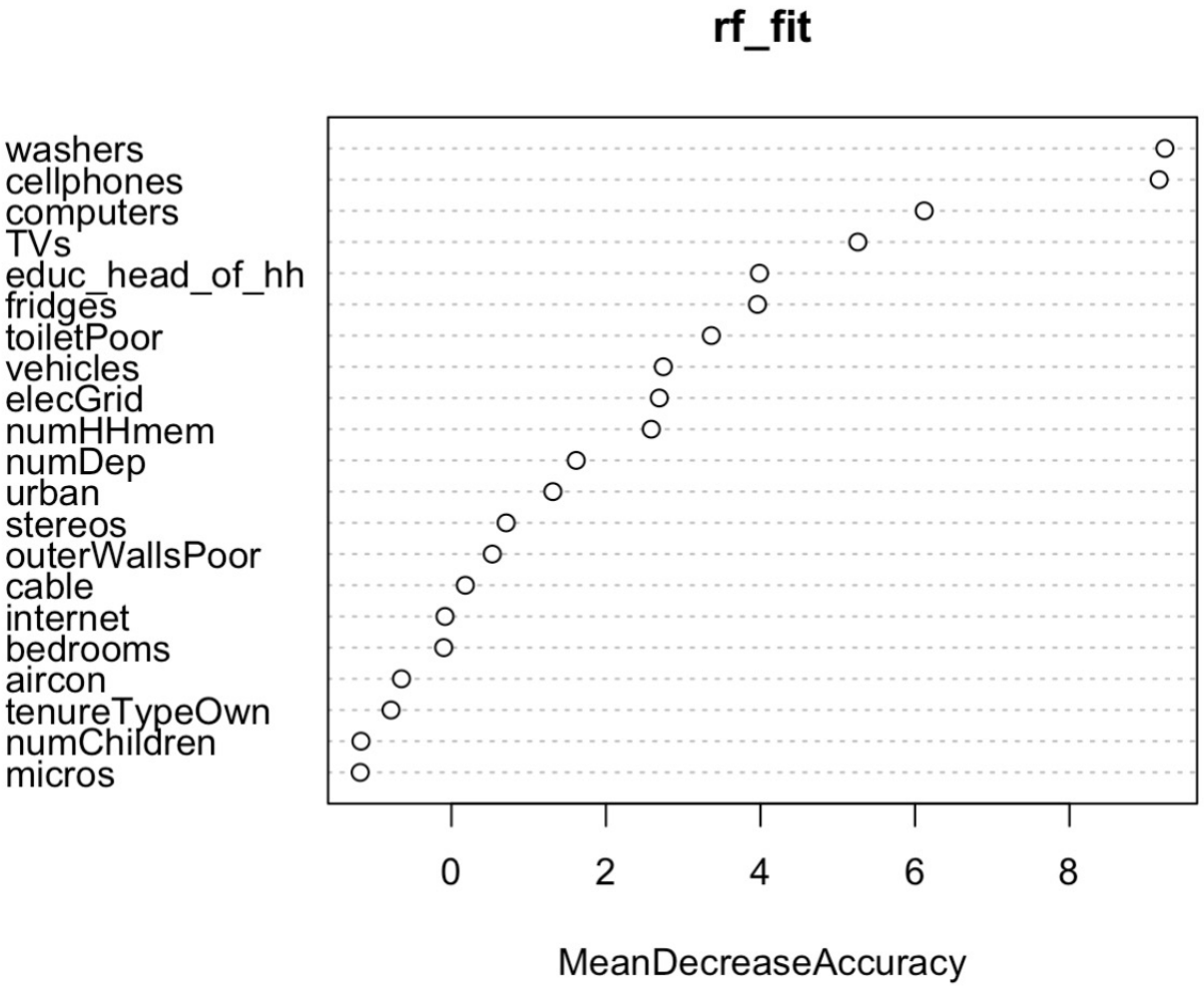
An algorithm widely used in hospitals to steer care prioritizes patients according to health-care spending, resulting in a bias against black patients, a study found.

PHOTO: GETTY IMAGES

By Melanie Evans and Anna Wilde Mathews

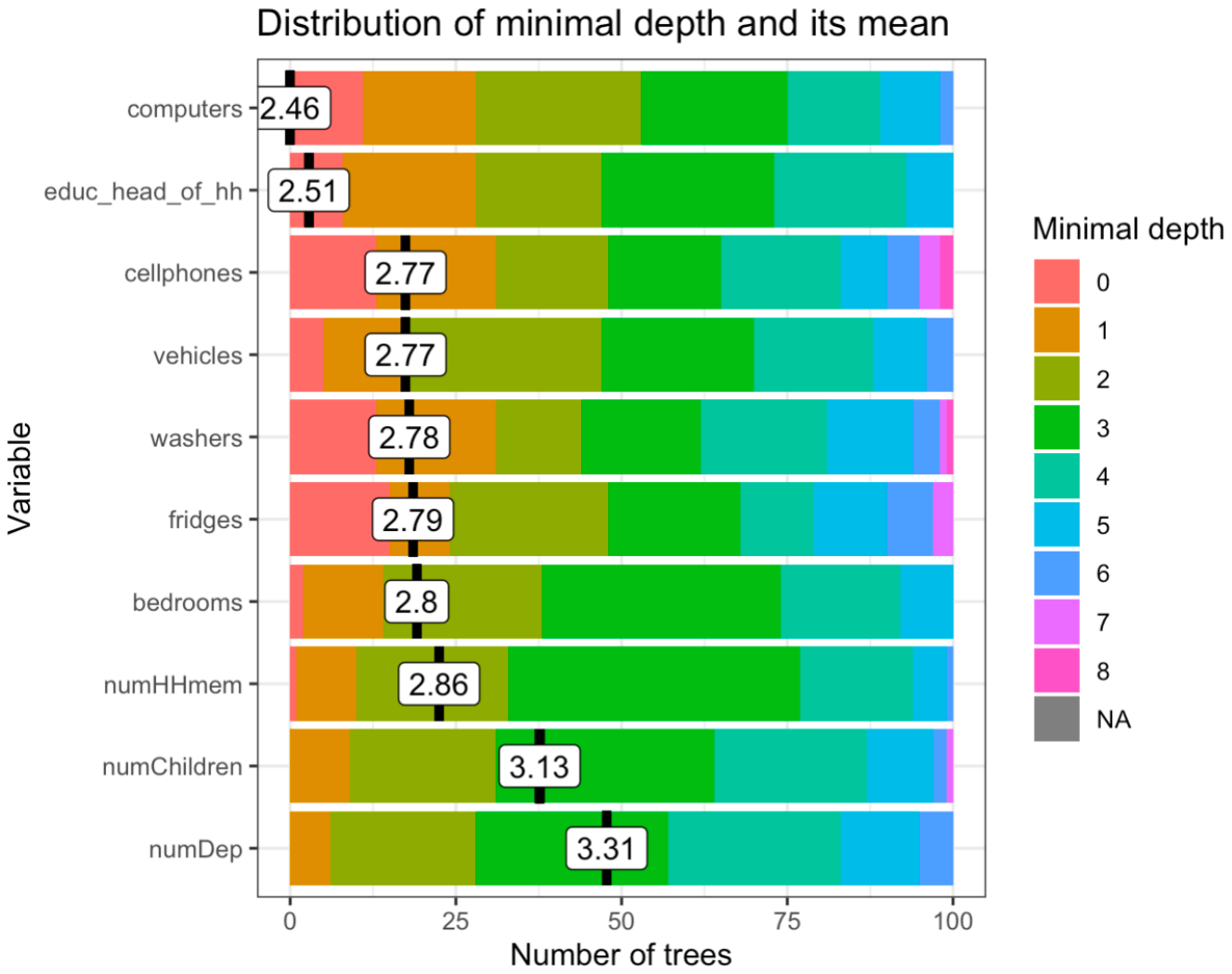
Updated Oct. 25, 2019 8:39 am ET

Which Variables Matter Most? Variable Importance



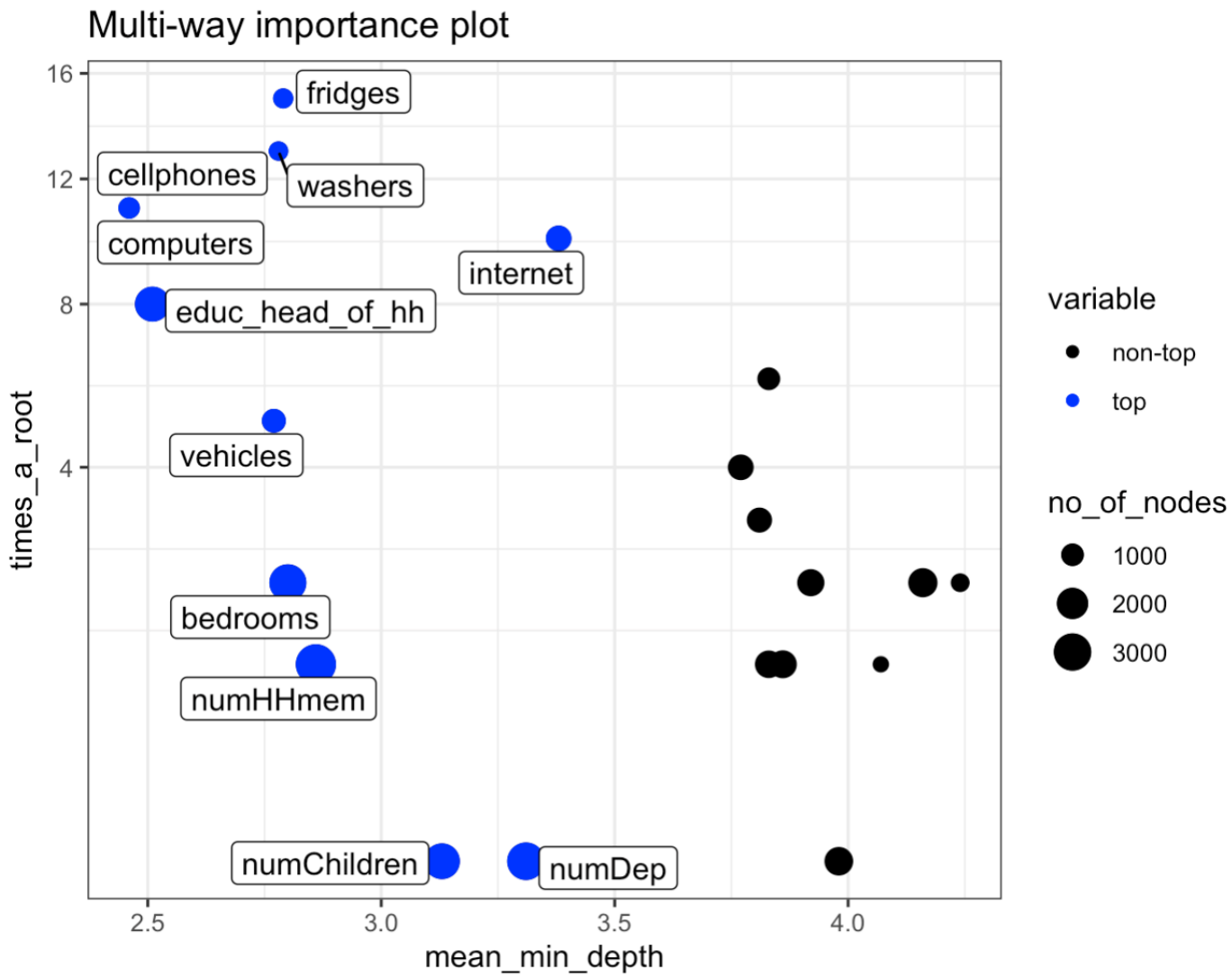
- Which variables are most important for the random forest model?
- One way of assessing importance is using **variable permutation**. This replaces a given variable (sequentially) with random noise (e.g. rnorm(., 0,1)) and re-estimates the model.
- Logic: more "important" variables result in worse models when these variables are absent (or are random noise)
- varImpPlot(rf_fit) plots these importance measures
- Mean educ is most important, followed by number of rooms, then dep rate

randomForestExplainer

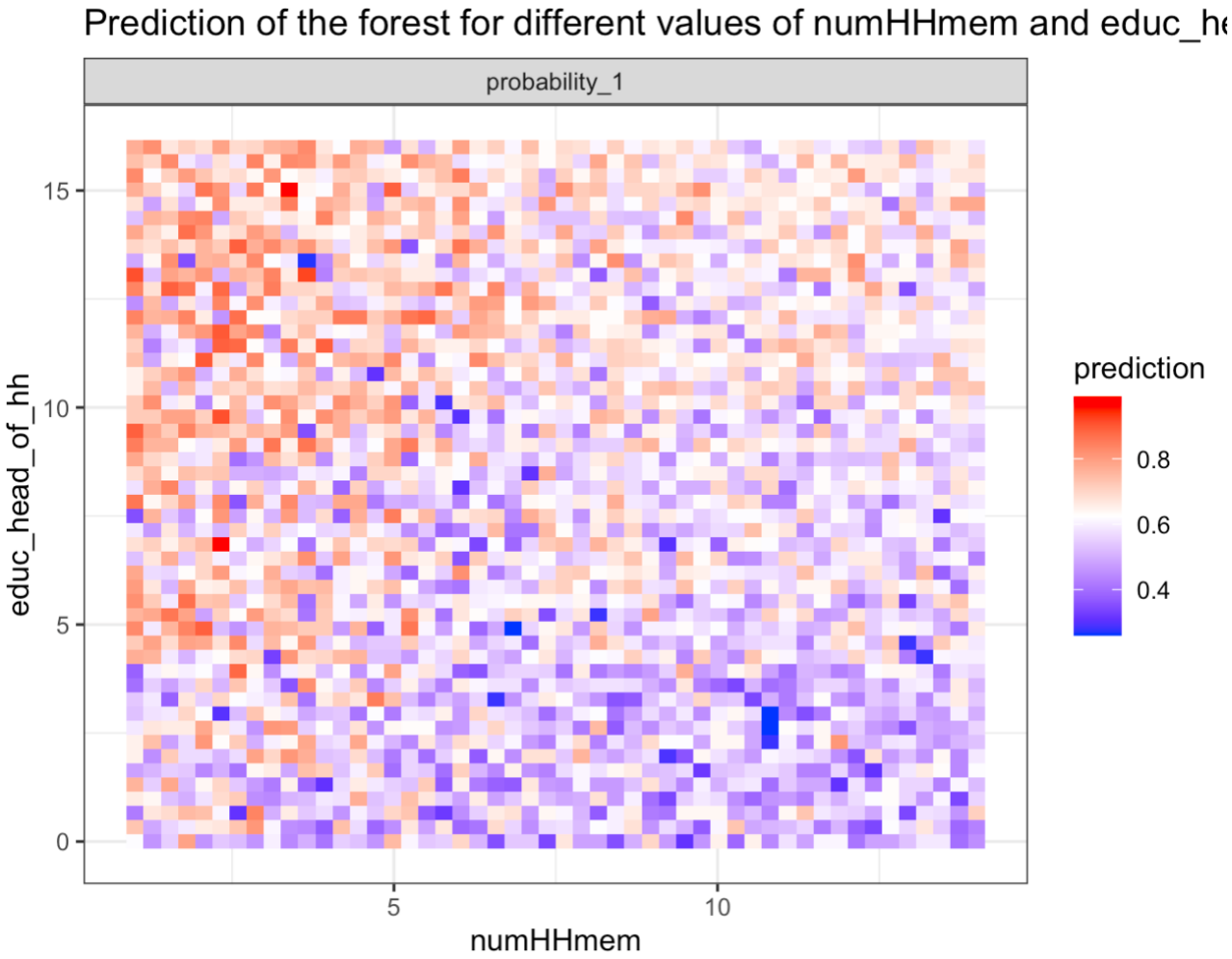


- Random forest explainer is a package that has many useful functions to explain and open up the random forest black box.
- This plots the average depth of trees in the ensemble that use these variables

randomForestExplainer



randomForestExplainer



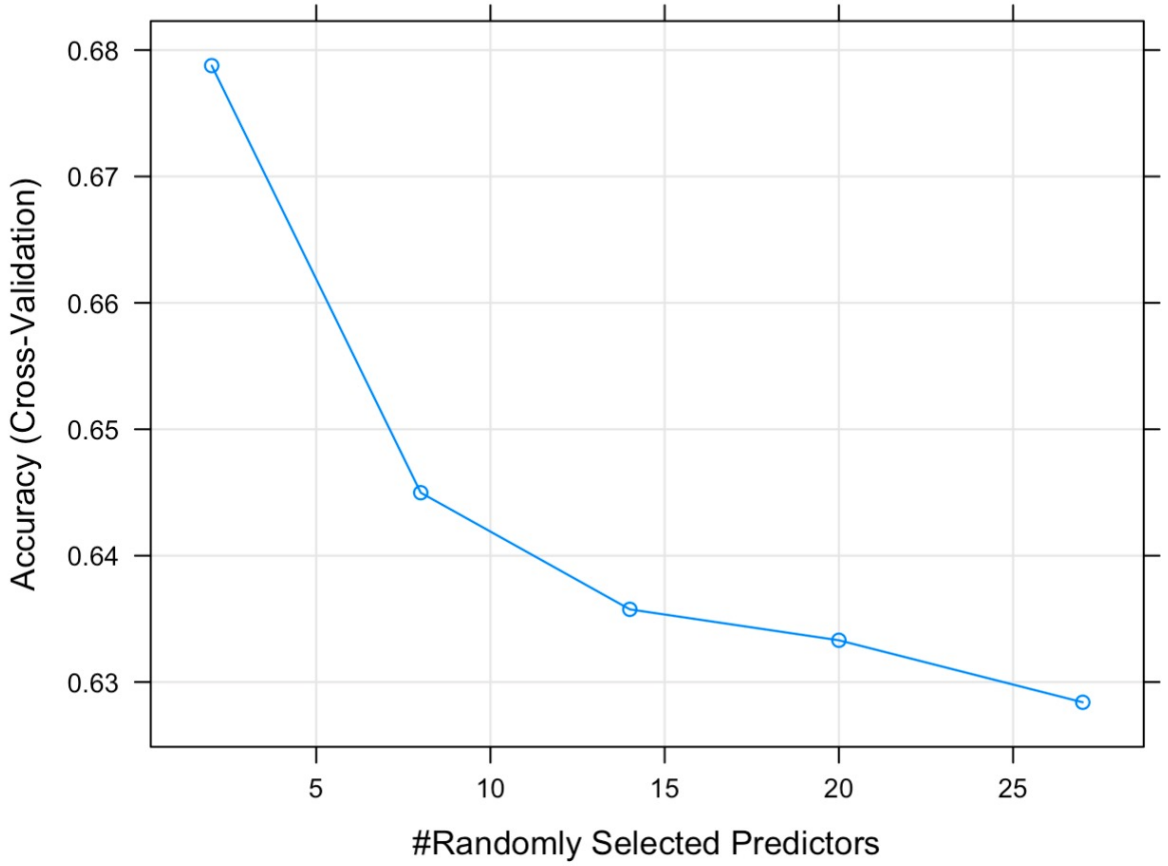
- We can see how the random forest classifies households with different characteristics of head of household education and number of children

Cross-Validating to Select mtry in caret

```
library('caret')

rf_caret <-
  train(any_bank_account ~ urban + tenureTypeOwn + outerWallsPoor
        + toiletPoor + elecGrid + bedrooms + aircon + fridges
        + micros + washers + stereos + TVs + cellphones +
          computers + vehicles + cable + educ_head_of_hh +
          internet + numHHmem + numDep + numChildren,
        data = LFS_train,
        weights = LFS_train$Weight,
        method = "rf",
        metric = "Accuracy",
        tuneLength = 5,
        trControl = trainControl(method = "cv",
                                 number = 5,
                                 verbose = TRUE))

plot(rf_caret)
```

- caret is a great machine learning package that holds many models.
- It handles many common machine learning tasks such as cross-validation to select optimal paramters

Cross-Validating to Select mtry in caret



- It seems we should set mtry = 2!