Random forests and wine

Random forests

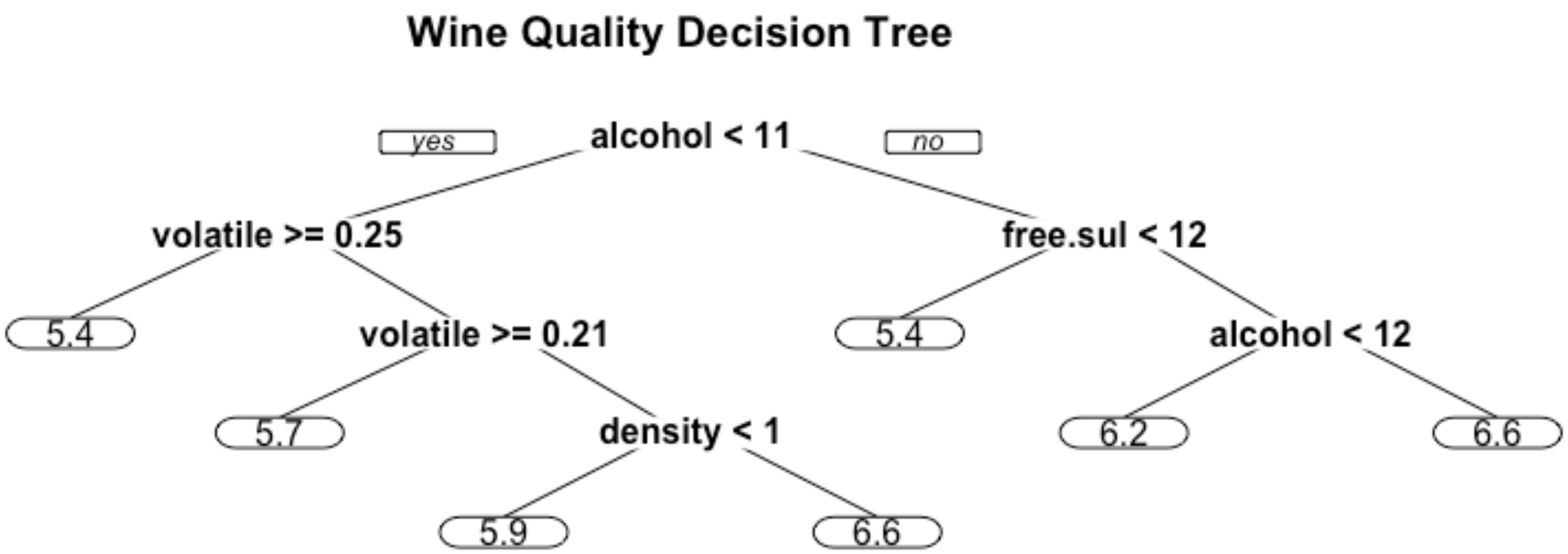
- Popular type of machine learning model
- Good for beginners
- Robust to overfitting
- Yield very accurate, non-linear models

Random forests

- Unlike linear models, they have *hyperparameters*
- Hyperparameters require manual specification
- Can impact model fit and vary from dataset-to-dataset
- Default values often OK, but occasionally need adjustment

Random forests

- Start with a simple decision tree
- Decision trees are fast, but not very accurate

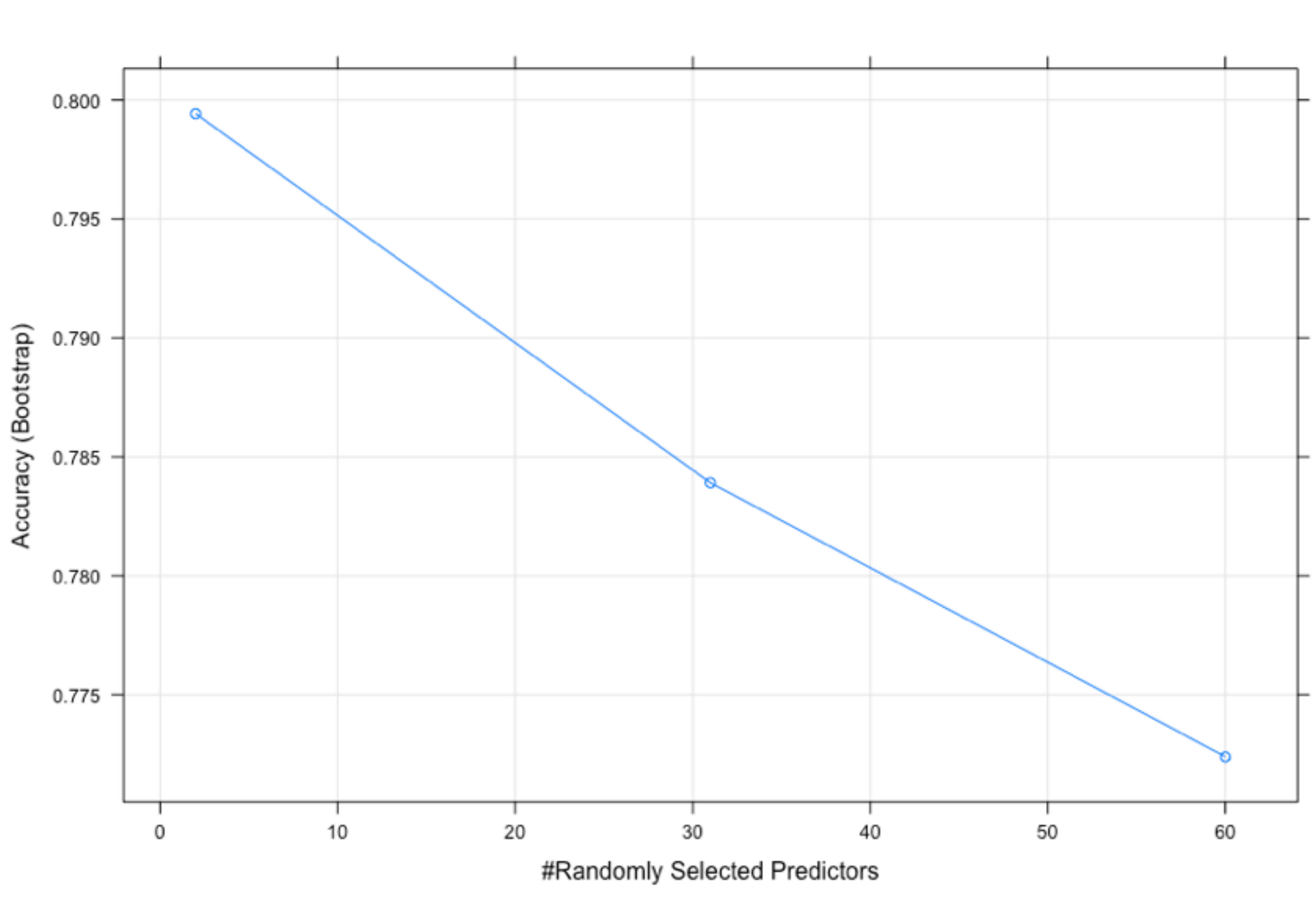


Random forests

- Improve accuracy by fitting many trees
- Fit each one to a bootstrap sample of your data
- Called *bootstrap aggregation* or *bagging*
- Randomly sample columns at each split

Random forests

```
# Load some data
> library(caret)
> library(mlbench)
> data(Sonar)

# Set seed
> set.seed(42)

# Fit a model
> model <- train(Class~.,
                 data = Sonar,
                 method = "ranger"
  )

# Plot the results
> plot(model)
```



Let's practice!

Explore a wider model space

Random forests require tuning

- *Hyperparameters* control how the model is fit
- Selected "by hand" before the model is fit
- Most important is `mtry`
  - Number of randomly selected variables used at each split
  - Lower value = more random
  - Higher value = less random
- Hard to know the best value in advance

`caret` to the rescue!

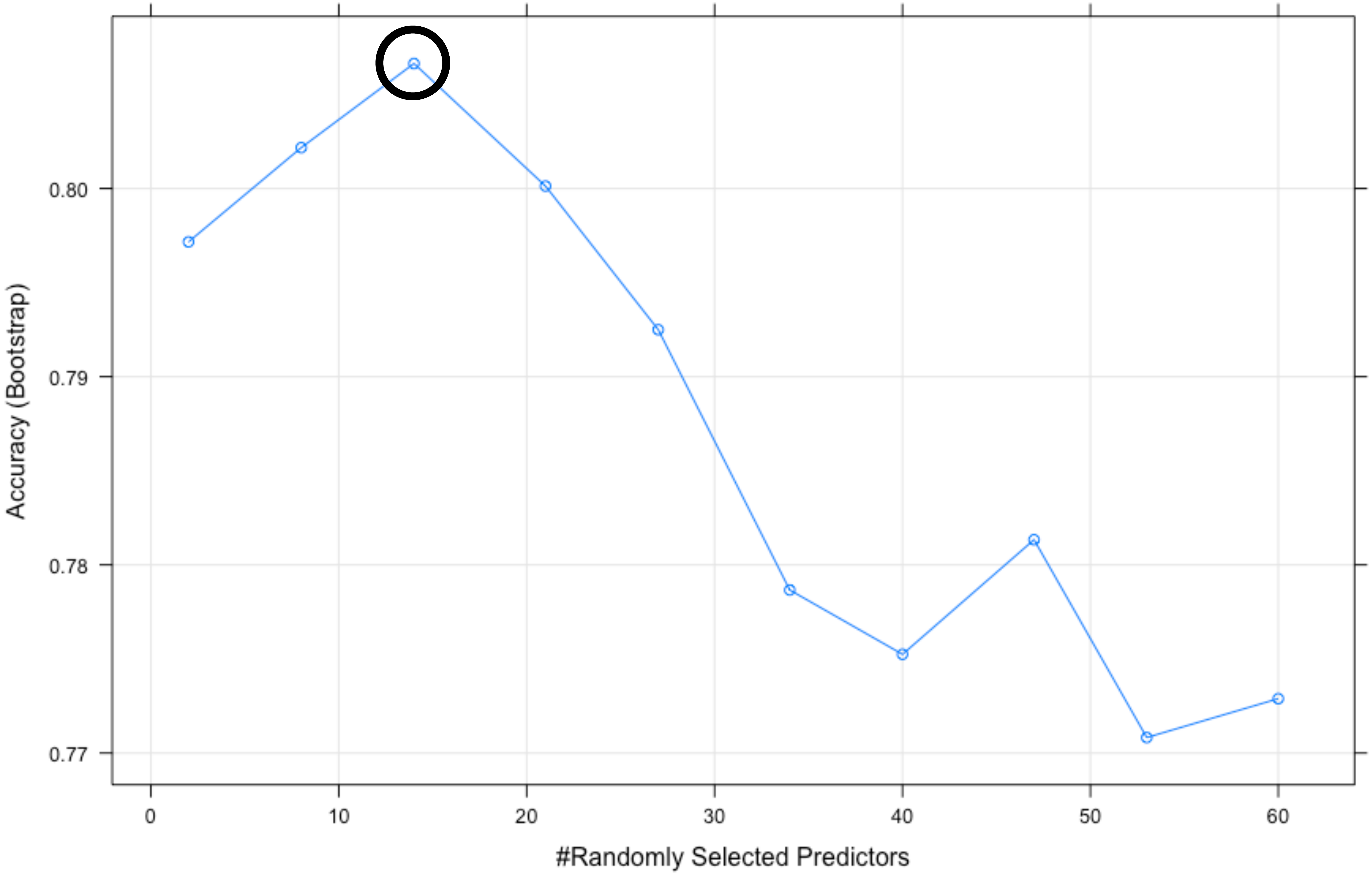
- Not only does `caret` do cross-validation...
- It also does *grid search*
- Select hyperparameters based on out-of-sample error

Example: sonar data

- `tuneLength` argument to `caret::train()`
- Tells `caret` how many different variations to try

```
# Load some data
> library(caret)
> library(mlbench)
> data(Sonar)

# Fit a model with a deeper tuning grid
> model <- train(Class~., data = Sonar,
                 method = "ranger", tuneLength = 10)

# Plot the results
> plot(model)
```

Plot the results



Let's practice!

Custom tuning grids

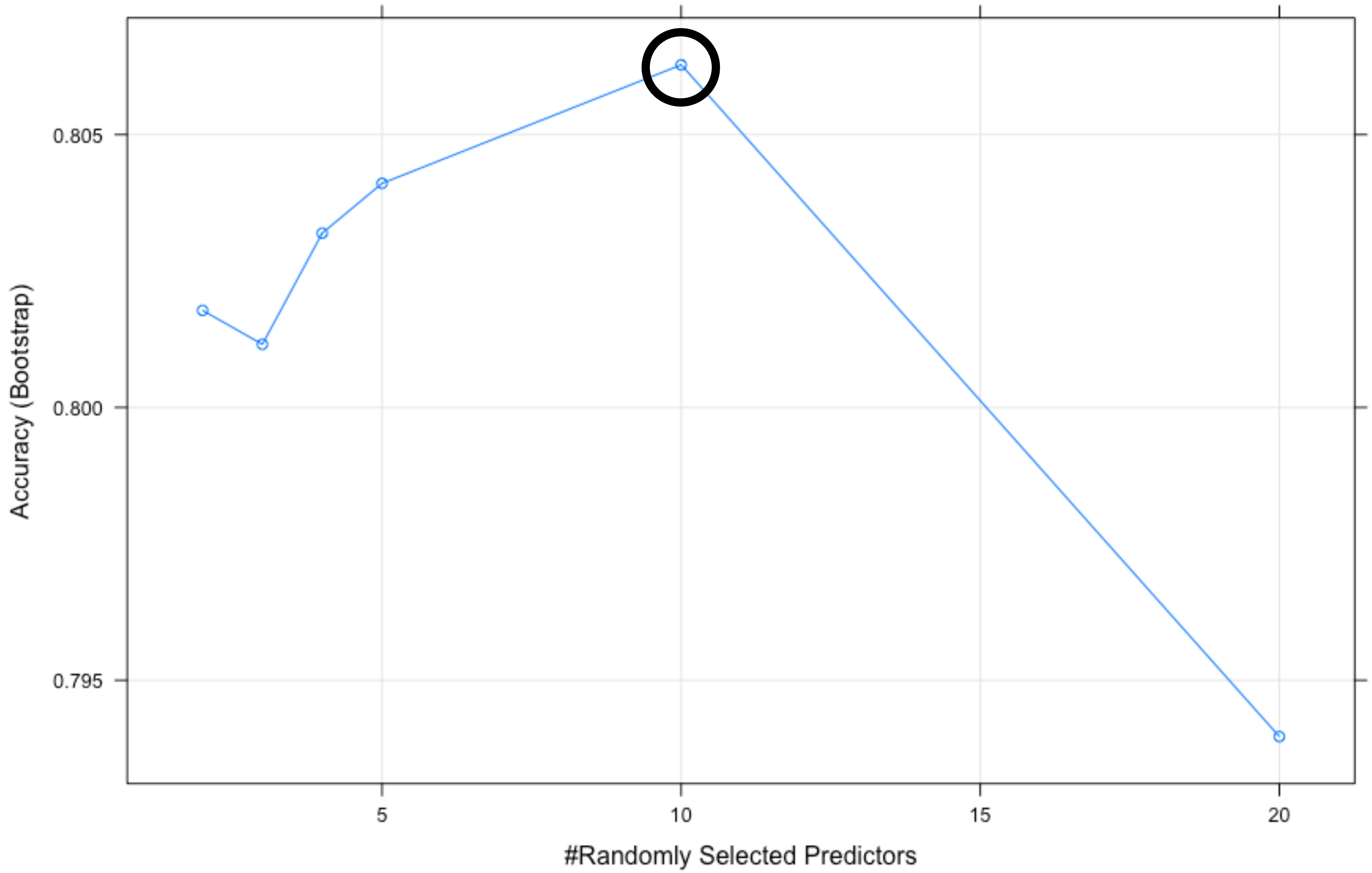
Pros and cons of custom tuning

- Pass custom tuning grids to `tuneGrid` argument
- Advantages
  - Most flexible method for fitting `caret` models
  - Complete control over how the model is fit
- Disadvantages
  - Requires some knowledge of the model
  - Can dramatically increase run time

Custom tuning example

```
# Define a custom tuning grid
> myGrid <- data.frame(mtry = c(2, 3, 4, 5, 10, 20))

# Fit a model with a custom tuning grid
> set.seed(42)
> model <- train(Class ~ ., data = Sonar, method = "ranger",
                 tuneGrid = myGrid)

# Plot the results
> plot(model)
```

Custom tuning



Let's practice!

Introducing `glmnet`

Introducing `glmnet`

- Extension of `glm` models with built-in variable selection
- Helps deal with *collinearity* and small samples sizes
- Two primary forms
  - Lasso regression **Penalizes number of non-zero coefficients**
  - Ridge regression **Penalizes absolute magnitude of coefficients**
- Attempts to find a parsimonious (i.e. simple) model
- Pairs well with random forest models

Tuning `glmnet` models

- Combination of lasso and ridge regression
- Can fit a mix of the two models
- `alpha` [0, 1]: pure lasso to pure ridge
- `lambda` (0, infinity): size of the penalty

Example: "don't overfit"

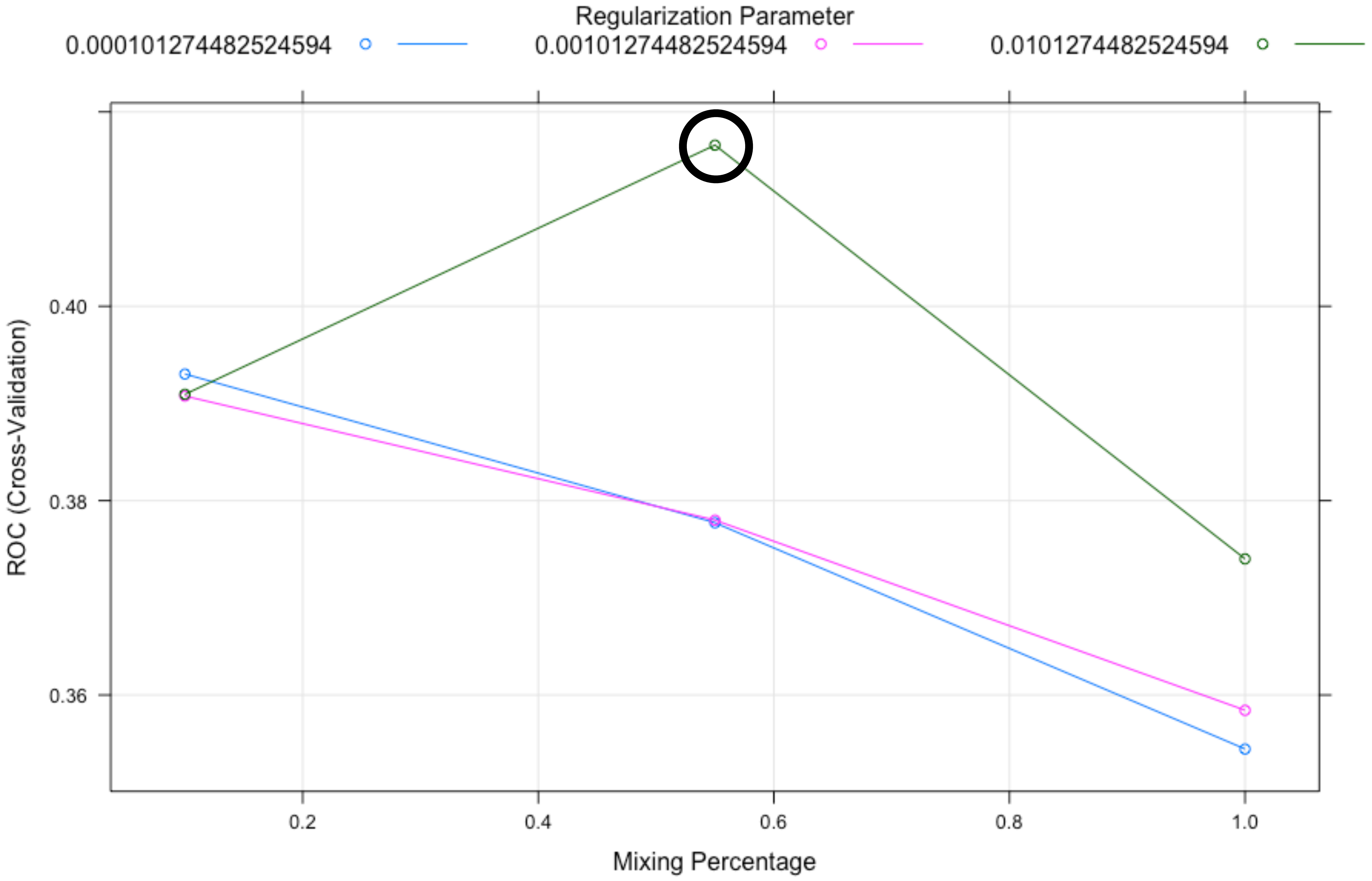
```
# Load data
> overfit <- read.csv("http://s3.amazonaws.com/assets.datacamp.com/
production/course_1048/datasets/overfit.csv")

# Make a custom trainControl
> myControl <- trainControl(
    method = "cv", number = 10,
    summaryFunction = twoClassSummary,
    classProbs = TRUE, # Super important!
    verboseIter = TRUE
  )
```

Try the defaults

```
# Fit a model
> set.seed(42)
> model <- train(y ~ ., overfit, method = "glmnet",
                 trControl = myControl)

# Plot results
> plot(model)
```

- 3 values of `alpha`
- 3 values of `lambda`

Plot the results



Let's practice!

`glmnet` with custom tuning grid

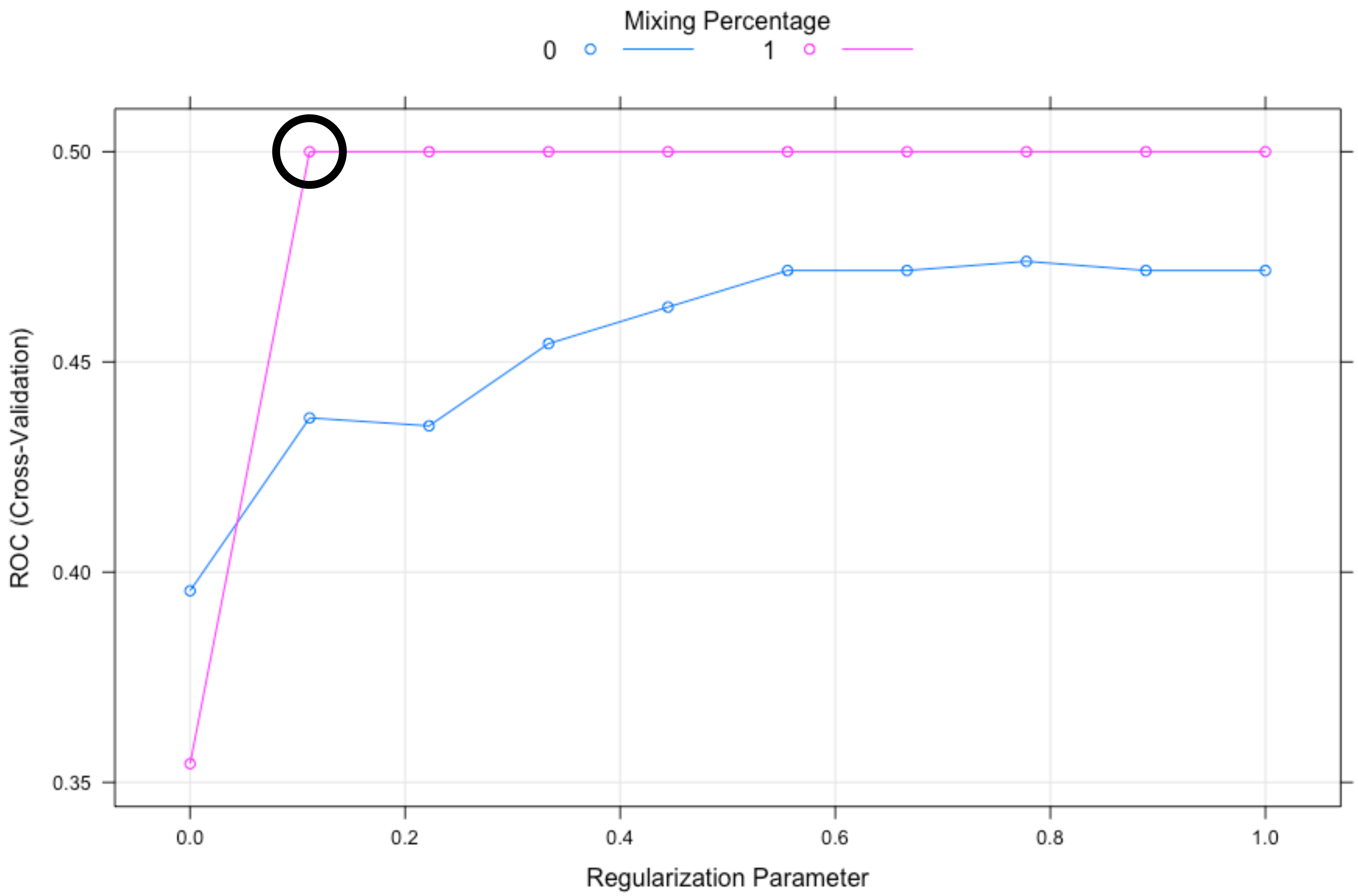
Custom tuning `glmnet` models

- 2 tuning parameters: `alpha` and `lambda`
- For single `alpha`, all values of `lambda` fit simultaneously
- Many models for the "price" of one

Example: `glmnet` tuning

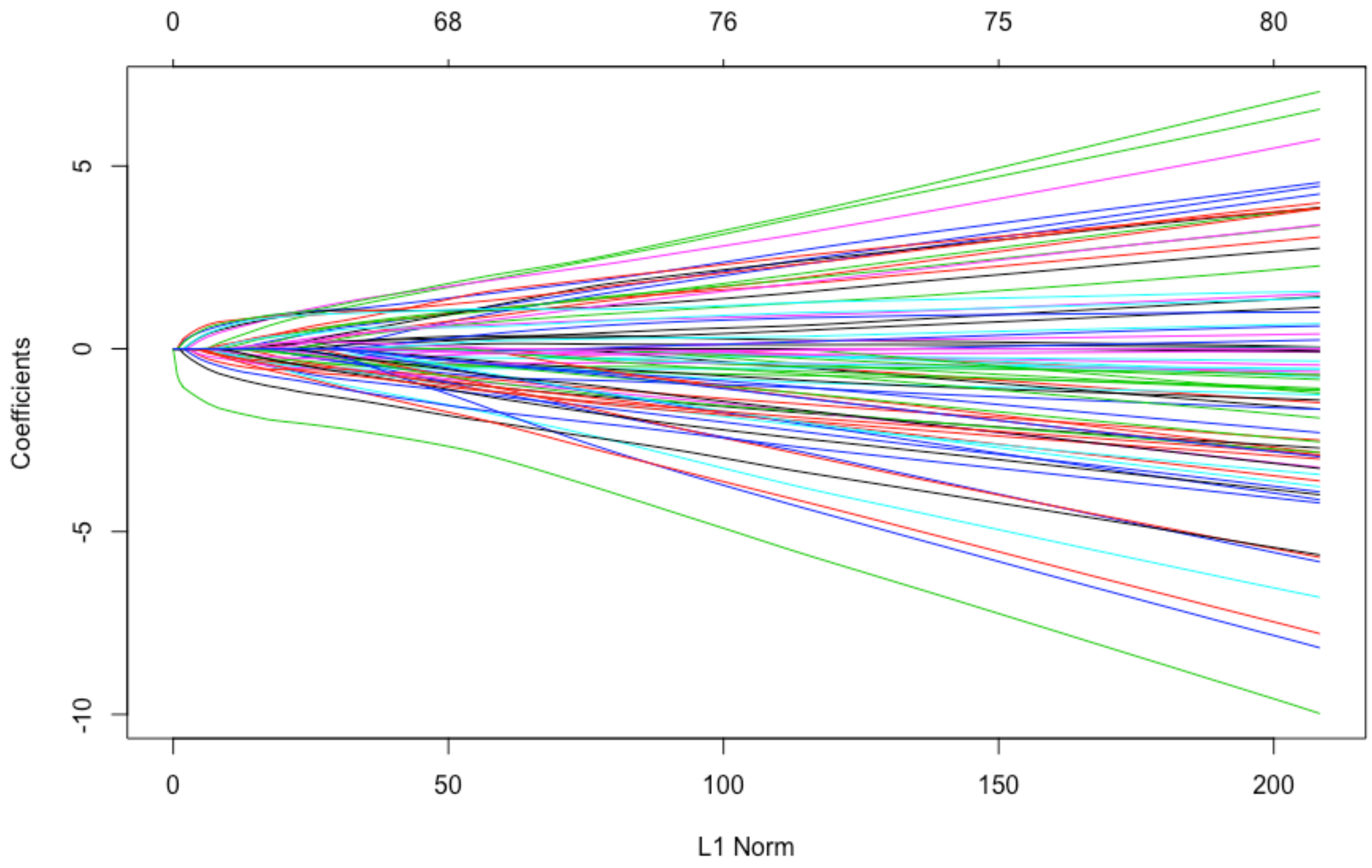
```
# Make a custom tuning grid
> myGrid <- expand.grid(
    alpha = 0:1,
    lambda = seq(0.0001, 0.1, length = 10)
  )

# Fit a model
> set.seed(42)
> model <- train(y ~ ., overfit, method = "glmnet",
                 tuneGrid = myGrid, trControl = myControl)

# Plot results
> plot(model)
```

Compare models visually



Full regularization path

```
> plot(model$finalModel)
```



Let's practice!