MACHINE LEARNING TOOLBOX

Welcome to the Machine Learning Toolbox!

Supervised learning

- `caret` R package
- Automates *supervised learning* (a.k.a. *predictive modeling*)
- Target variable

Supervised learning

- Two types of predictive models
  - Classification → **Qualitative**
  - Regression → **Quantitative**
- Use *metrics* to evaluate models
  - Quantifiable
  - Objective
- *Root Mean Squared Error* (RMSE) for regression (e.g. `lm()`)

Evaluating model performance

- Common to calculate in-sample RMSE
  - Too optimistic
  - Leads to overfitting
- Better to calculate out-of-sample error (a la `caret`)
  - Simulates real-world usage
  - Helps avoid overfitting

In-sample error

```
> # Fit a model to the mtcars data
> data(mtcars)
> model <- lm(mpg ~ hp, mtcars[1:20, ])

> # Predict in-sample
> predicted <- predict(model, mtcars[1:20, ], type = "response")

> # Calculate RMSE
> actual <- mtcars[1:20, "mpg"]
> sqrt(mean((predicted - actual)^2))
[1] 3.172132
```

The Machine Learning Toolbox

Let’s practice!

MACHINE LEARNING TOOLBOX

Out-of-sample error measures

Out-of-sample error

- Want models that don't overfit and generalize well
- Do the models perform well on new data?
- Test models on new data, or a *test set*
  - Key insight of machine learning
  - In-sample validation almost guarantees overfitting
- Primary goal of `caret` and this course: don’t overfit

Example: out-of-sample RMSE

```
> # Fit a model to the mtcars data
> data(mtcars)
> model <- lm(mpg ~ hp, mtcars[1:20, ])

> # Predict out-of-sample
> predicted <- predict(model, mtcars[21:32, ], type = "response")

> # Evaluate error
> actual <- mtcars[21:32, "mpg"]
> sqrt(mean((predicted - actual)^2))
[1] 5.507236
```

Alternatives:

`createResamples()`
`createFolds()`

Compare to in-sample RMSE

```
> # Fit a model to the full dataset
> model2 <- lm(mpg ~ hp, mtcars)

> # Predict in-sample
> predicted2 <- predict(model, mtcars, type = "response")

> # Evaluate error
> actual2 <- mtcars[, "mpg"]
> sqrt(mean((predicted2 - actual2)^2))
[1] 3.74
```

Compare to out-of-sample RMSE of 5.5

MACHINE LEARNING TOOLBOX

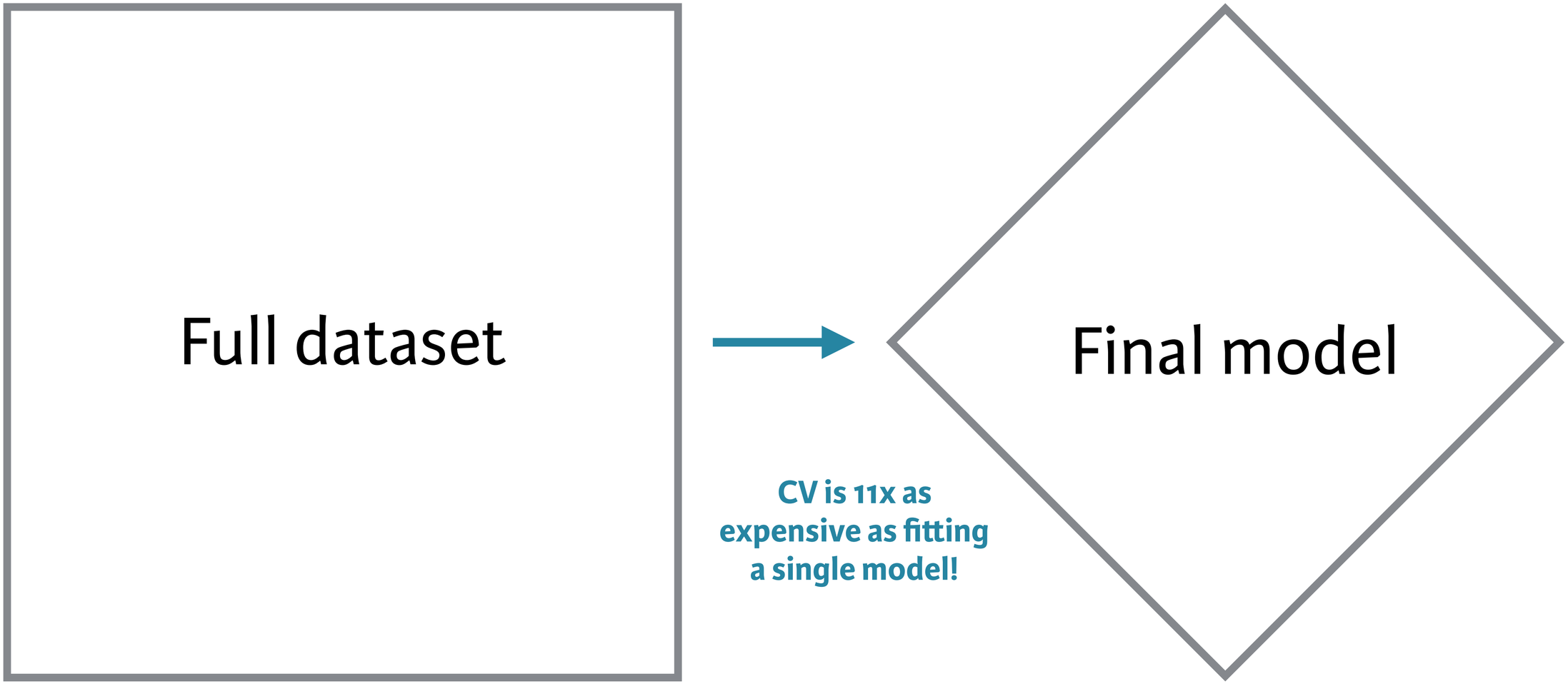
Let’s practice!

MACHINE LEARNING TOOLBOX

Cross-validation

Cross-validation

Fit final model on full dataset



Cross-validation

```
> # Set seed for reproducibility
> library(caret)
> data(mtcars)
> set.seed(42)

> # Fit linear regression model
> model <- train(mpg ~ hp, mtcars,
                 method = "lm",
                 trControl = trainControl(
                   method = "cv", number = 10,
                   verboseIter = TRUE
                 )
  )
+ Fold01: parameter=none
+ Fold02: parameter=none
          ...
- Fold10: parameter=none
Aggregating results
Fitting final model on full training set
```

MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

Logistic regression on Sonar

Classification models

- Categorical (i.e. qualitative) target variable
- Example: will a loan default?
- Still a form of supervised learning
- Use a train/test split to evaluate performance
- Use the `Sonar` dataset
- Goal: distinguish rocks from mines

Example: Sonar data

```
> # Load the Sonar dataset
> library(mlbench)
> data(Sonar)

> # Look at the data
> Sonar[1:6, c(1:5, 61)]
      V1     V2     V3     V4     V5 Class
1 0.0200 0.0371 0.0428 0.0207 0.0954     R
2 0.0453 0.0523 0.0843 0.0689 0.1183     R
3 0.0262 0.0582 0.1099 0.1083 0.0974     R
4 0.0100 0.0171 0.0623 0.0205 0.0205     R
5 0.0762 0.0666 0.0481 0.0394 0.0590     R
6 0.0286 0.0453 0.0277 0.0174 0.0384     R
```

Splitting the data

- Randomly split data into training and test sets
- Use a 60/40 split, instead of 80/20
- `Sonar` dataset is small, so 60/40 gives a larger, more reliable test set

Splitting the data

```
# Randomly order the dataset
> rows <- sample(nrow(Sonar))
> Sonar <- Sonar[rows, ]

# Find row to split on
> split <- round(nrow(Sonar) * .60)
> train <- Sonar[1:split, ]
> test <- Sonar[(split + 1):nrow(Sonar), ]

# Confirm test set size
> nrow(train) / nrow(Sonar)
[1] 0.6009615
```

MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

Confusion matrix

Confusion matrix

Prediction \ Reference	Yes	No
Yes	True positive	False positive
No	False negative	True negative

Confusion matrix

```
# Fit a model
> model <- glm(Class ~ ., family = binomial(link = "logit"),
train)
> p <- predict(model, test, type = "response")
> summary(p)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  0.9885  0.5296  1.0000  1.0000

# Turn probabilities into classes and look at their frequencies
> p_class <- ifelse(p > .50, "M", "R")
> table(p_class)
p_class
 M  R
44 39
```

Confusion matrix

- Make a 2-way frequency table
- Compare predicted vs. actual classes

```
# Make simple 2-way frequency table
> table(p_class, test[["Class"]])
p_class  M  R
      M 13 31
      R 30  9
```

Confusion matrix

```
# Use caret's helper function to calculate additional statistics
> confusionMatrix(p_class, test[["Class"]])
          Reference
Prediction  M  R
         M 13 31
         R 30  9

               Accuracy : 0.2651
                 95% CI : (0.1742, 0.3734)
    No Information Rate : 0.5181
    P-Value [Acc > NIR] : 1

                  Kappa : -0.4731
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.3023
            Specificity : 0.2250
         Pos Pred Value : 0.2955
         Neg Pred Value : 0.2308
```

MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

Class probabilities and class predictions

Different thresholds

- Not limited to 50% threshold
  - 10% would catch more mines with less certainty
  - 90% would catch fewer mines with more certainty
- Balance true positive and false positive rates
- Cost-benefit analysis

Confusion matrix

```
# Use a larger cutoff
> p_class <- ifelse(p > .99, "M", "R")
> table(p_class)
p_class
 M  R
41 42

# Make simple 2-way frequency table
> table(p_class, test[["Class"]])
p_class  M  R
      M 13 28
      R 30 12
```

Confusion matrix with `caret`

```
# Use caret to produce confusion matrix
> confusionMatrix(p_class, test[["Class"]])
          Reference
Prediction  M  R
         M 13 28
         R 30 12

               Accuracy : 0.3012
                 95% CI : (0.2053, 0.4118)
    No Information Rate : 0.5181
    P-Value [Acc > NIR] : 1.0000

                  Kappa : -0.397
 Mcnemar's Test P-Value : 0.8955

            Sensitivity : 0.3023
            Specificity : 0.3000
         Pos Pred Value : 0.3171
         Neg Pred Value : 0.2857
```

MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

Introducing the ROC curve

The challenge

- Many possible classification thresholds
- Requires manual work to choose
- Easy to overlook a particular threshold
- Need a more systematic approach

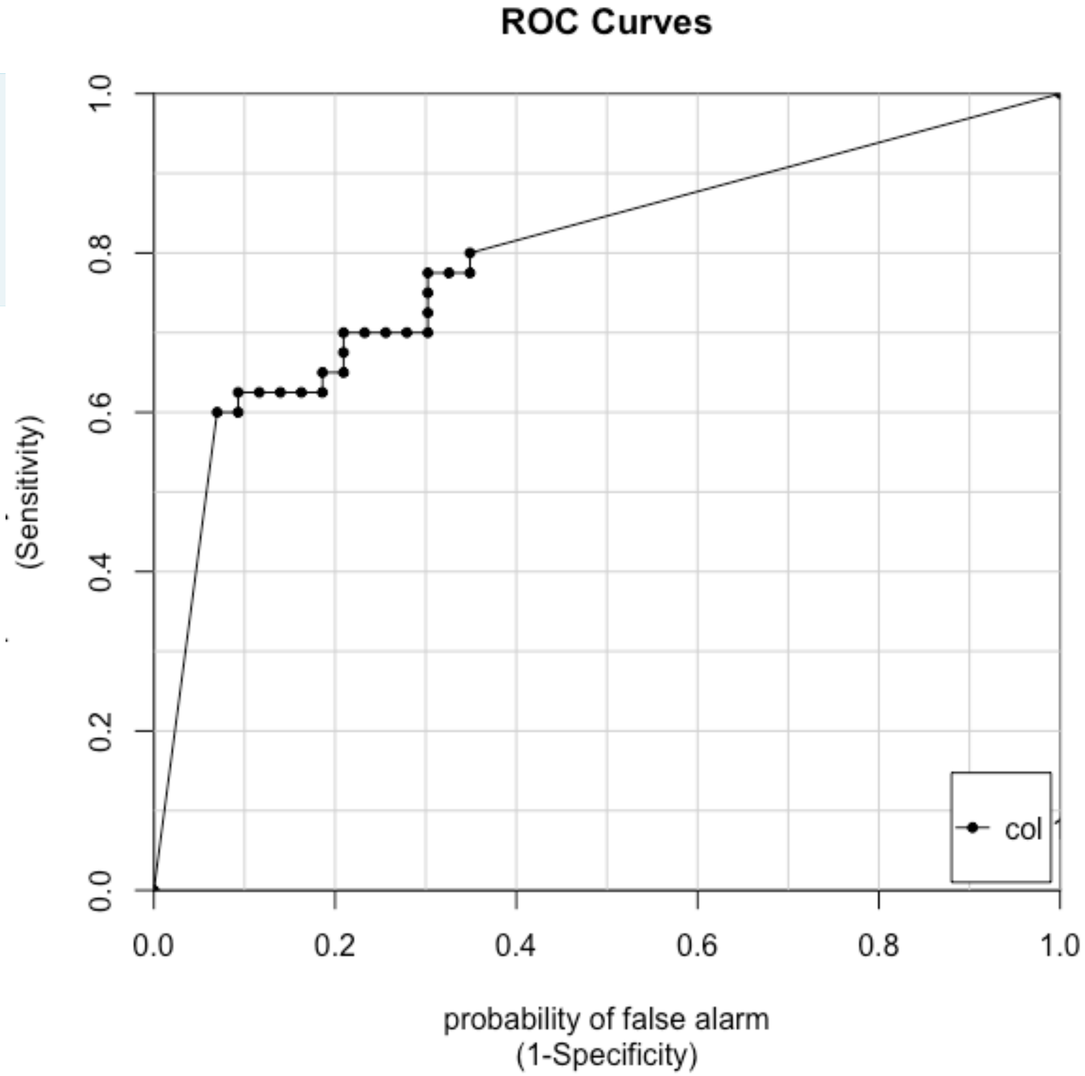
ROC curves

- Plot true/false positive rate at every possible threshold
- Visualize tradeoffs between two extremes **100% true positive rate vs. 0% false positive rate**
- Result is an ROC curve
- Developed as a method for analyzing radar signals

An example ROC curve

```
# Create ROC curve
> library(caTools)
> colAUC(p, test[["Class"]], plotROC = TRUE)
```

- X-axis: false positive rate
- Y-axis: true positive rate
- Each point along the curve represents a different threshold



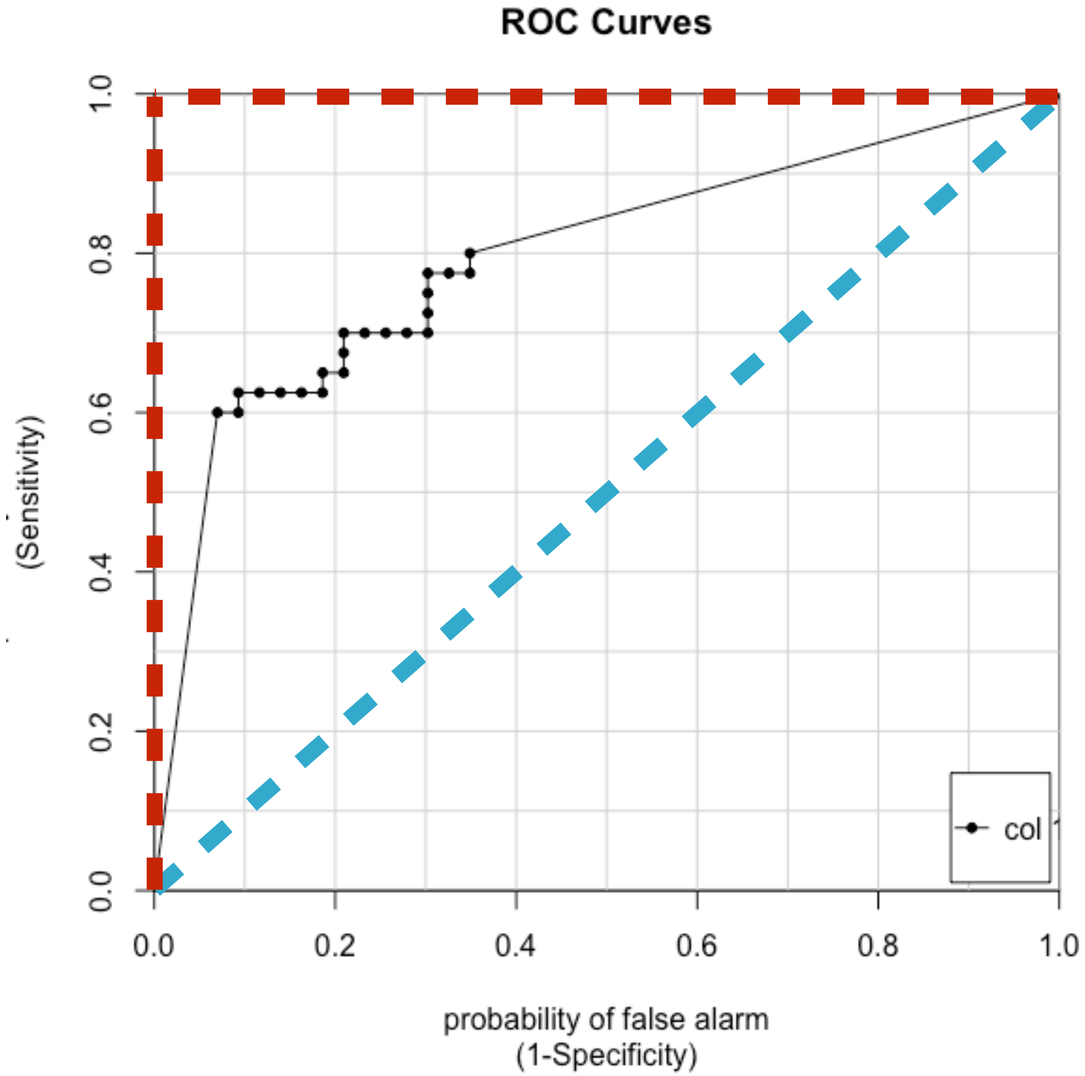
MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

Area under the curve (AUC)

From ROC to AUC



Defining AUC

- Single-number summary of model accuracy
- Summarizes performance across all thresholds
- Rank different models within the same dataset

Defining AUC

- Ranges from 0 to 1
  - 0.5 = random guessing
  - 1 = model always right
  - 0 = model always wrong
- Rule of thumb: AUC as a letter grade
  - 0.9 = "A"
  - 0.8 = "B"
  - ...

MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

Random forests and wine

Random forests

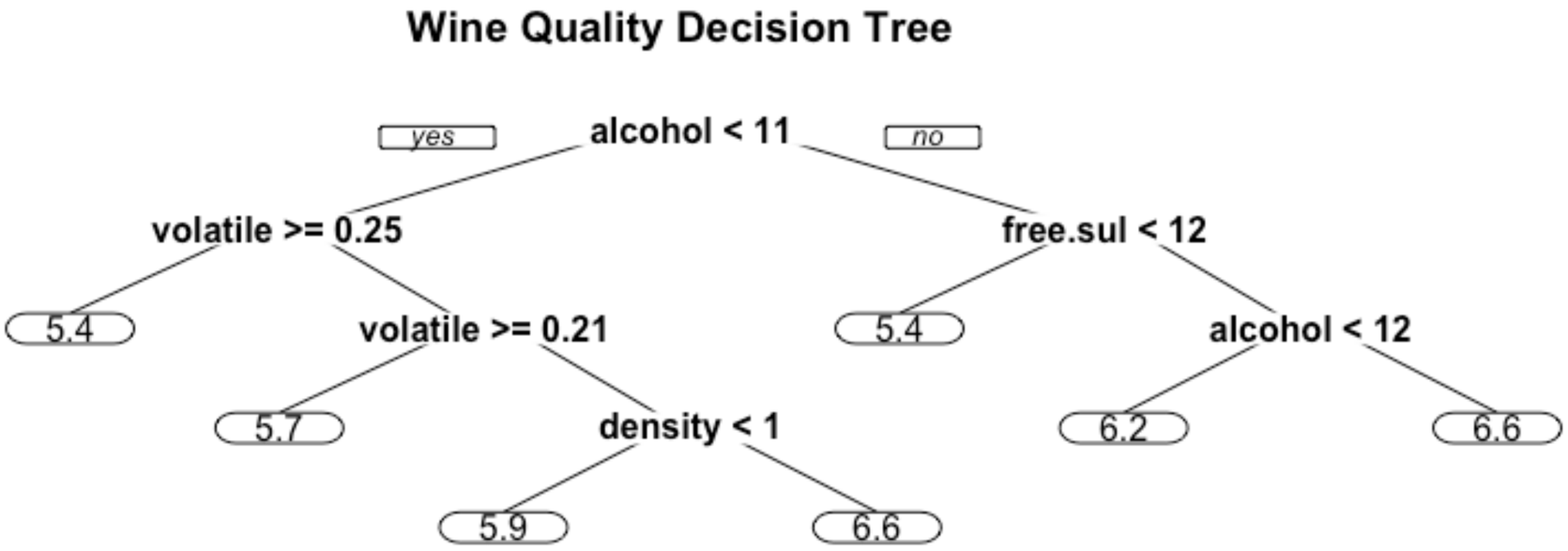
- Popular type of machine learning model
- Good for beginners
- Robust to overfitting
- Yield very accurate, non-linear models

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

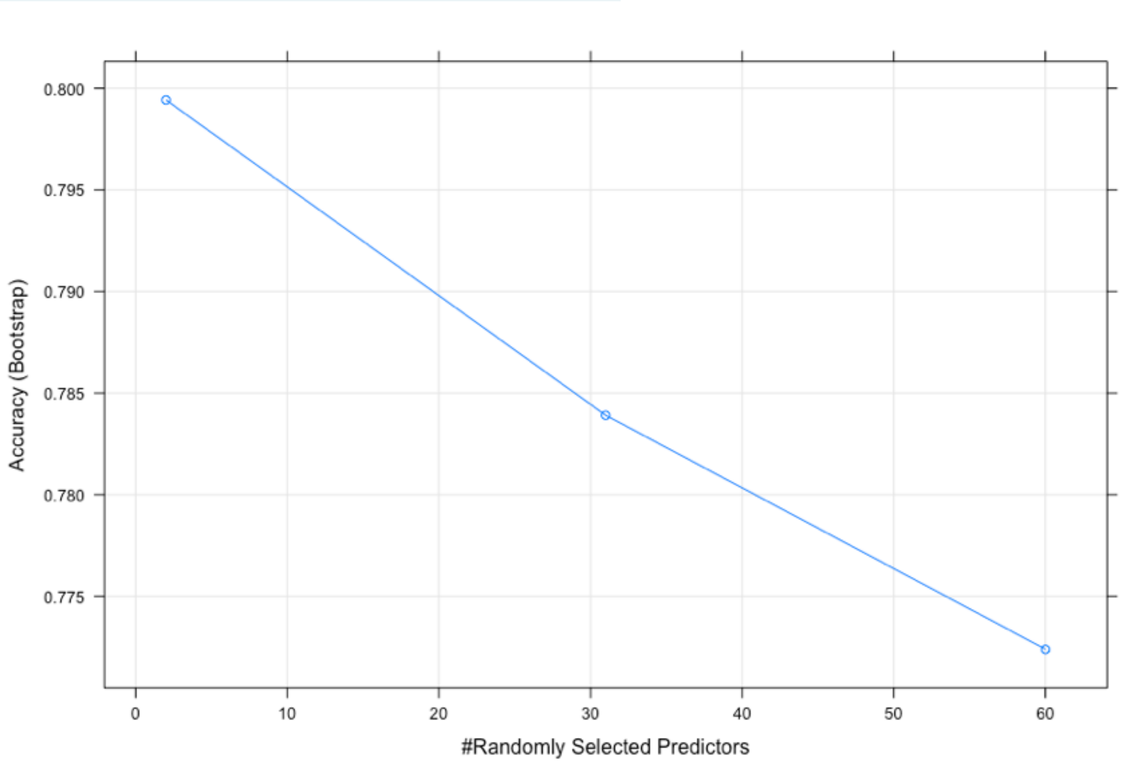


Random forests

- 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)
```



MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

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!

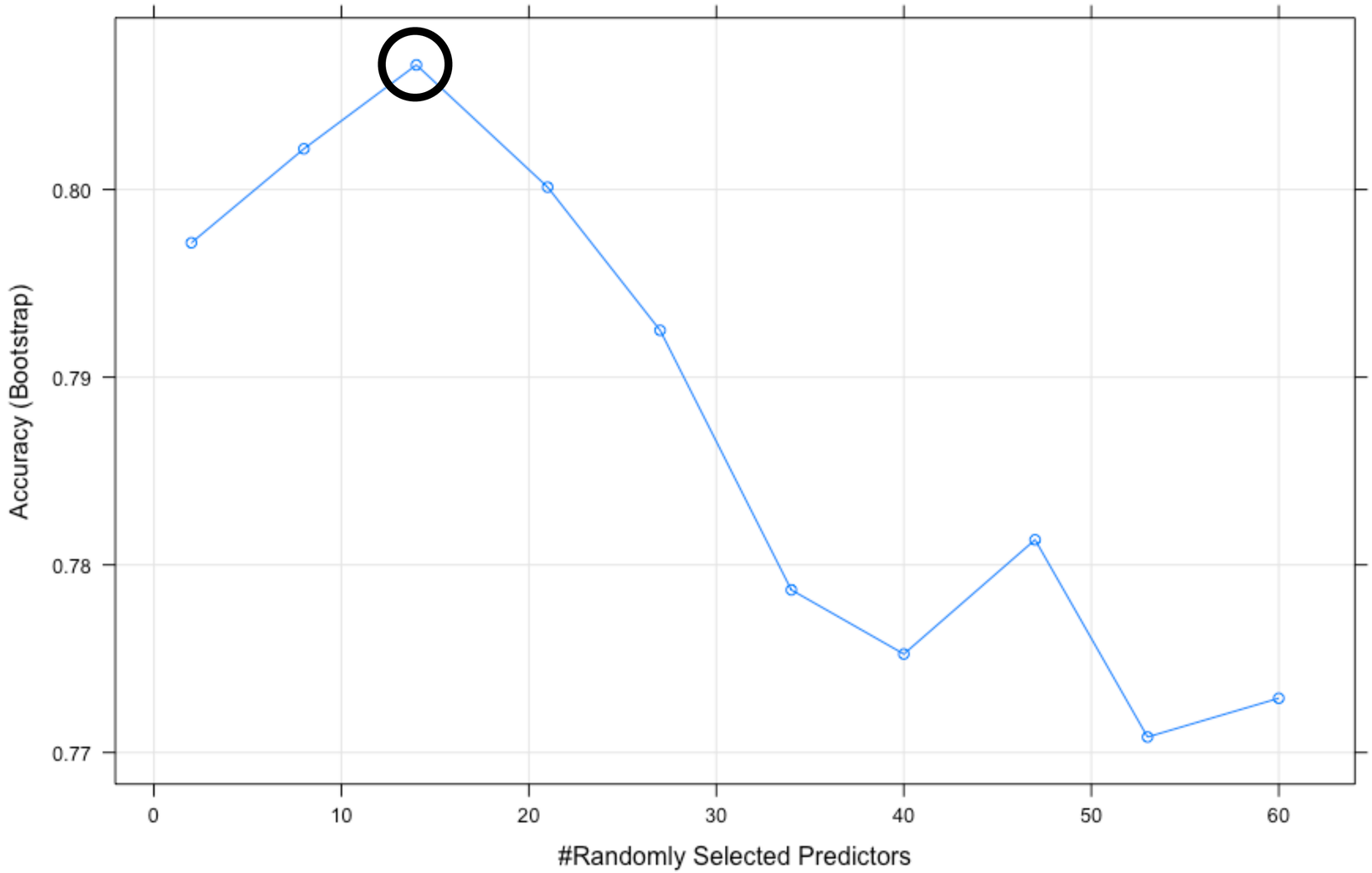
- Not only does `caret` do cross-validation...
- It also does *grid search*
- Select hyperparameters based on out-of-sample error

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



MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

Custom tuning grids

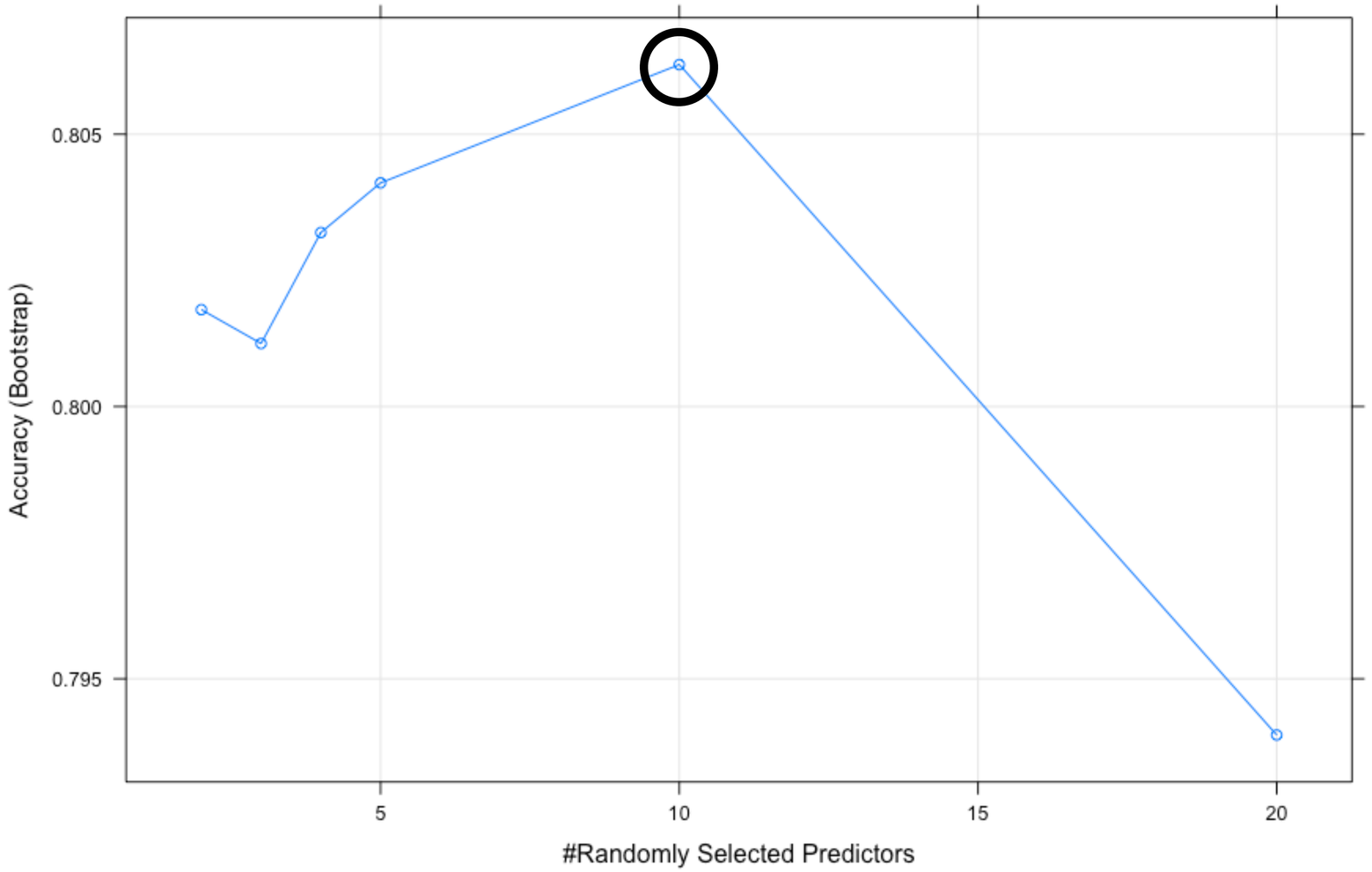
Pros and cons of custom tuning

- 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



MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

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"

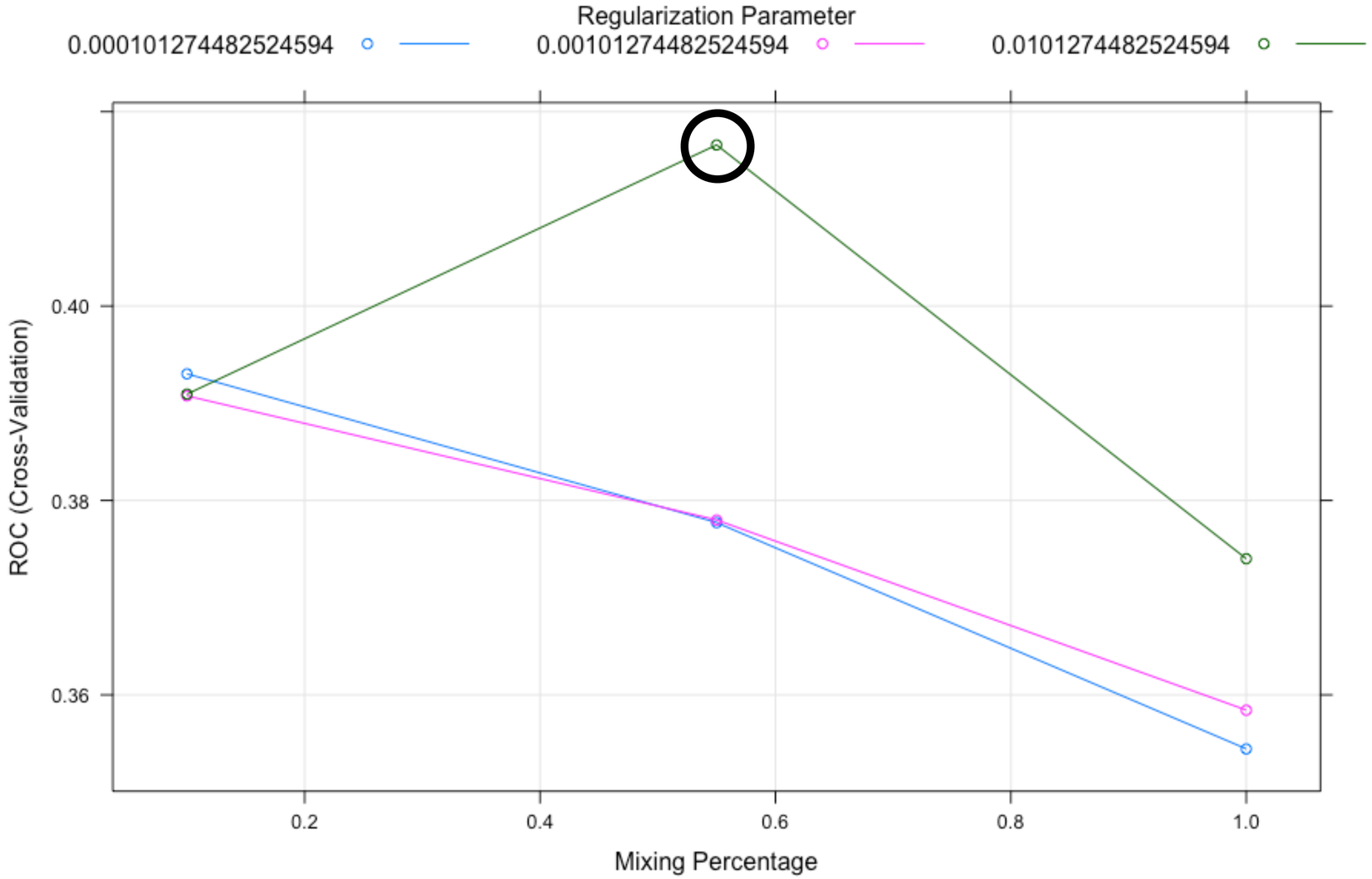
```
# Load data
> overfit <- read.csv("http://s3.amazonaws.com/assets.datacamp.com/
production/course_1048/datasets/overfit.csv")

# 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



MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

`glmnet` with custom tuning grid

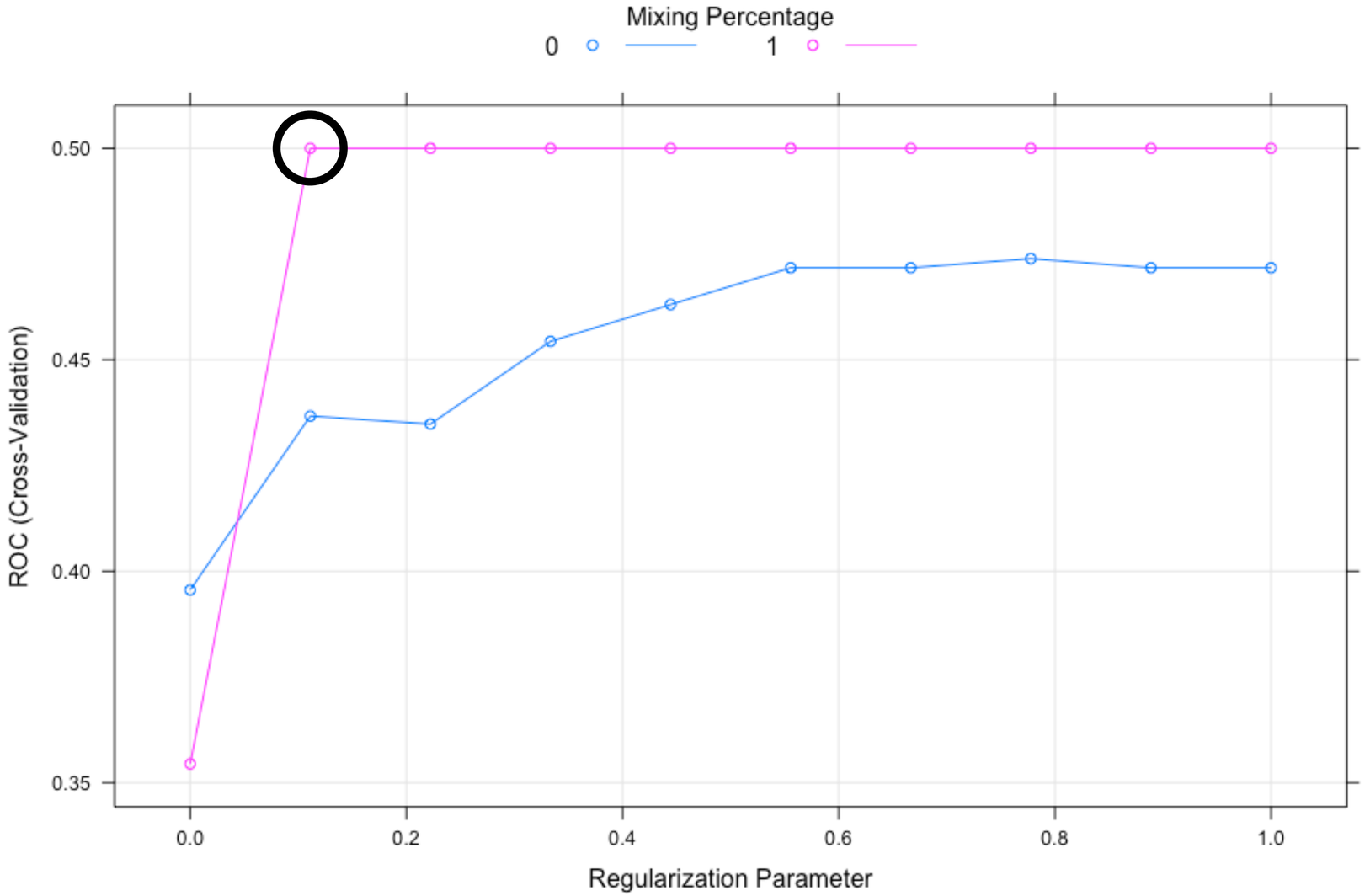
Custom tuning `glmnet` models

- 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

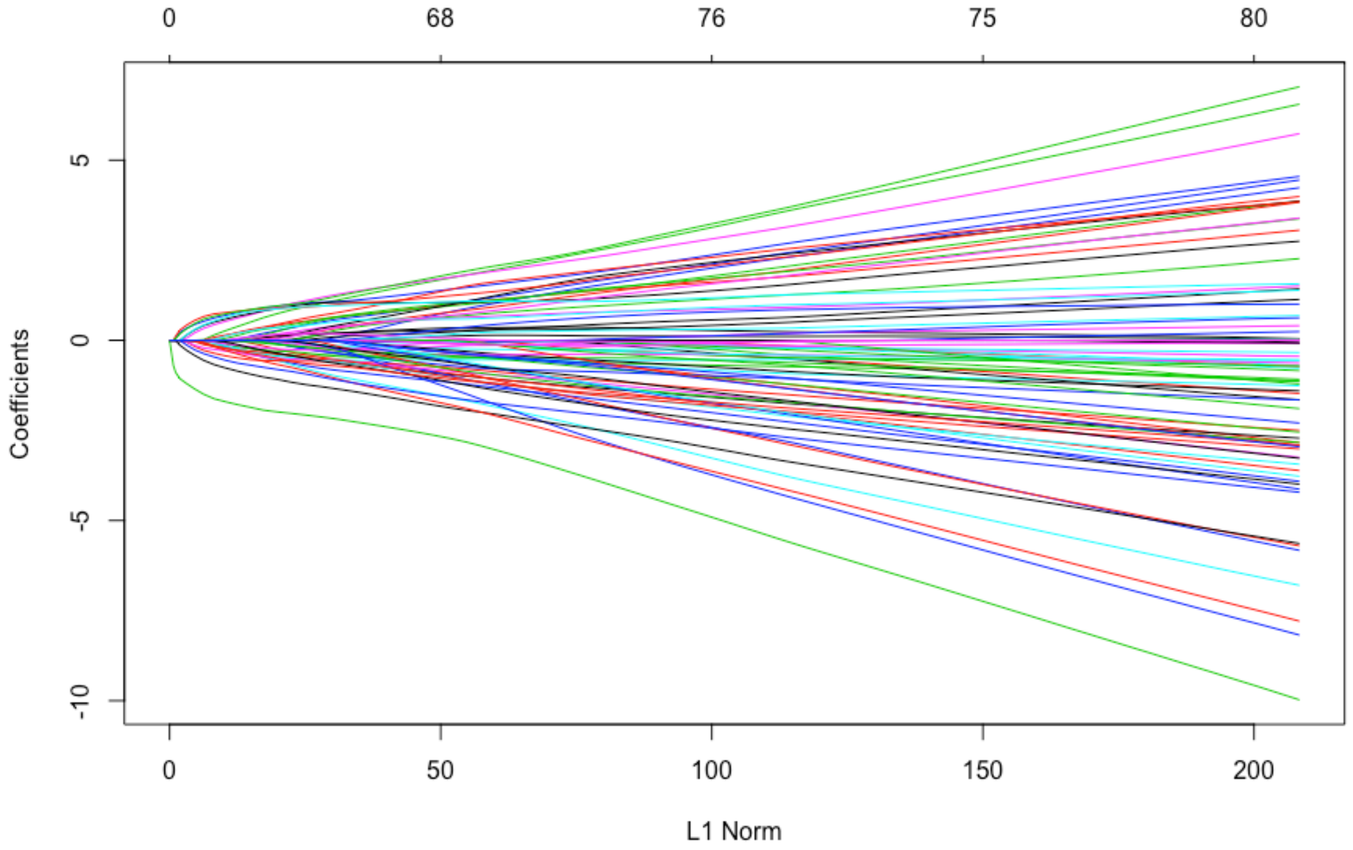
```
# Make a custom tuning grid
> myGrid <- expand.grid(
    alpha = 0:1,
    lambda = seq(0.0001, 0.1, length = 10)
  )

# 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)
```



MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

Median imputation

Dealing with missing values

- Most models require numbers, can't handle missing data
- Common approach: remove rows with missing data
  - Can lead to biases in data
  - Generate over-confident models
- Better strategy: median imputation!
  - Replace missing values with medians
  - Works well if data *missing at random* (MAR)

Example: mtcars

```
# Generate some data with missing values
> data(mtcars)
> set.seed(42)
> mtcars[sample(1:nrow(mtcars), 10), "hp"] <- NA

# Split target from predictors
> Y <- mtcars$mpg
> X <- mtcars[, 2:4]

# Try to fit a caret model
> library(caret)
> model <- train(x = X, y = Y)
Error in train.default(x = X, y = Y) : Stopping
```

A simple solution

```
# Now fit with median imputation
> model <- train(x = X, y = Y, preProcess = "medianImpute")
> print(model)
Random Forest

32 samples
 3 predictor

Pre-processing: median imputation (3)
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 32, 32, 32, 32, 32, 32, ...
Resampling results across tuning parameters:

  mtry  RMSE      Rsquared
  2     2.617096  0.8234652
  3     2.670550  0.8164535

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was mtry = 2.
```

MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

KNN imputation

Dealing with missing values

- Median imputation is fast, but...
- Can produce incorrect results if data *missing not at random*
- k-nearest neighbors (KNN) imputation
- Imputes based on "similar" non-missing rows

Example: missing not at random

- Pretend smaller cars don't report horsepower
- Median imputation incorrect in this case

Assumes small cars have medium-large horsepower

```
# Generate data with missing values
> data(mtcars)
> mtcars[mtcars$disp < 140, "hp"] <- NA
> Y <- mtcars$mpg
> X <- mtcars[, 2:4]

# Use median imputation
> set.seed(42)
> model <- train(x = X, y = Y, method = "glm",
                 preProcess = "medianImpute")
> print(min(model$results$RMSE))
[1] 3.612713
```

Example: missing not at random

- KNN imputation is better
- Uses cars with similar `disp` / `cyl` to impute
- Yields a more accurate (but slower) model

```
# Use KNN imputation
> set.seed(42)
> model <- train(x = X, y = Y,
                 method = "glm",
                 preProcess = "knnImpute"
  )
> print(min(model$results$RMSE))
[1] 3.558881
```

Compare to 3.61 for median imputation

MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

Multiple preprocessing methods

The wide world of `preProcess`

- You can do a lot more than median or knn imputation!
- Can chain together multiple preprocessing steps
- Common "recipe" for linear models (order matters!)

  Median imputation -> center -> scale -> fit `glm`
- See `?preProcess` for more detail

Example: preprocessing `mtcars`

```
# Generate some data with missing values
> data(mtcars)
> set.seed(42)
> mtcars[sample(1:nrow(mtcars), 10), "hp"] <- NA
> Y <- mtcars$mpg
> X <- mtcars[,2:4]

# Use linear model "recipe"
> set.seed(42)
> model <- train(
    x = X, y = Y, method = "glm",
    preProcess = c("medianImpute", "center", "scale")
  )
> print(min(model$results$RMSE))
[1] 3.612713
```

Missing at random

Example: preprocessing `mtcars`

```
# PCA before modeling
> set.seed(42)
> model <- train(
    x = X, y = Y, method = "glm",
    preProcess = c("medianImpute", "center", "scale", "pca")
  )
> min(model$results$RMSE)
[1] 3.402557
```

Example: preprocessing `mtcars`

```
# Spatial sign transform
> set.seed(42)
> model <- train(
    x = X, y = Y, method = "glm",
    preProcess = c("medianImpute", "center", "scale", "spatialSign"))
> min(model$results$RMSE)
[1] 4.284904
```

Preprocessing cheat sheet

- Start with median imputation

Try KNN imputation if data missing not at random

- For linear models...
  - Center and scale
  - Try PCA and spatial sign
- Tree-based models don't need much preprocessing

MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

Handling low-information predictors

No (or low) variance variables

- Some variables don't contain much information
  - Constant (i.e. no variance)
  - Nearly constant (i.e. low variance)
- Easy for one fold of CV to end up with constant column
- Can cause problems for your models
- Usually remove extremely low variance variables

Example: constant column in `mtcars`

```
# Reproduce dataset from last video
> data(mtcars)
> set.seed(42)
> mtcars[sample(1:nrow(mtcars), 10), "hp"] <- NA
> Y <- mtcars$mpg
> X <- mtcars[, 2:4]

# Add constant-valued column to mtcars
> X$bad <- 1
```

Example: constant column in `mtcars`

```
# Try to fit a model with PCA + glm
> model <- train(
    x = X, y = Y, method = "glm",
    preProcess = c("medianImpute", "center", "scale", "pca")
  )

Warning in preProcess.default(thresh = 0.95, k = 5, method =
c("medianImpute",  :
  These variables have zero variances: bad
Something is wrong; all the RMSE metric values are missing:
      RMSE        Rsquared
 Min.   : NA   Min.   : NA
 1st Qu.: NA   1st Qu.: NA
 Median : NA   Median : NA
 Mean   :NaN   Mean   :NaN
 3rd Qu.: NA   3rd Qu.: NA
 Max.   : NA   Max.   : NA
 NA's   :1     NA's   :1
```

`caret` to the rescue (again)

- `"zv"` removes constant columns
- `"nzv"` removes nearly constant columns

```
# Have caret remove those columns during modeling
> set.seed(42)
> model <- train(
    x = X, y = Y, method = "glm",
    preProcess = c("zv", "medianImpute", "center", "scale", "pca")
  )
> min(model$results$RMSE)
[1] 3.402557
```

MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

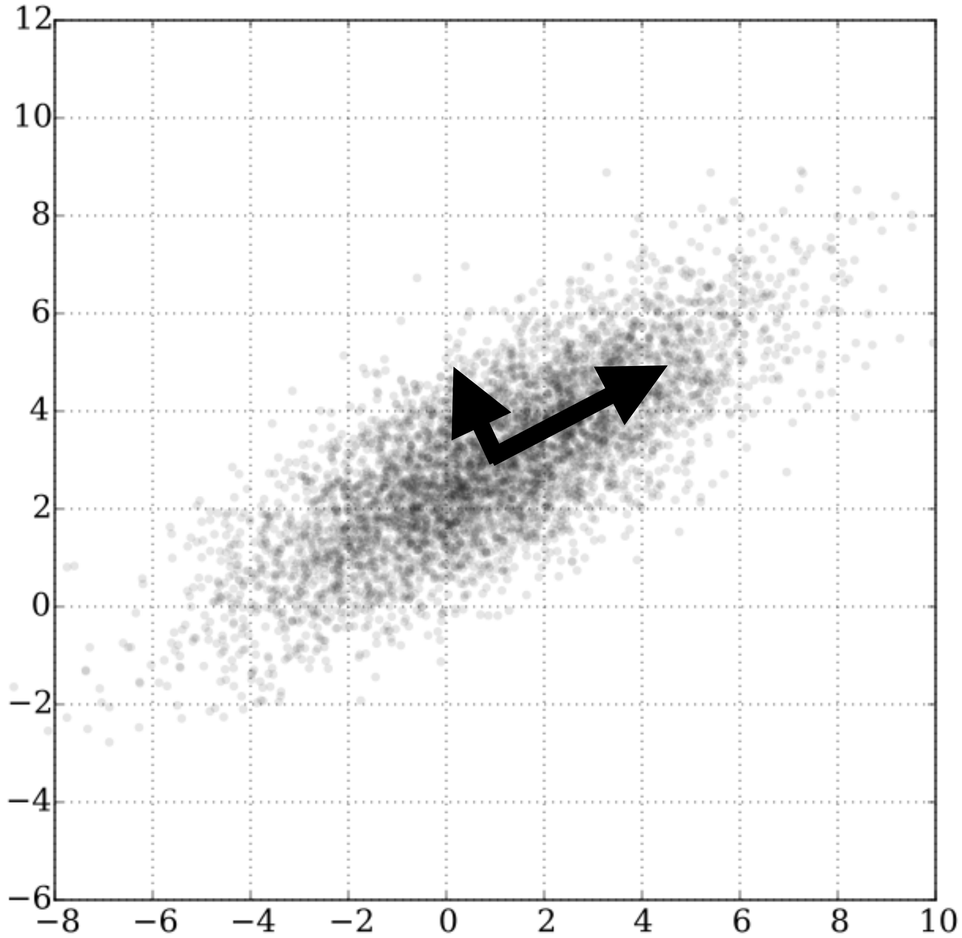
Principle components analysis (PCA)

Principle components analysis

- Combines low-variance and correlated variables
- Single set of high-variance, perpendicular predictors
- Prevents collinearity (i.e. correlation among predictors)

PCA: a visual representation

- First component has highest variance
- Second component has second highest variance
- And so on...



Example: blood-brain data

- Lots of predictors
- Many of them low-variance

```
# Load the blood brain dataset
> data(BloodBrain)
> names(bbbDescr)[nearZeroVar(bbbDescr)]
[1] "negative"     "peoe_vsa.2.1" "peoe_vsa.3.1" "a_acid"
[5] "vsa_acid"     "frac.anion7." "alert"
```

Example: blood-brain data

```
# Basic model
> set.seed(42)
> data(BloodBrain)
> model <- train(
    x = bbbDescr, y = logBBB, method = "glm",
    trControl = trainControl(method = "cv", number = 10, verbose = TRUE),
    preProcess = c("zv", "center", "scale")
  )
> min(model$results$RMSE)
[1] 1.107702
```

Example: blood-brain data

```
# Remove low-variance predictors
> set.seed(42)
> data(BloodBrain)
> model <- train(
    x = bbbDescr, y = logBBB, method = "glm",
    trControl = trainControl(method = "cv", number = 10, verbose = TRUE),
    preProcess = c("nzv", "center", "scale")
  )
> min(model$results$RMSE)
[1] 0.9796199
```

Example: blood-brain data

```
# Add PCA
> set.seed(42)
> data(BloodBrain)
> model <- train(
    x = bbbDescr, y = logBBB, method = "glm",
    trControl = trainControl(method = "cv", number = 10, verbose = TRUE),
    preProcess = c("zv", "center", "scale", "pca")
  )
> min(model$results$RMSE)
[1] 0.9796199
```

MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

Reusing a `trainControl`

A real-world example

- The data: customer churn at telecom company
- Fit different models and choose the best
- Models must use the same training/test splits
- Create a shared `trainControl` object

Example: customer churn data

```
# Summarize the target variables
> library(caret)
> library(C50)
> data(churn)
> table(churnTrain$churn) / nrow(churnTrain)

      yes        no
0.1449145 0.8550855

# Create train/test indexes
> set.seed(42)
> myFolds <- createFolds(churnTrain$churn, k = 5)

# Compare class distribution
> i <- myFolds$Fold1
> table(churnTrain$churn[i]) / length(i)

      yes        no
0.1441441 0.8558559
```

Example: customer churn data

```
> myControl <- trainControl(
    summaryFunction = twoClassSummary,
    classProbs = TRUE,
    verboseIter = TRUE,
    savePredictions = TRUE,
    index = myFolds
  )
```

- Use folds to create a `trainControl` object
- Exact same cross-validation folds for each model

MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

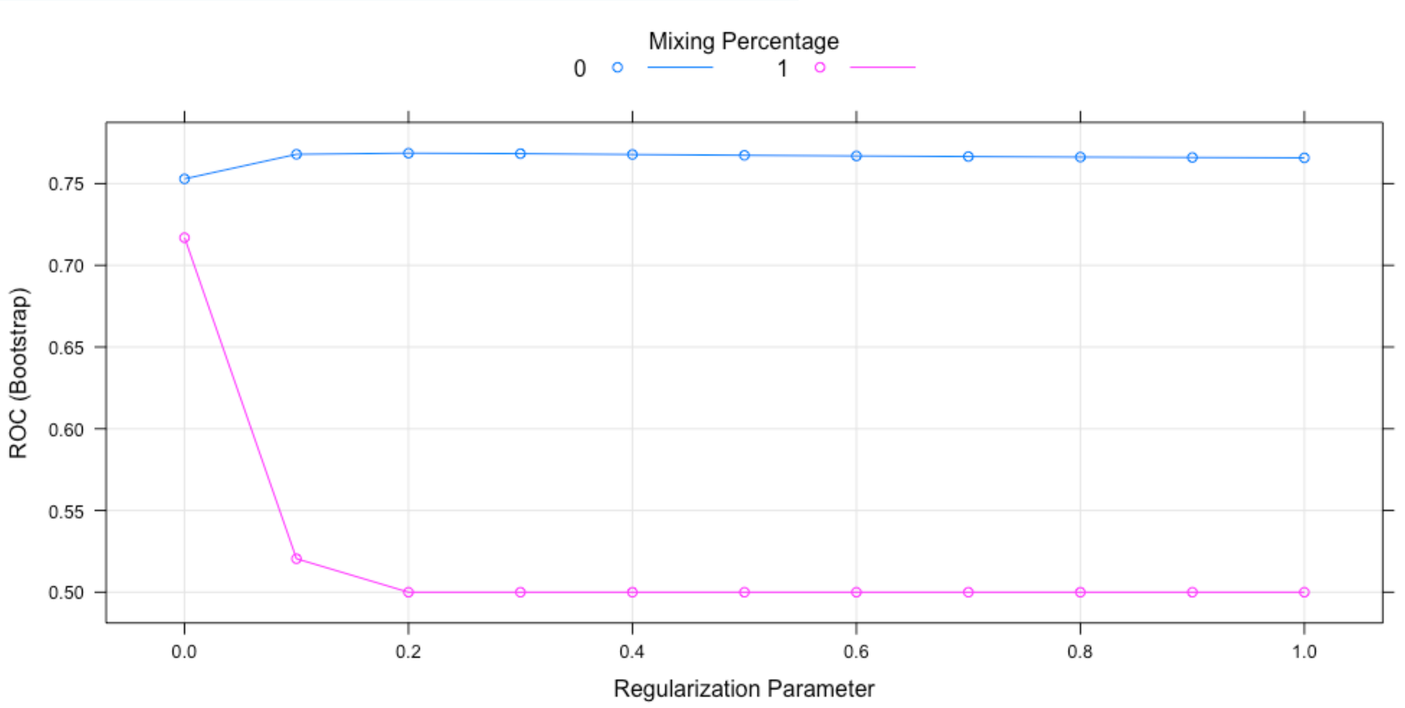
Reintroduce `glmnet`

`glmnet` review

- Linear model with built-in variable selection
- Great baseline model
- Advantages
  - Fits quickly
  - Ignores noisy variables
  - Provides interpretable coefficients

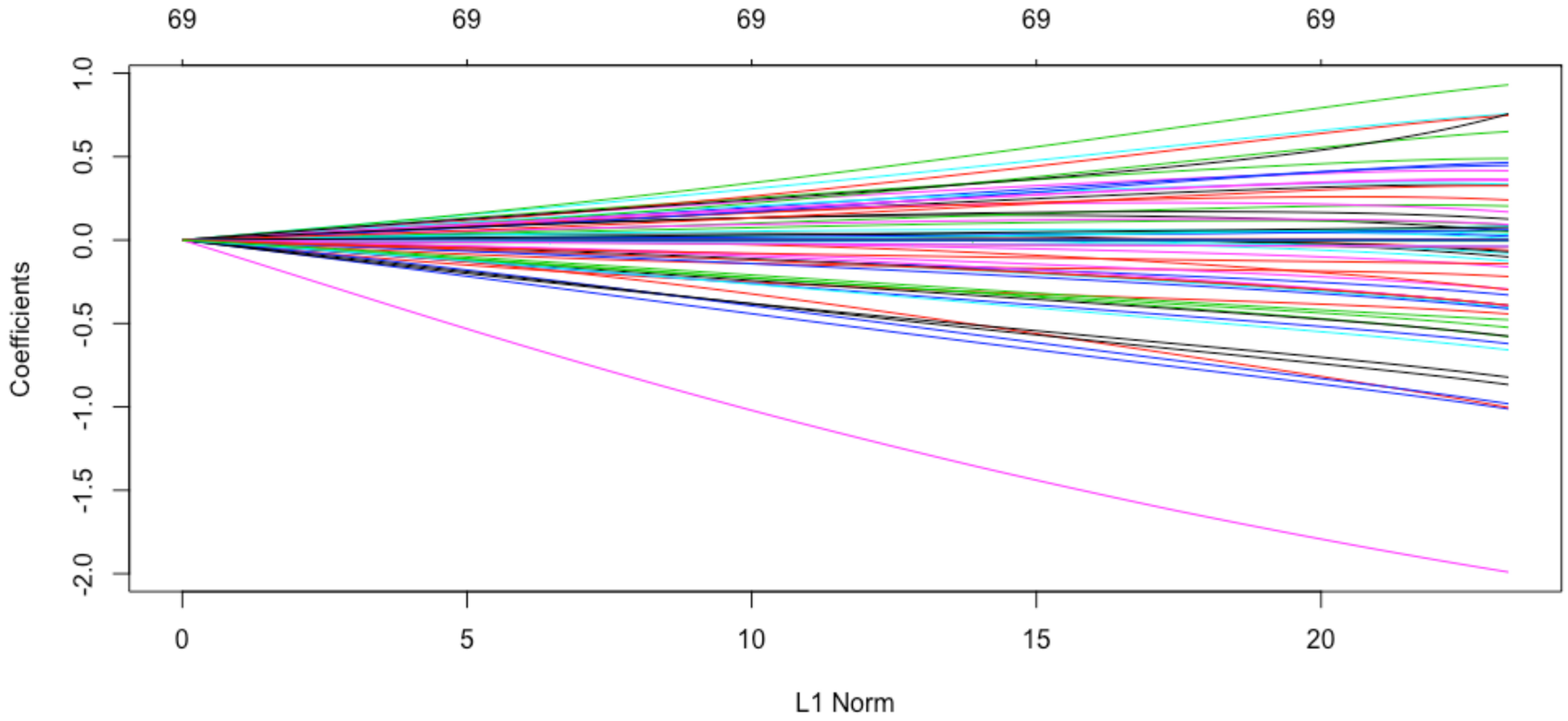
Example: `glmnet` on churn data

```
> # Fit the model
> set.seed(42)
> model_glmnet <- train(
    churn ~ ., churnTrain,
    metric = "ROC",
    method = "glmnet",
    tuneGrid = expand.grid(
      alpha = 0:1,
      lambda = 0:10/10
    ),
    trControl = myControl
  )

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



Plot the coefficients

```
> plot(model_glmnet$finalModel)
```



MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

Reintroduce random forest

Random forest review

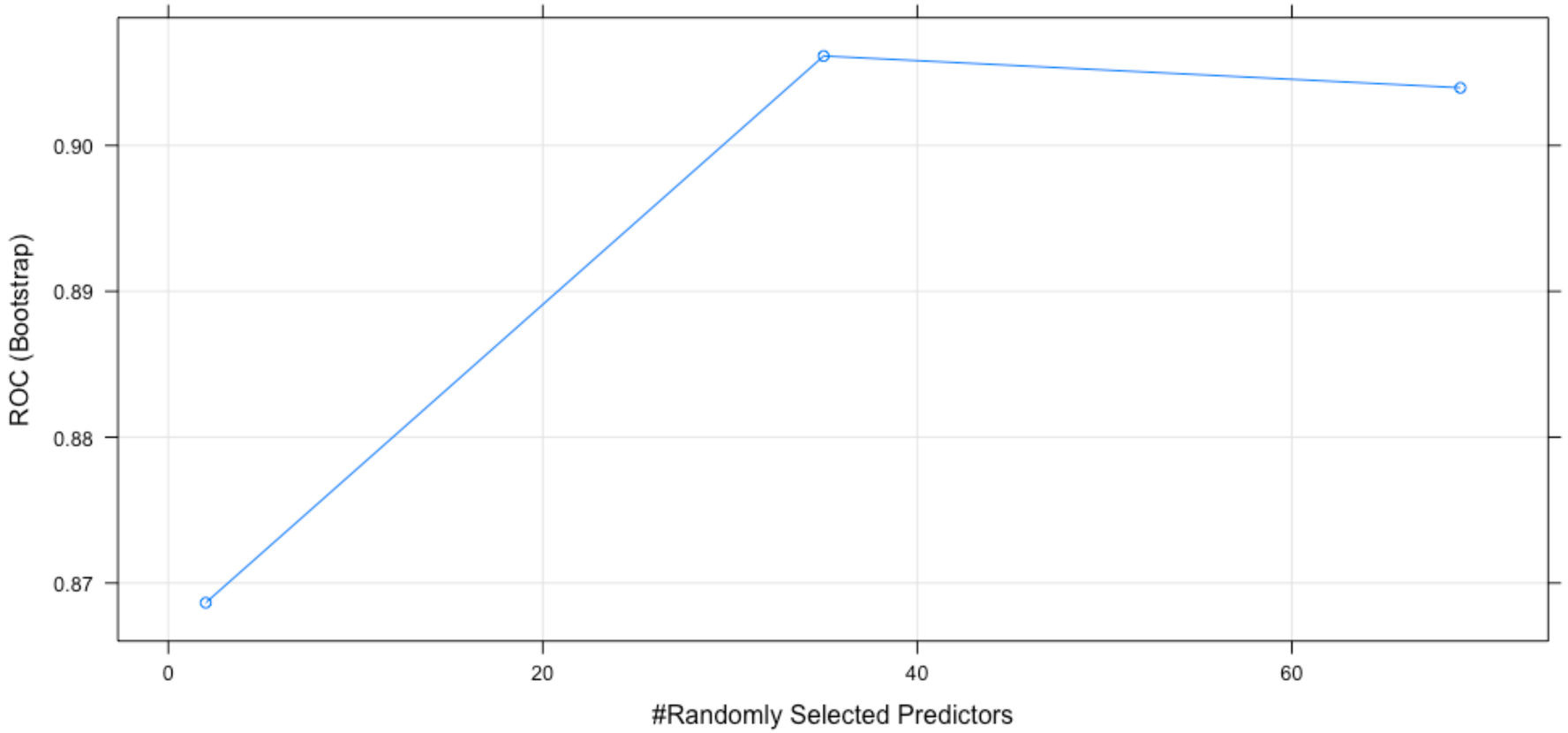
- Slower to fit than `glmnet`
- Less interpretable
- Often (but not always) more accurate than `glmnet`
- Easier to tune
- Require little preprocessing
- Capture threshold effects and variable interactions

Random forest on churn data

```
> set.seed(42)
> churnTrain$churn <- factor(churnTrain$churn, levels = c("no", "yes"))
> model_rf <- train(
    churn ~ ., churnTrain,
    metric = "ROC",
    method = "ranger",
    trControl = myControl
)
```

Random forest on churn data

```
> plot(model_rf)
```



MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

Comparing models

Comparing models

- Make sure they were fit on the same data!
- Selection criteria
  - Highest average AUC
  - Lowest standard deviation in AUC
- The `resamples()` function is your friend

Example: `resamples()` on churn data

```
# Make a list
> model_list <- list(
    glmnet = model_glmnet,
    rf = model_rf
  )

# Collect resamples from the CV folds
> resamps <- resamples(model_list)
> resamps

Call:
resamples.default(x = model_list)

Models: glmnet, rf
Number of resamples: 5
Performance metrics: ROC, Sens, Spec
Time estimates for: everything, final model fit
```

Summarize the results

```
# Summarize the results
> summary(resamps)
Call:
summary.resamples(object = resamps)

Models: glmnet, rf
Number of resamples: 5

ROC
          Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
glmnet 0.7526  0.7624 0.7719 0.7686  0.7722 0.7840    0
rf     0.8984  0.9028 0.9077 0.9061  0.9093 0.9125    0
```

MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

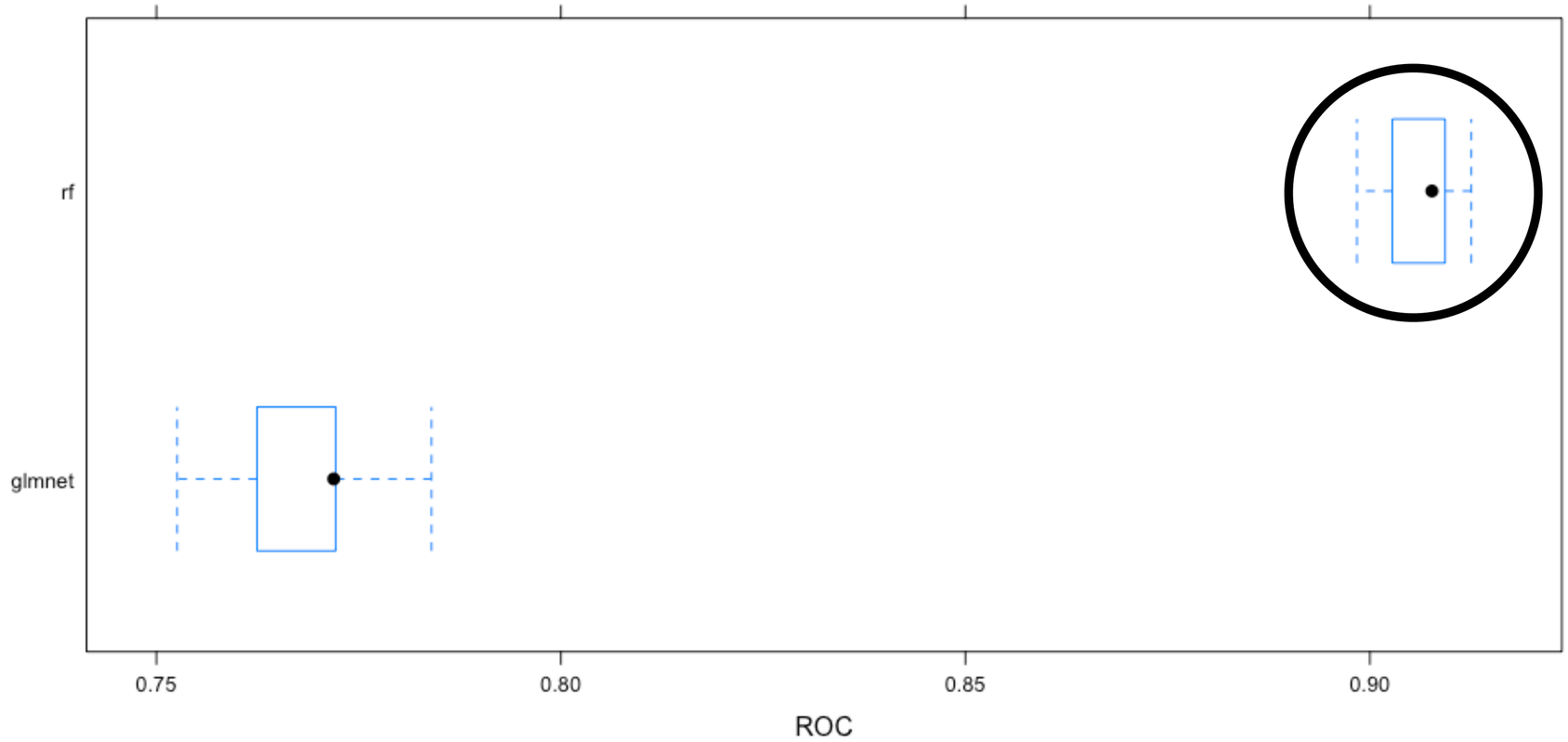
More on `resamples`

Comparing models

- Resamples has tons of cool methods
- One of my favorite functions (thanks Max!)
- Inspired the `caretEnsemble` package

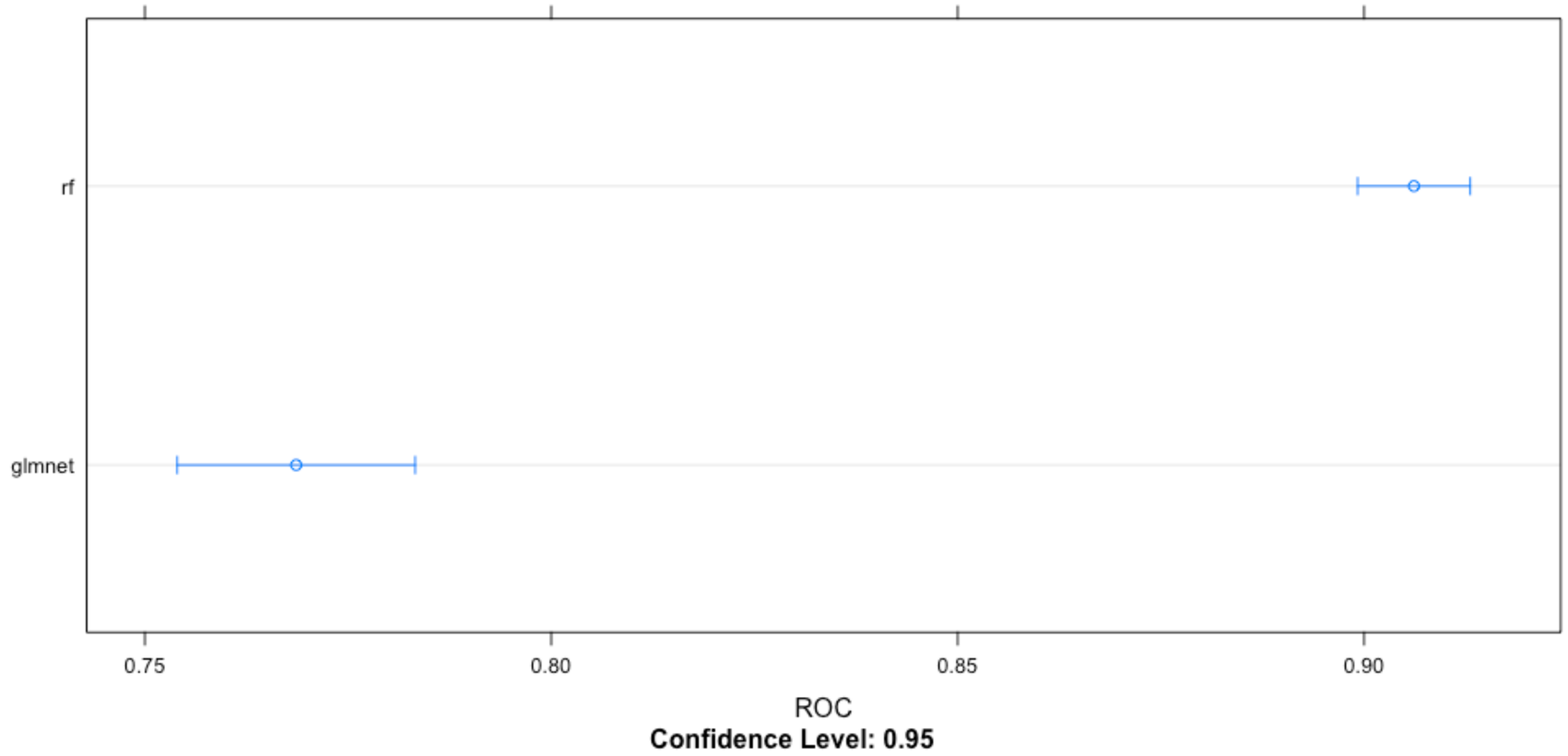
Box-and-whisker

```
> bwplot(resamps, metric = "ROC")
```



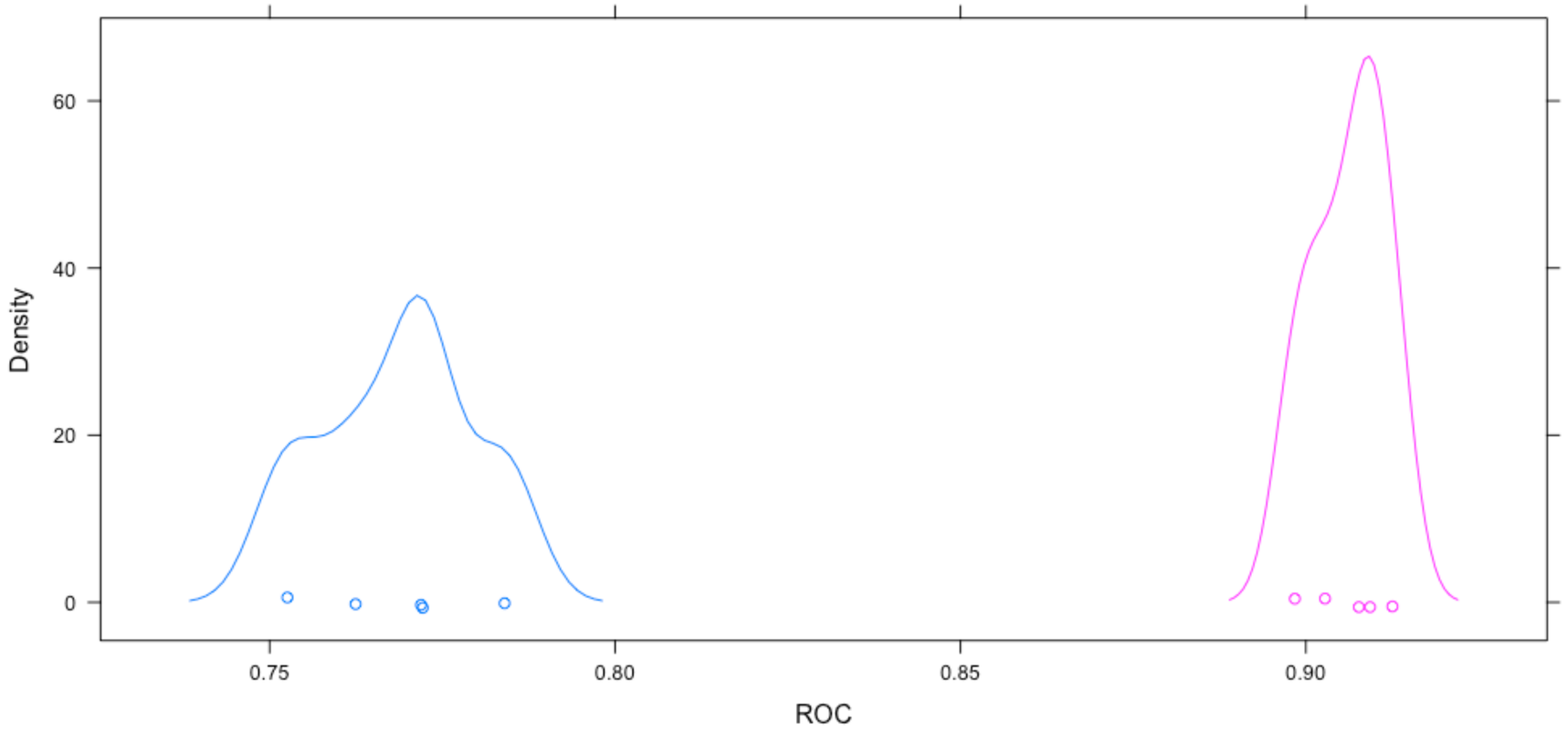
Dot plot

```
> dotplot(resamps, metric = "ROC")
```



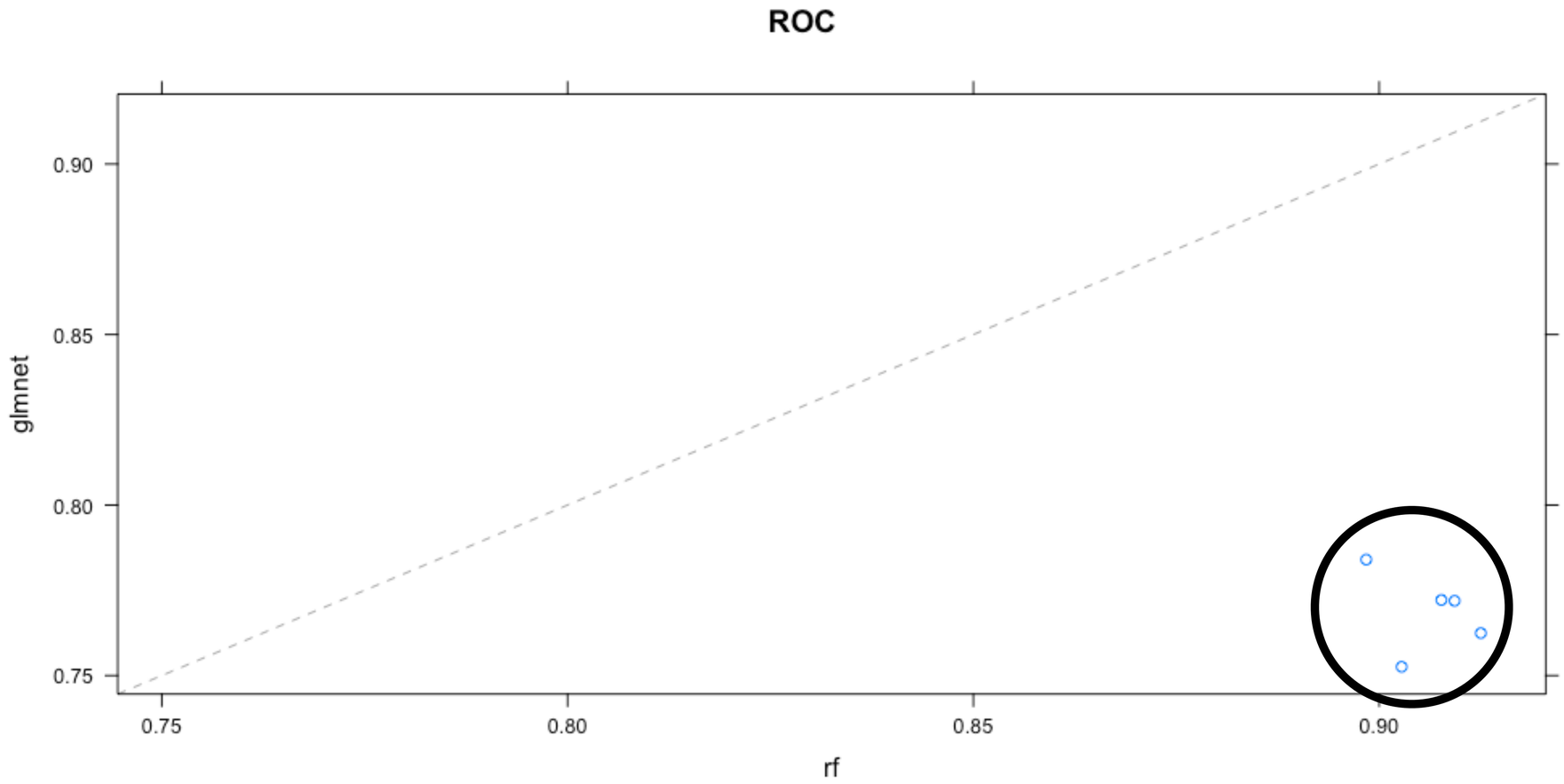
Density plot

```
> densityplot(resamps, metric = "ROC")
```



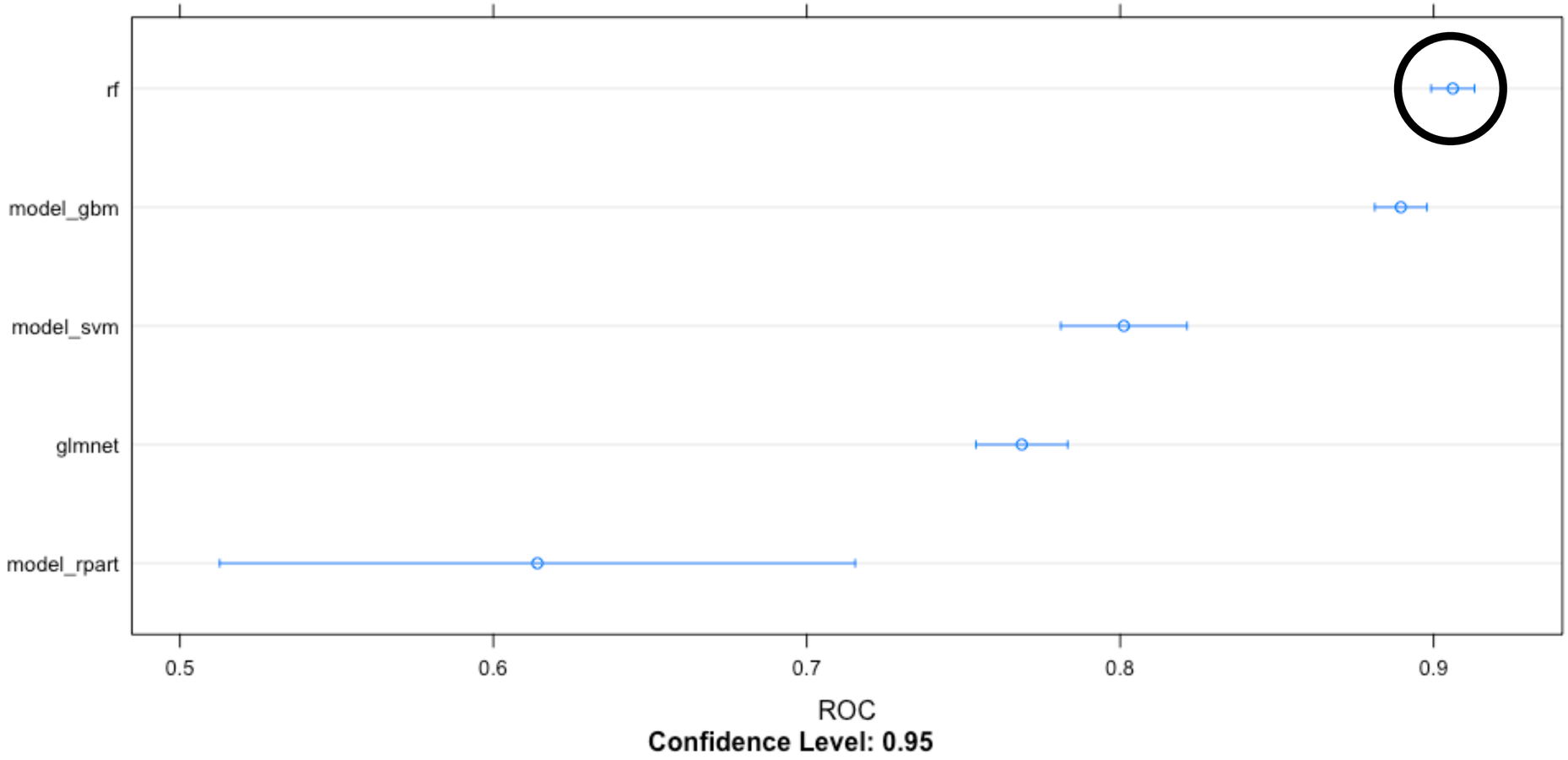
Scatter plot

```
> xyplot(resamps, metric = "ROC")
```



Another dot plot

```
> dotplot(lots_of_models, metric = "ROC")
```



MACHINE LEARNING TOOLBOX

Let’s practice!

MACHINE LEARNING TOOLBOX

Summary

What you've learned

- How to use the caret package
- Model fitting and evaluation
- Parameter tuning for better results
- Data preprocessing

Goals of the `caret` package

- Simplify the predictive modeling process
- Make it easy to try many models and techniques
- Provide common interface to many useful packages

MACHINE LEARNING TOOLBOX

Go build some models!