Class 18: Random Forests

MGSC 310

Prof. Jonathan Hersh

Class 18 Announcements

1. Quiz 7 Due Thursday @ midnight
2. Final project one sheet due November 10
3. Pset 5 due Tuesday, November 17 (posted)
   - *Addendum: J) and k) extra credit*
4. Almost all midterms graded
   - Avg: 86%

Data Analytics Association Guest Speaker

Annie Wang

JD Candidate, Yale

Formerly:

Director of Data Science, Warren for President

Director of Research and Analytics, Analyst Institute

Civis Analytics, Senior Applied Data Scientist

Tuesday, Nov 10 @ 7pm

Final Project: One-Page Outline Due Nov 10

- **Your final project should take a real-world data set, and estimate a series of predictive models against this dataset.**
- You should identify a business use-case for the prediction, and stimate at least three predictive models we covered in this class against the dataset.

November 10th – Due: students must upload to Canvas a one-page outline of their project. This outline should include:

a) identify a dataset you will use
b) the outcome you are trying to predict, and what variables you will use to predict it
c) motivation to your project -- as in the business or practical management use case of such a prediction
d) three methods you will use to analyze your question of interest
e) **the names of the students who will be part of your group (up to four)**

Where to Find Datasets?

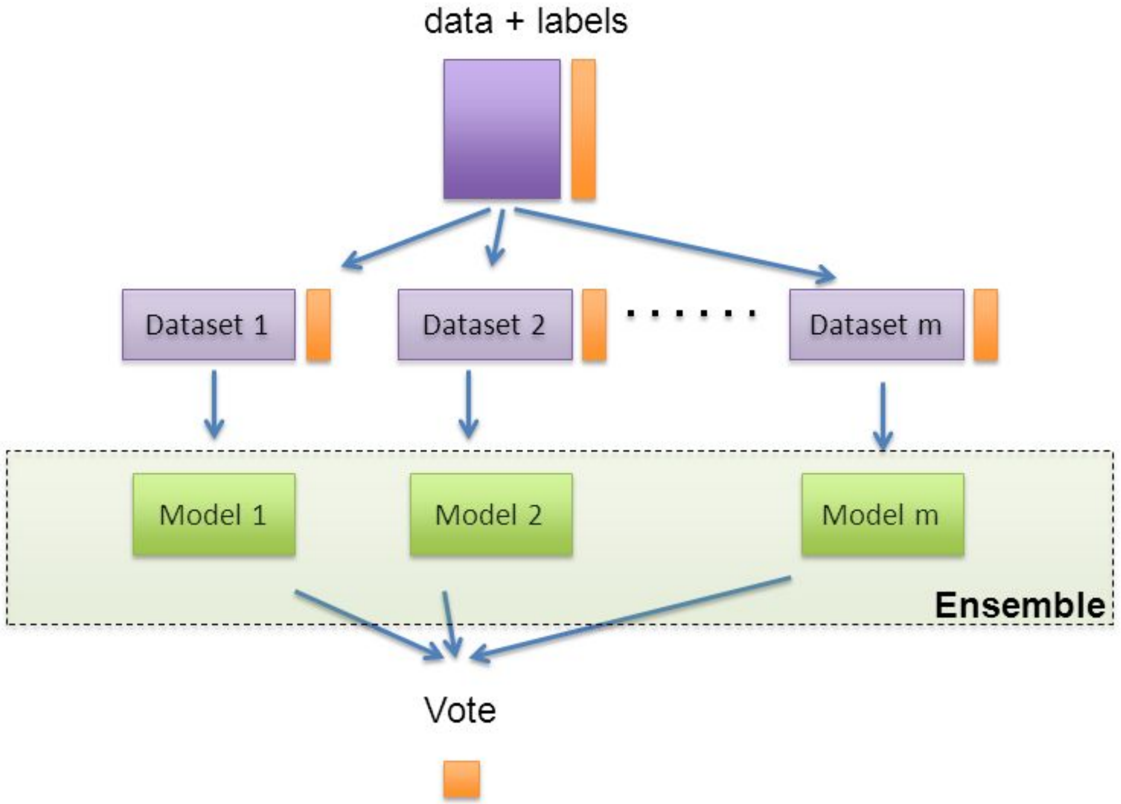
- Kaggle: https://www.kaggle.com/datasets
- Kaggle: https://www.kaggle.com/annavictoria/ml-friendly-public-datasets
- FiveThirtyEight https://data.fivethirtyeight.com/
- TidyTuesday: https://github.com/rfordatascience/tidytuesday
- UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets.php

Class 18: Outline

1. **Bagging**
2. Random Forests
3. Random Forest Lab

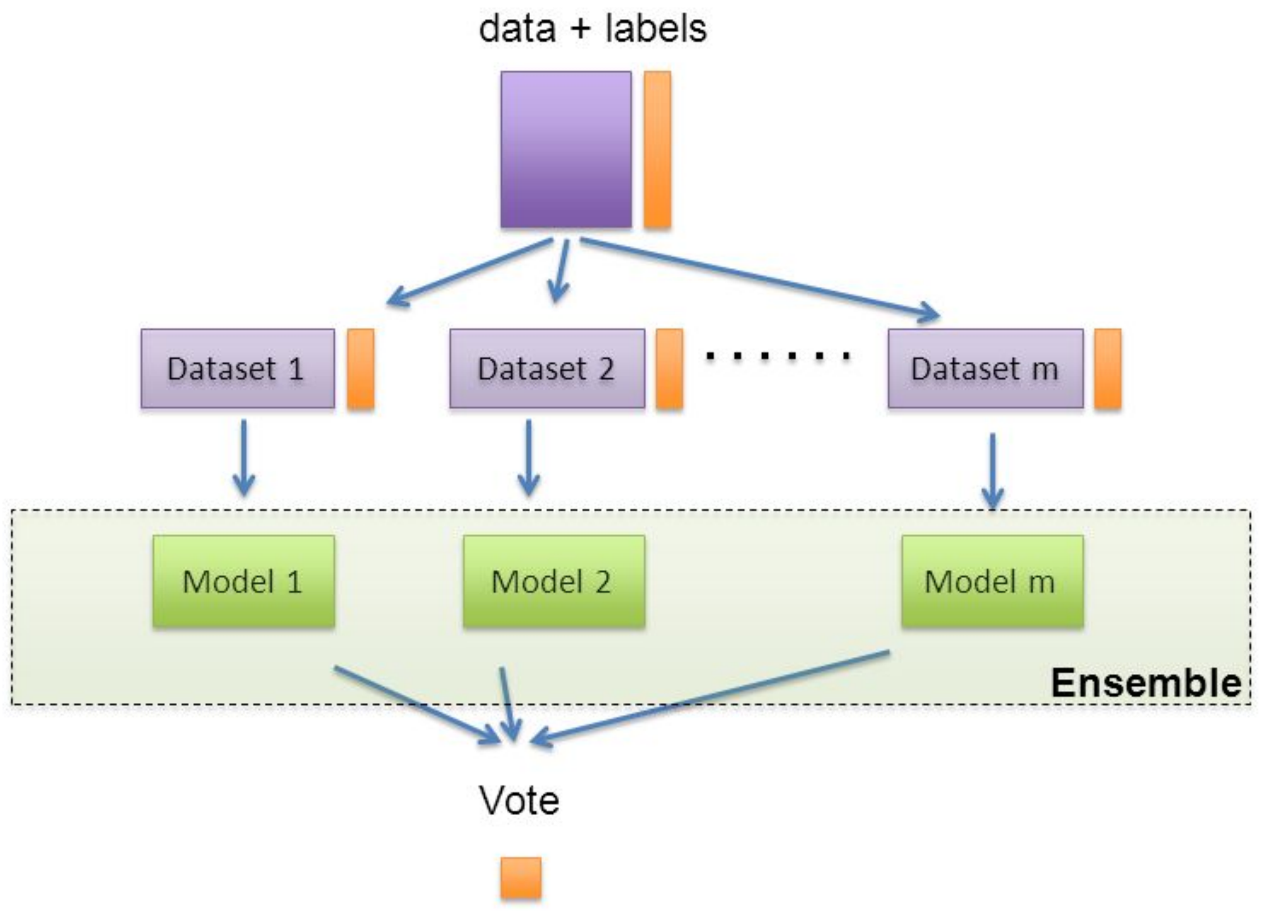
Netflix prize winners

NETFLIX
2009
DATE 09.21.09
PAY TO THE ORDER OF BellKor's Pragmatic Chaos
$ 1,000,000 00
AMOUNT ONE MILLION 00/100
FOR The Netflix Prize



- 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, $z_1, z_2, \dots, z_n$ 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

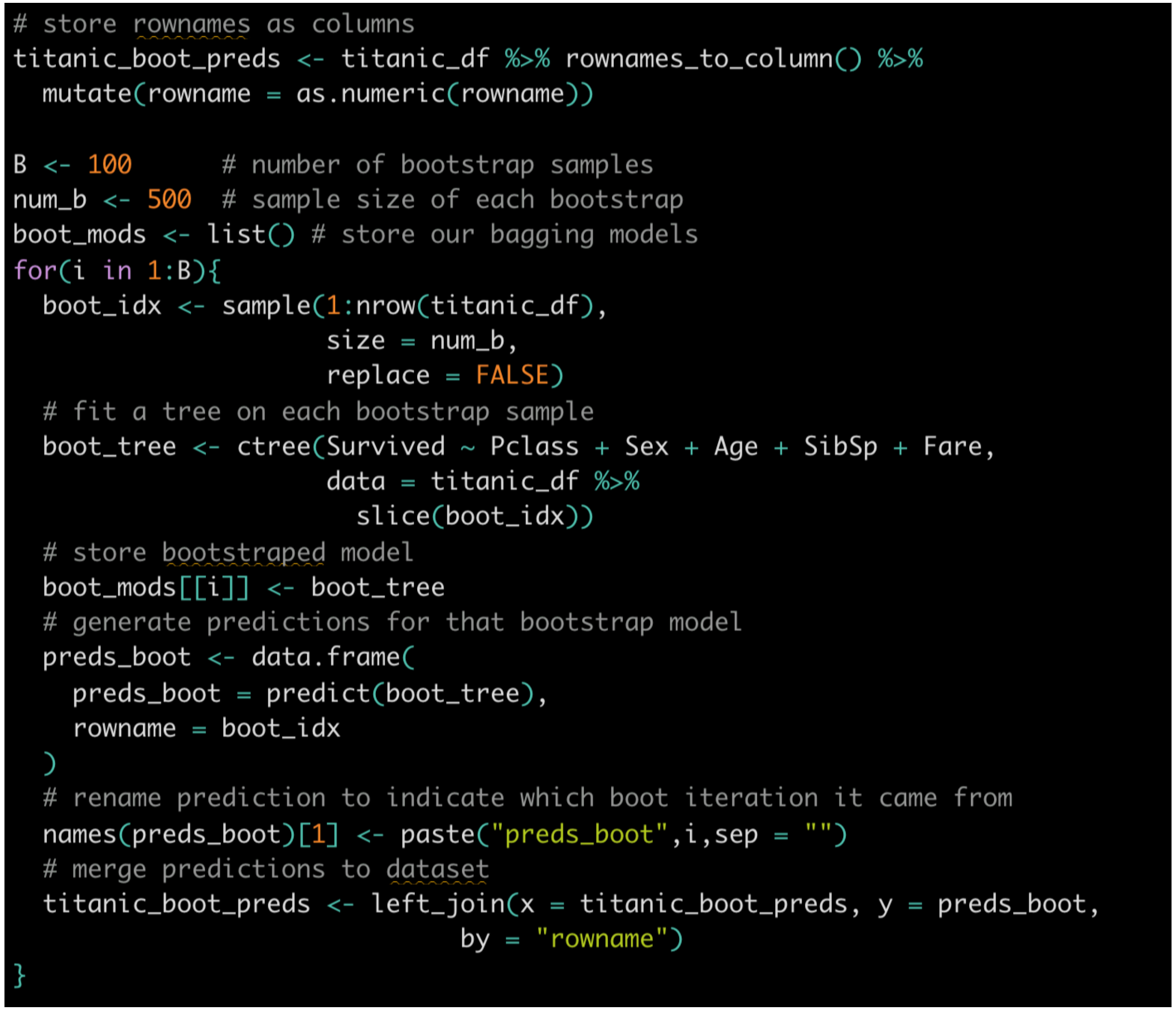
Bagging (Bootstrap Aggregation) Algorithm

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

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

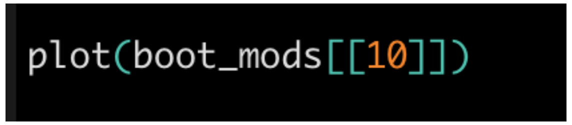
- Netflix prize intuition: average of many models will perform better than a single model

Bagging Model of Titanic Data

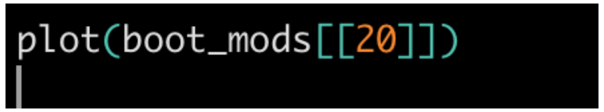


```r
# store rownames as columns
titanic_boot_preds <- titanic_df %>% rownames_to_column() %>%
  mutate(rowname = as.numeric(rowname))

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")
}
```

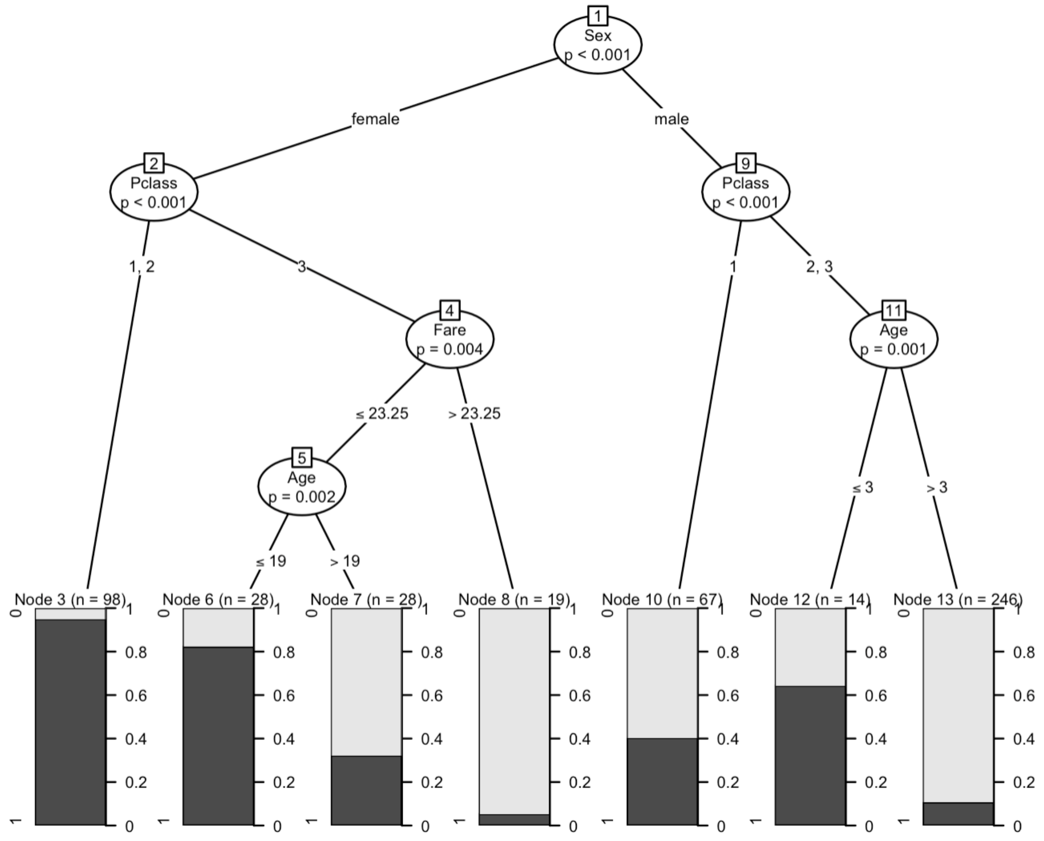
Examine Individual Tree Models In Bagging Aggregated Model

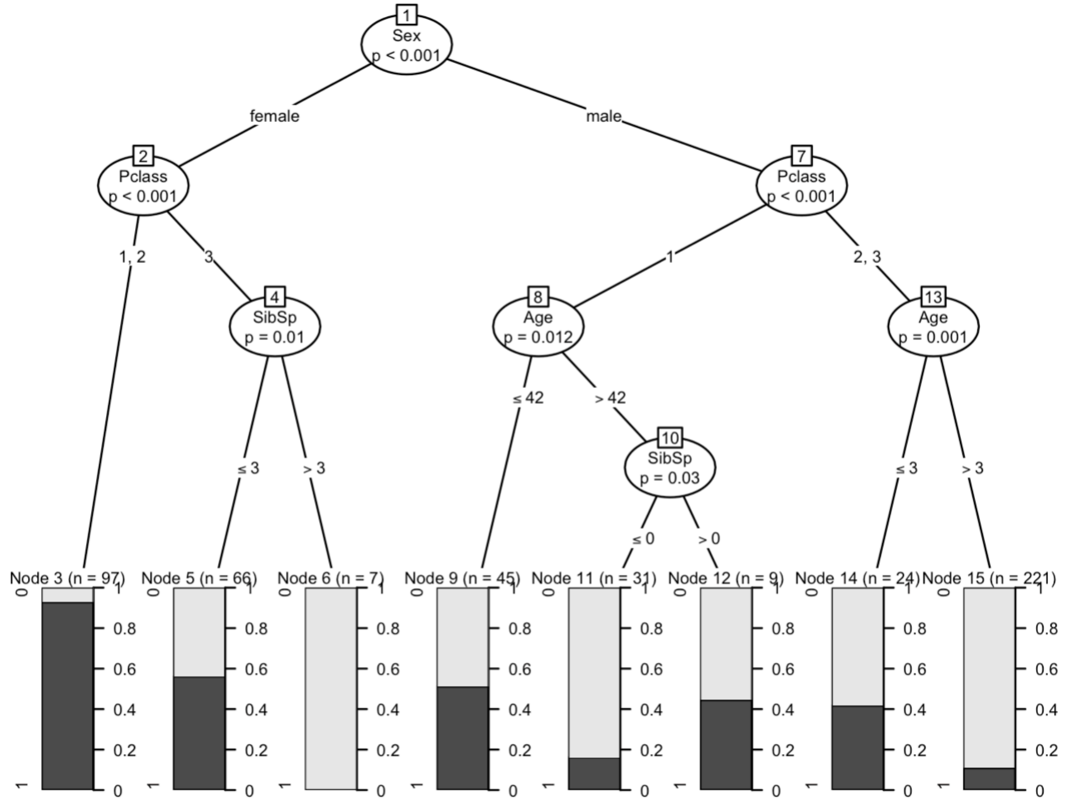


```
plot(boot_mods[[10]])
```

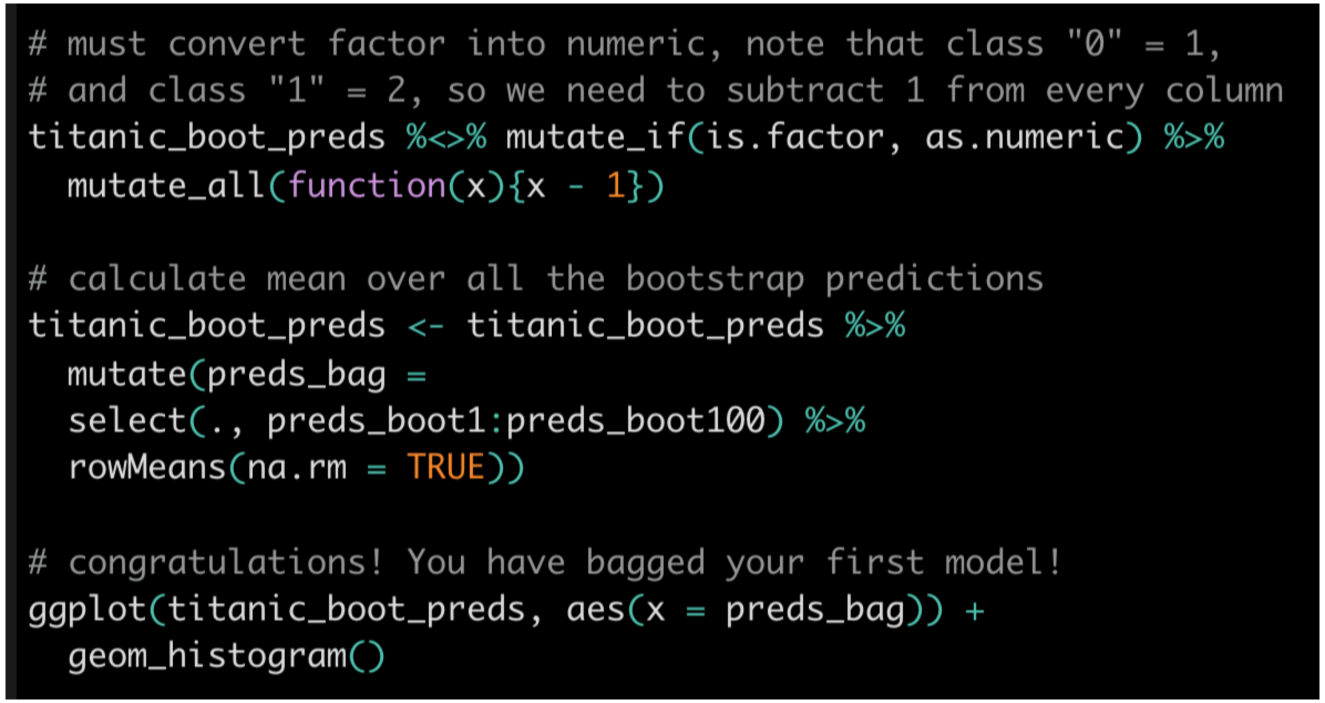


```
plot(boot_mods[[20]])
```

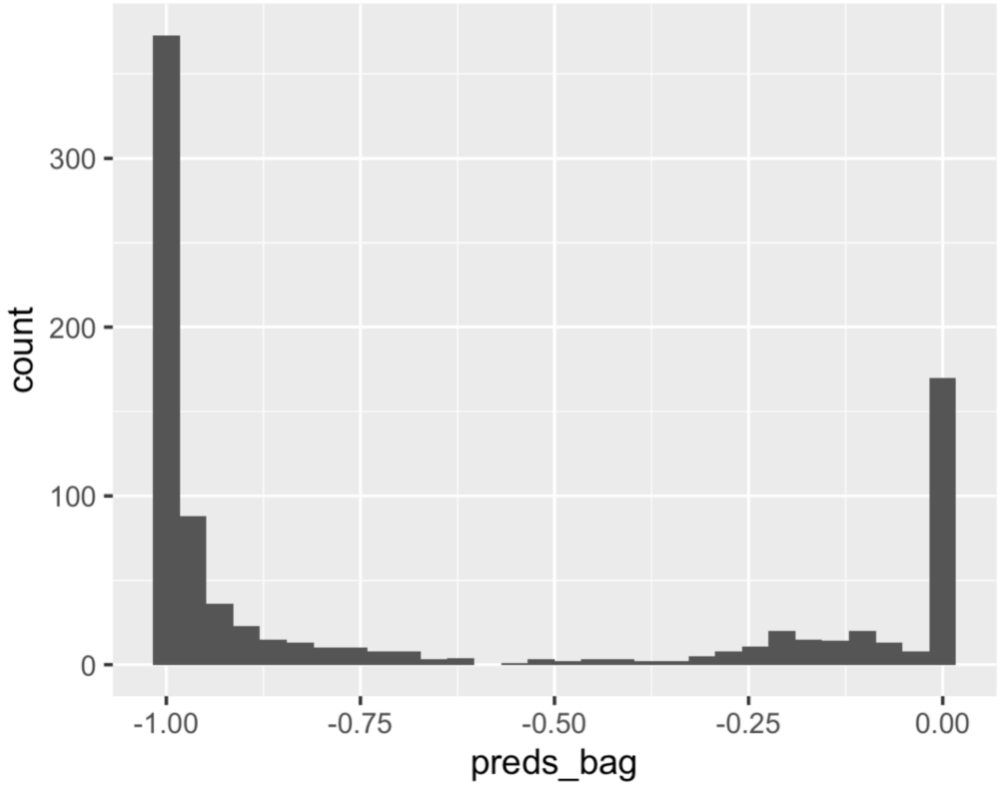




Row Means Over Columns to Average Predictions

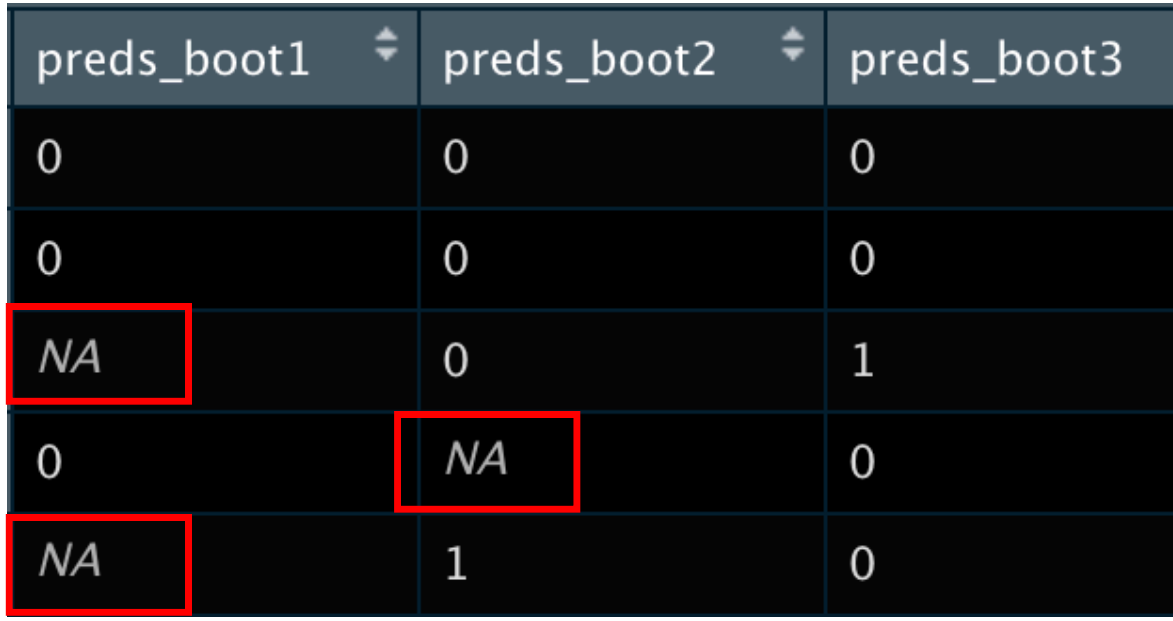


```
# must convert factor into numeric, note that class "0" = 1,
# and class "1" = 2, so we need to subtract 1 from every column
titanic_boot_preds %<>% mutate_if(is.factor, as.numeric) %>%
  mutate_all(function(x){x - 1})

# 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()
```



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



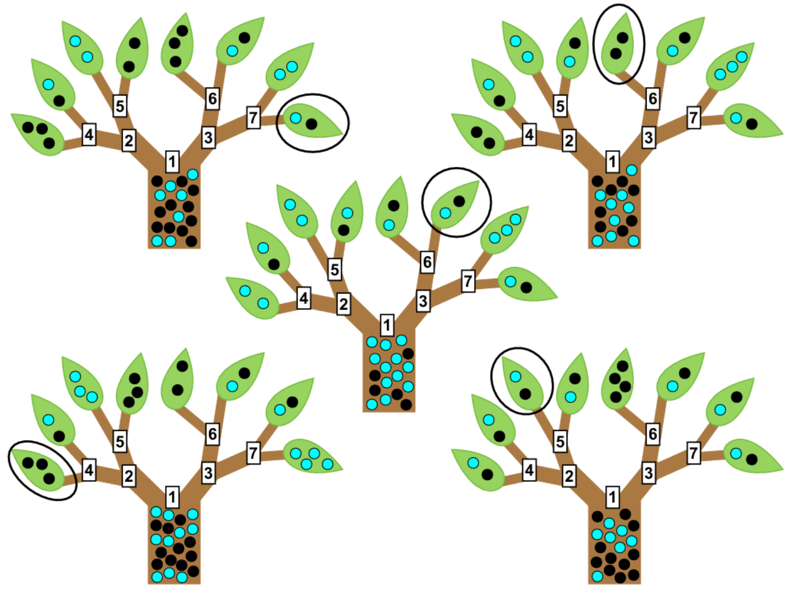
preds_boot1	preds_boot2	preds_boot3
0	0	0
0	0	0
NA	0	1
0	*NA*	0
NA	1	0

Class 18: Outline

1. Bagging
2. **Random Forests**
3. Random Forest Lab (Time Permitting)

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.



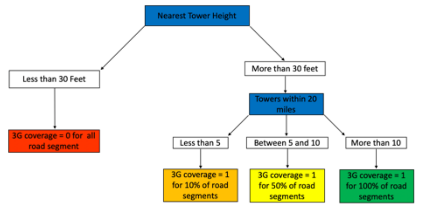
Random Forest Model: Many Decision Trees

B bootstrap sampes of data (~ 1000)

…

Bootstrap 1

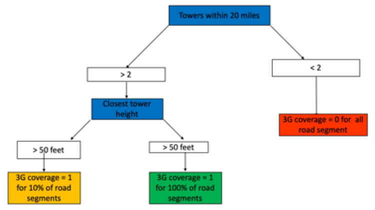
Tree 1


Nearest Tower Height
Less than 30 Feet
More than 30 feet
3G coverage = 0 for all road segment
Towers within 20 miles
Less than 5
Between 5 and 10
More than 10
3G coverage = 1 for 10% of road segments
3G coverage = 1 for 50% of road segments
3G coverage = 1 for 100% of road segments

Vote road i has 3G access = 0 or 1

Bootstrap b

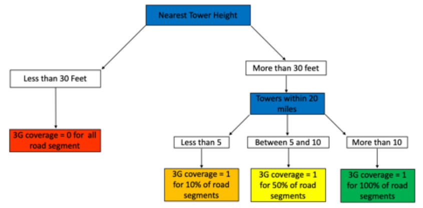
Tree b


Towers within 20 miles
> 2
< 2
Closest tower height
3G coverage = 0 for all road segment
> 50 feet
> 50 feet
3G coverage = 1 for 10% of road segments
3G coverage = 1 for 100% of road segments

Vote road i has 3G access = 0 or 1

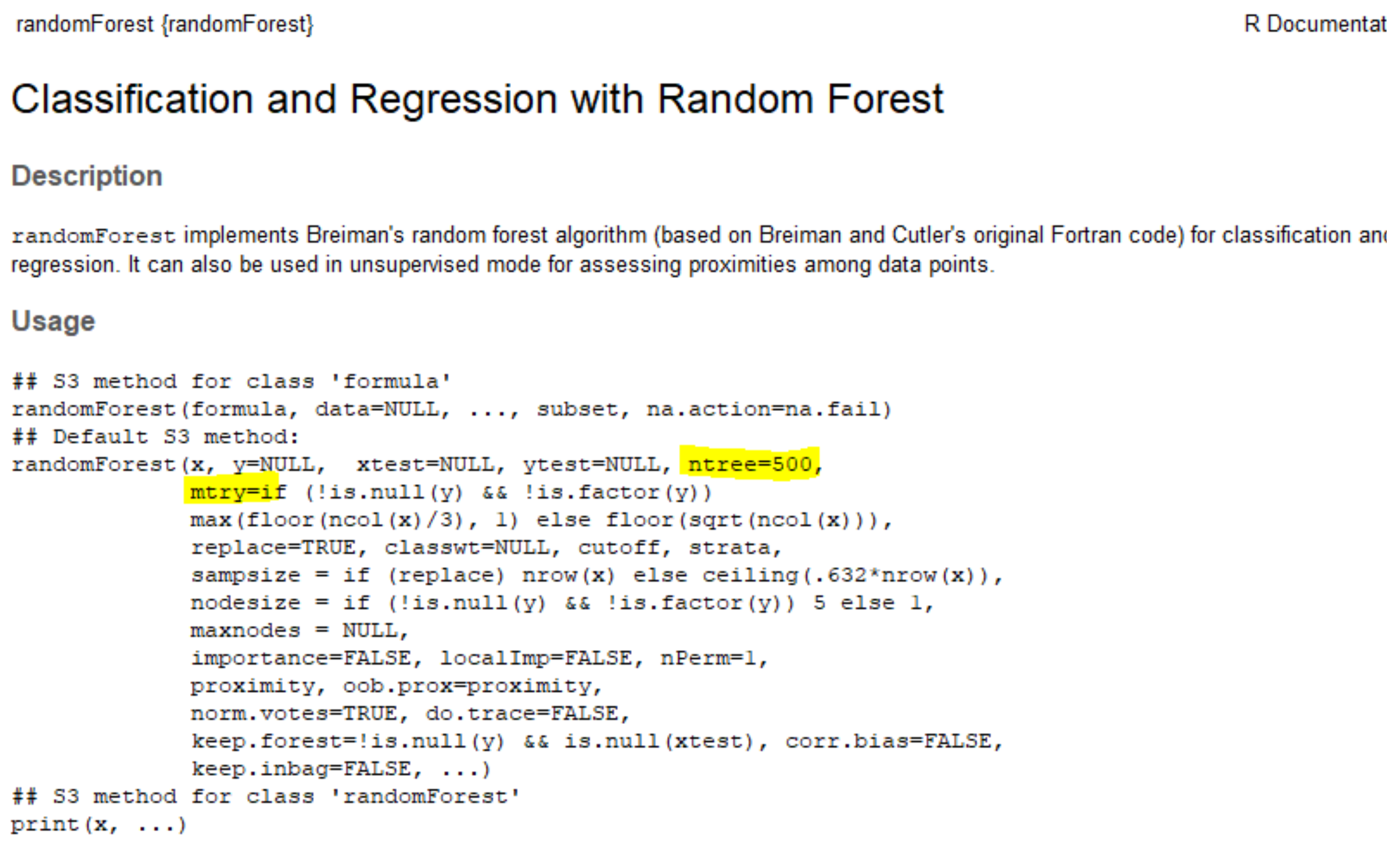
Bootstrap 1000

Tree 1000


Nearest Tower Height
Less than 30 Feet
More than 30 feet
3G coverage = 0 for all road segment
Towers within 20 miles
Less than 5
Between 5 and 10
More than 10
3G coverage = 1 for 10% of road segments
3G coverage = 1 for 50% of road segments
3G coverage = 1 for 100% of road segments

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

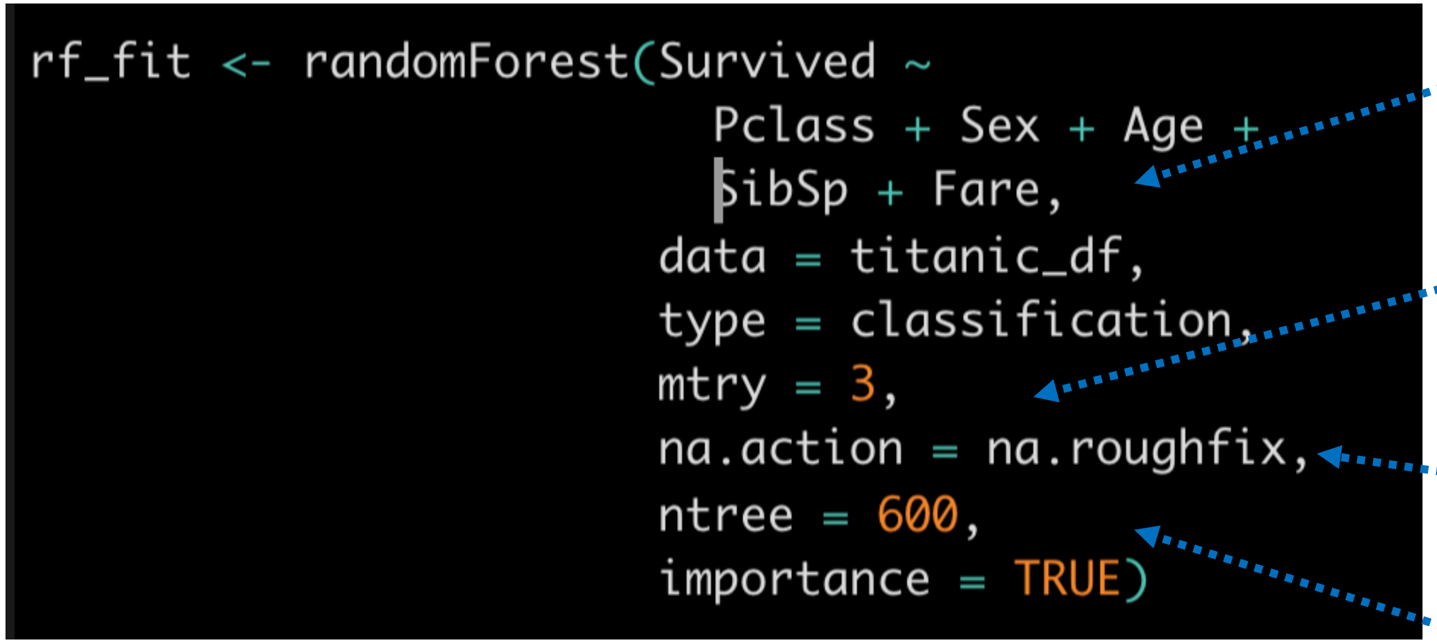
randomForest implements Breiman's random forest algorithm (based on Breiman and Cutler's original Fortran code) for classification an regression. It can also be used in unsupervised mode for assessing proximities among data points.

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

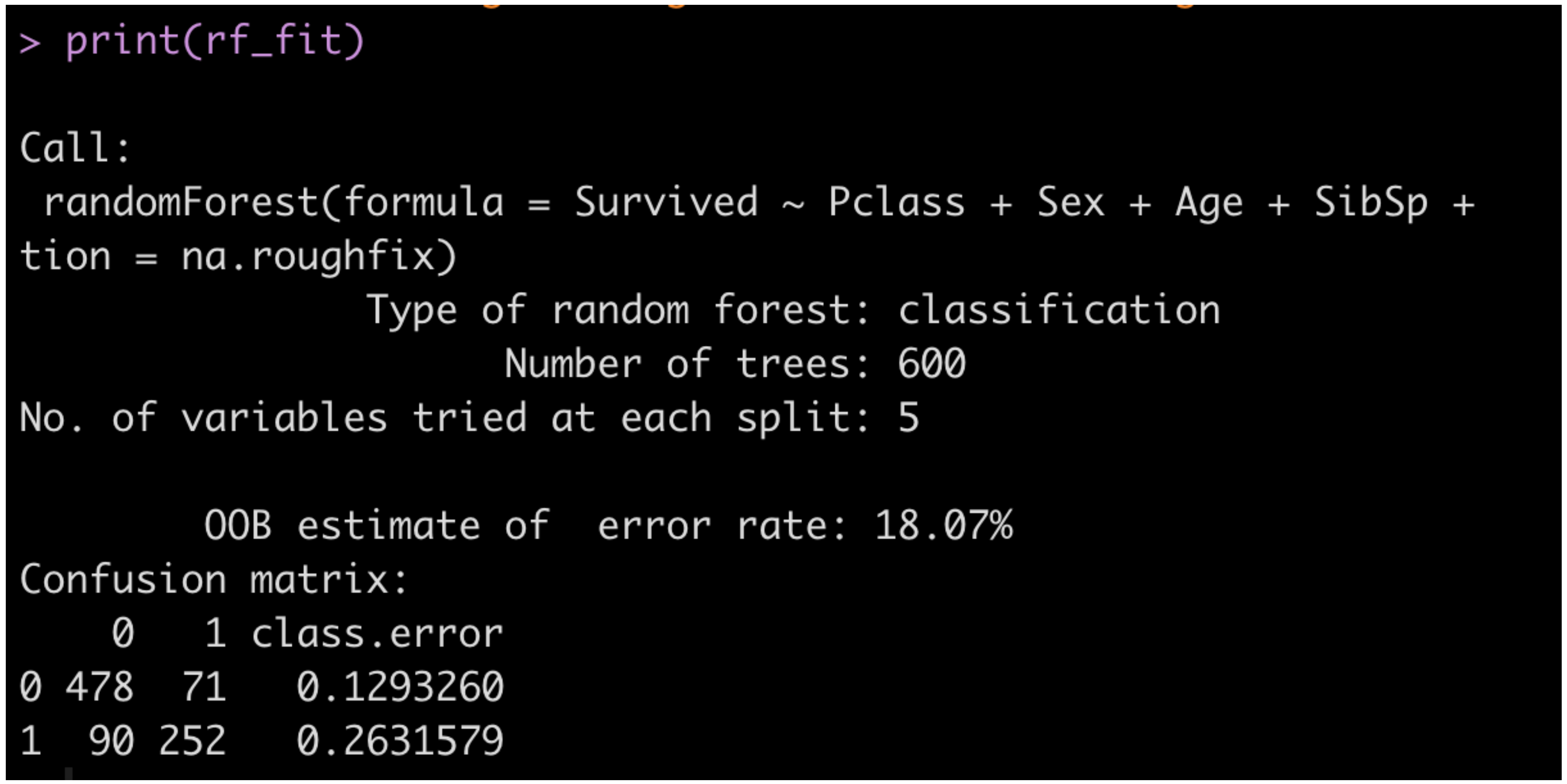
Estimating Random Forest Model Using randomForest() package



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

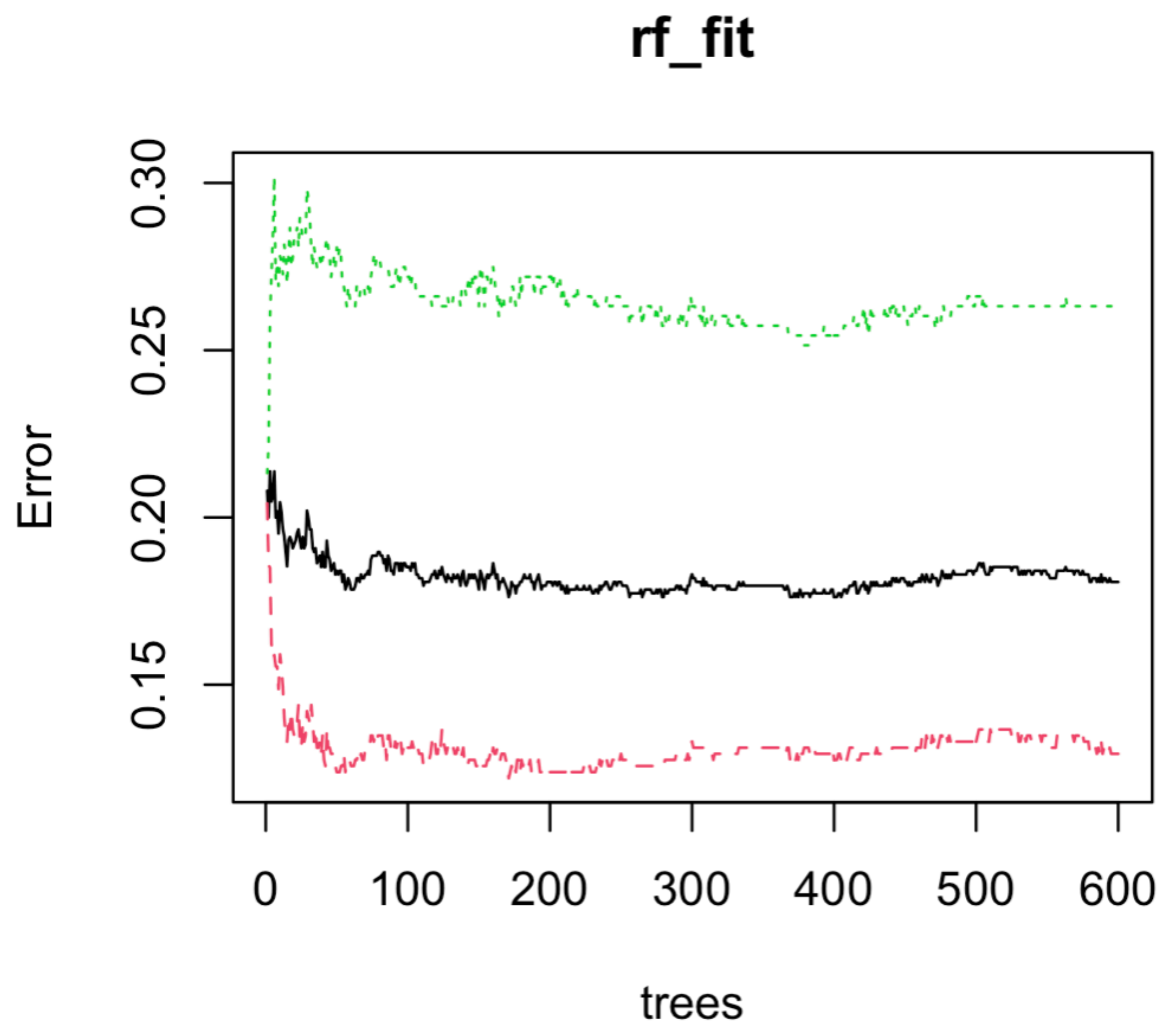
- Must specify formula and dataset to use per usual
- Additionally must specify "mtry" the number of variables to sample (randomly) for each node
- Missing values will cause an error and this rough fix replaces them with median values
- Ntree specifices the number of trees in the random forest

Random Forest Model Object



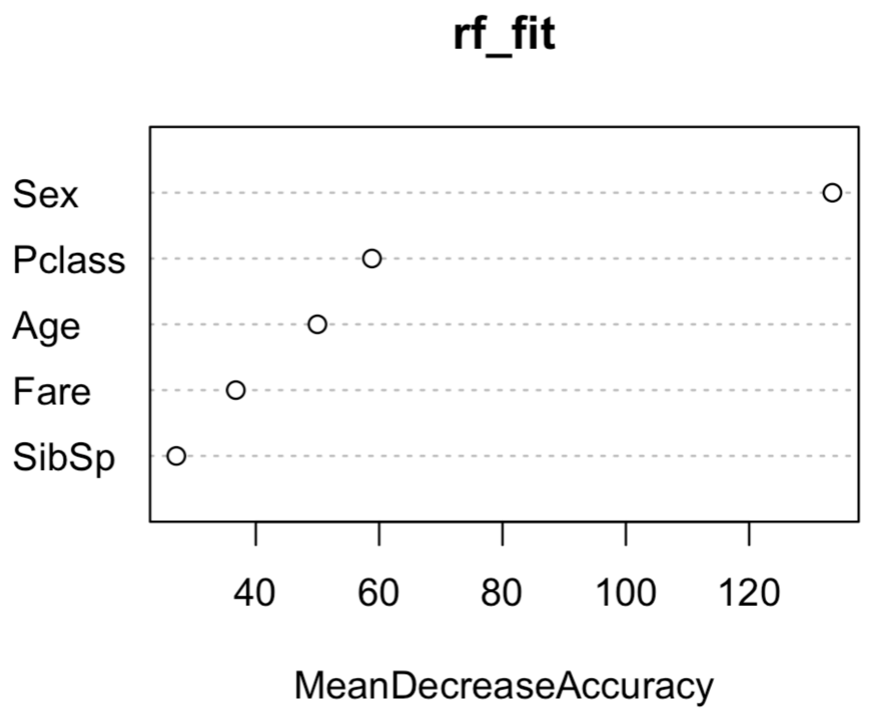
```
> print(rf_fit)

Call:
 randomForest(formula = 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
```

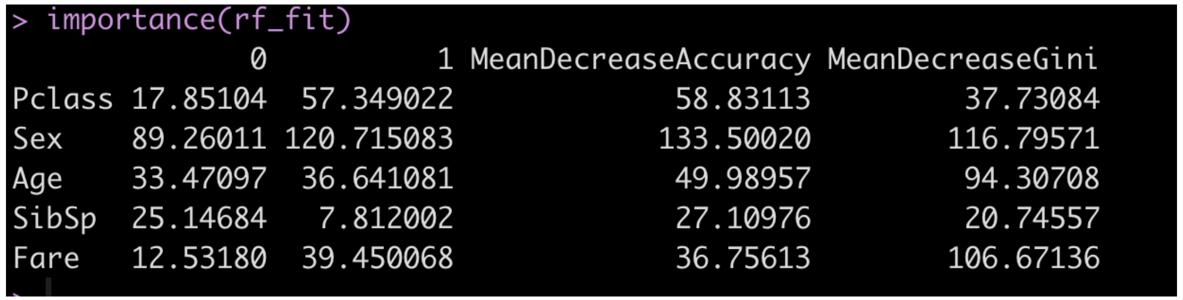
plot() function against rf object – 600 trees, mtry = 3



- 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

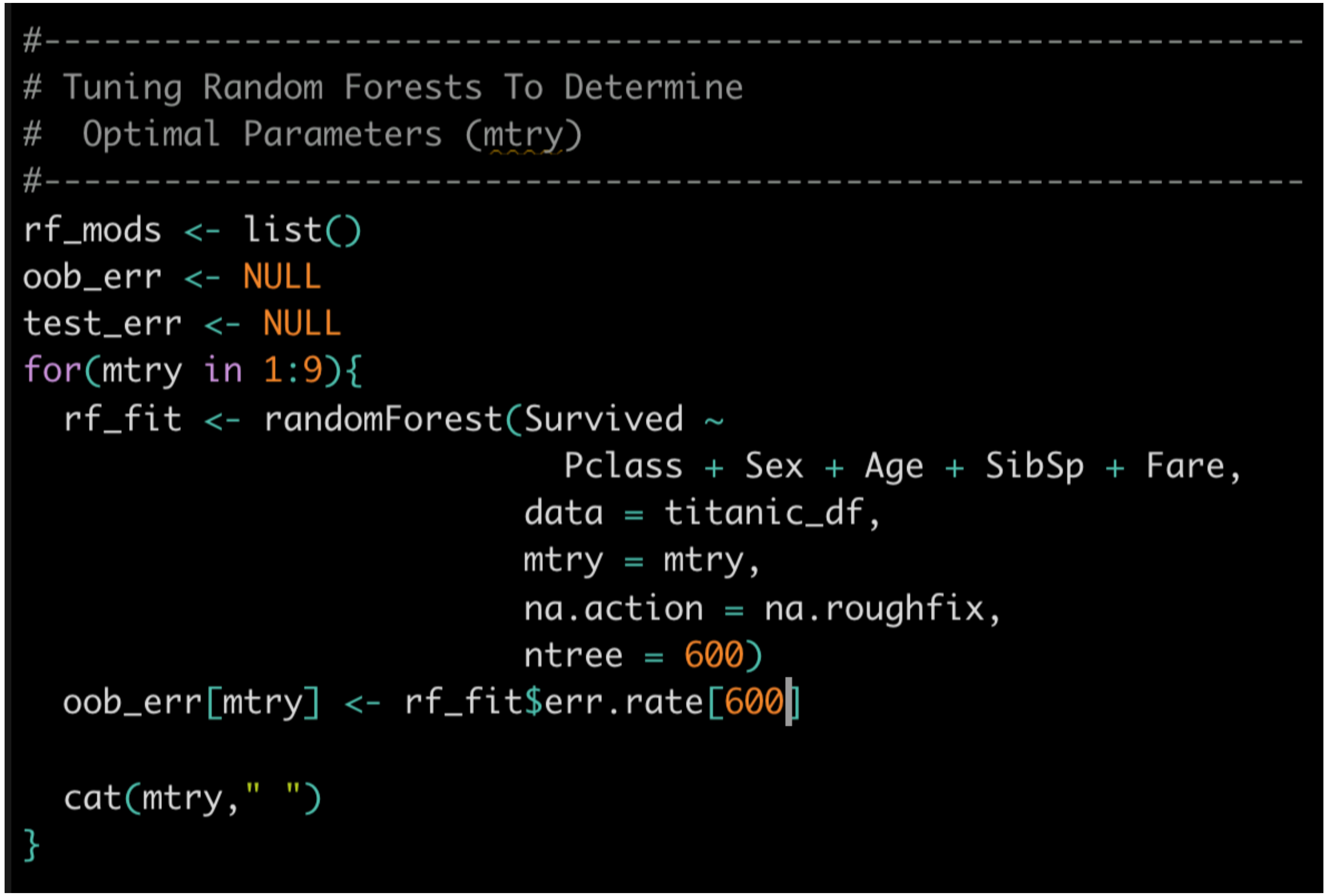
Which Variables Matter Most? Variable Importance




```
> importance(rf_fit)
              0          1 MeanDecreaseAccuracy MeanDecreaseGini
Pclass 17.85104  57.349022             58.83113         37.73084
Sex    89.26011 120.715083            133.50020        116.79571
Age    33.47097  36.641081             49.98957         94.30708
SibSp  25.14684   7.812002             27.10976         20.74557
Fare   12.53180  39.450068             36.75613        106.67136
```

- 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
- Sex is most important, followed by class, then age, fare and sibsp

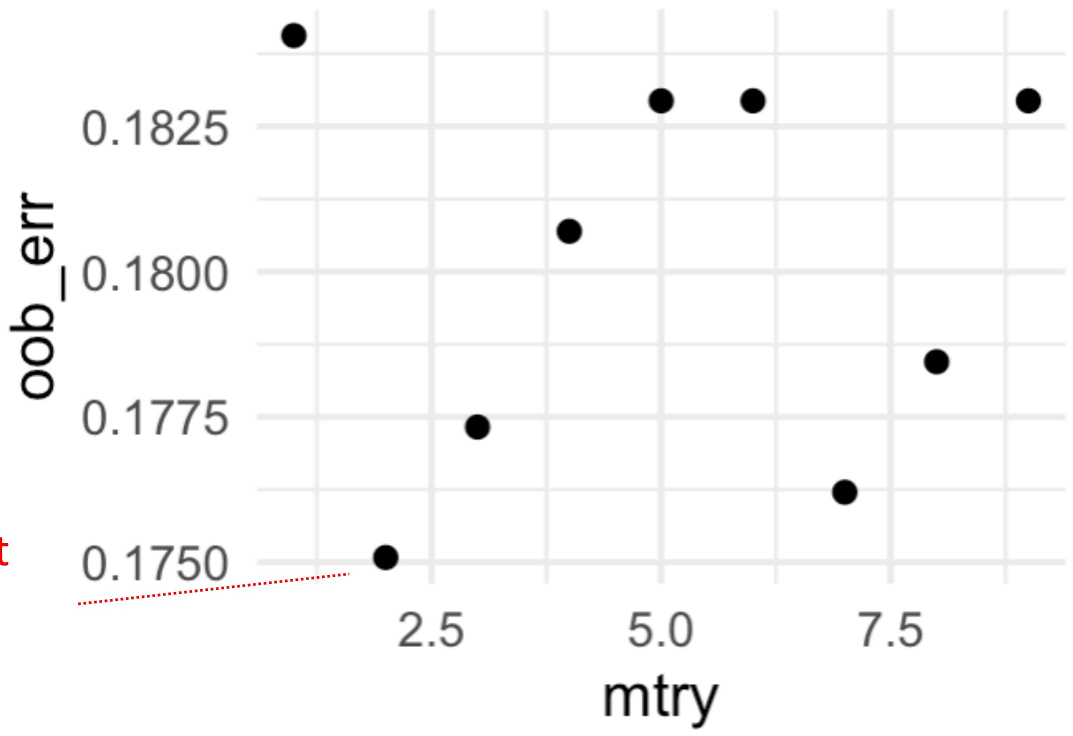
Cross-validate to Tune/Select Optimal "mtry"



```
#------------------------------------------------------------------
# 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," ")
}
```

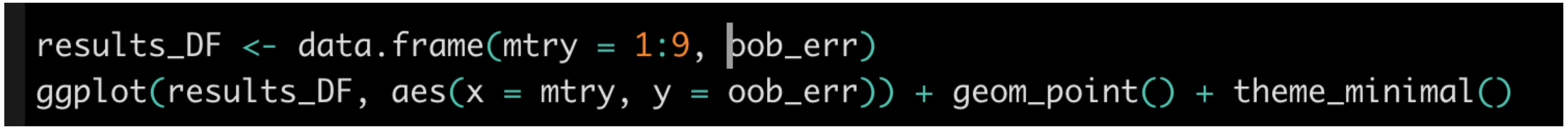
- In the previous, we just guessed that mtry = 3 is the right parameter.
- To get a better model fit, we can estimate a model for every value of mtry and choose the parameter that maximizes cross-validated (out of bag) fit
- Most packages will cross-validate mtry for us, but in principle it is the same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

Cross-validate to select “mtry”



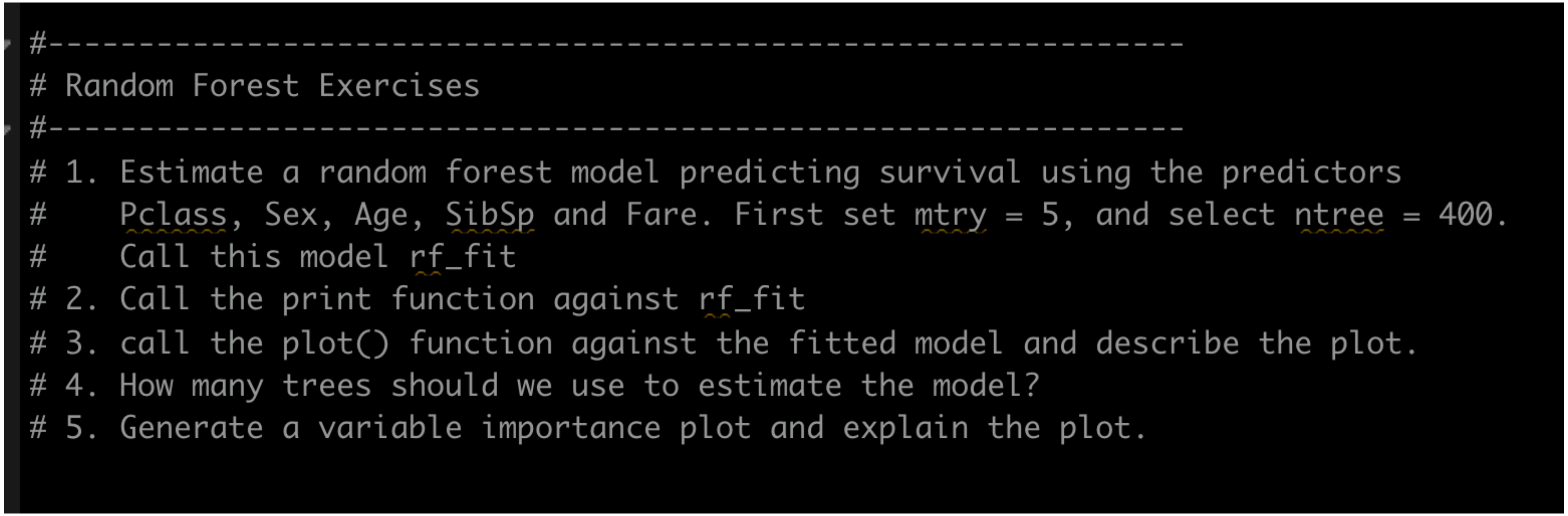
Mtry value
With lowest
error

- 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
- Note, many modern ML systems (caret, scikit-learn) do this parameter optimization automatically



```
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?
# 5. Generate a variable importance plot and explain the plot.
```

Class 18 Summary

- 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!