Machine Learning - The Caret Way

Enrique J. De La Hoz D.

Supervised Learning

- Caret R package
- Automates supervised learning (a.k.a. predictive modeling)
- Target variable

Supervised Learning

- Two types of predictive models
  - Classification
  - Regression
- 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 overfi!ing
- Better to calculate out-of-sample error (a la caret)
  - Simulates real-world usage
  - Helps avoid overfi!ing

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 overfi!ing
- Primary goal of caret and this course: don't overfit

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

Random Forest

A very Popular and powerful type of machine learning model.

- Good for beginners
- Robust to overfi!ing
- 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 o“en 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 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

Tuning and Refining with caret

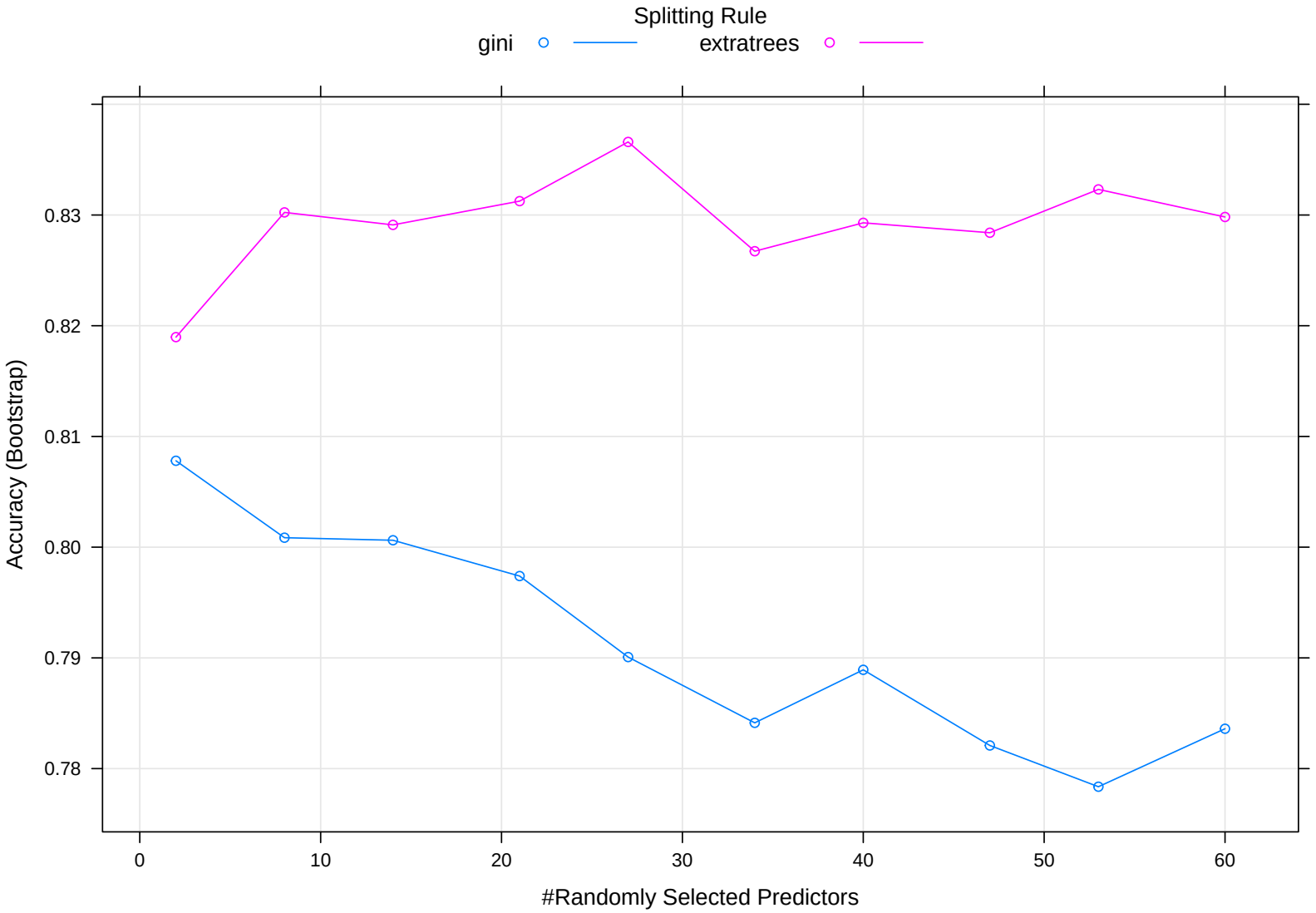
- Not only does caret do cross-validation...
- It also does grid search
- Select hyperparameters based on out-of-sample error

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



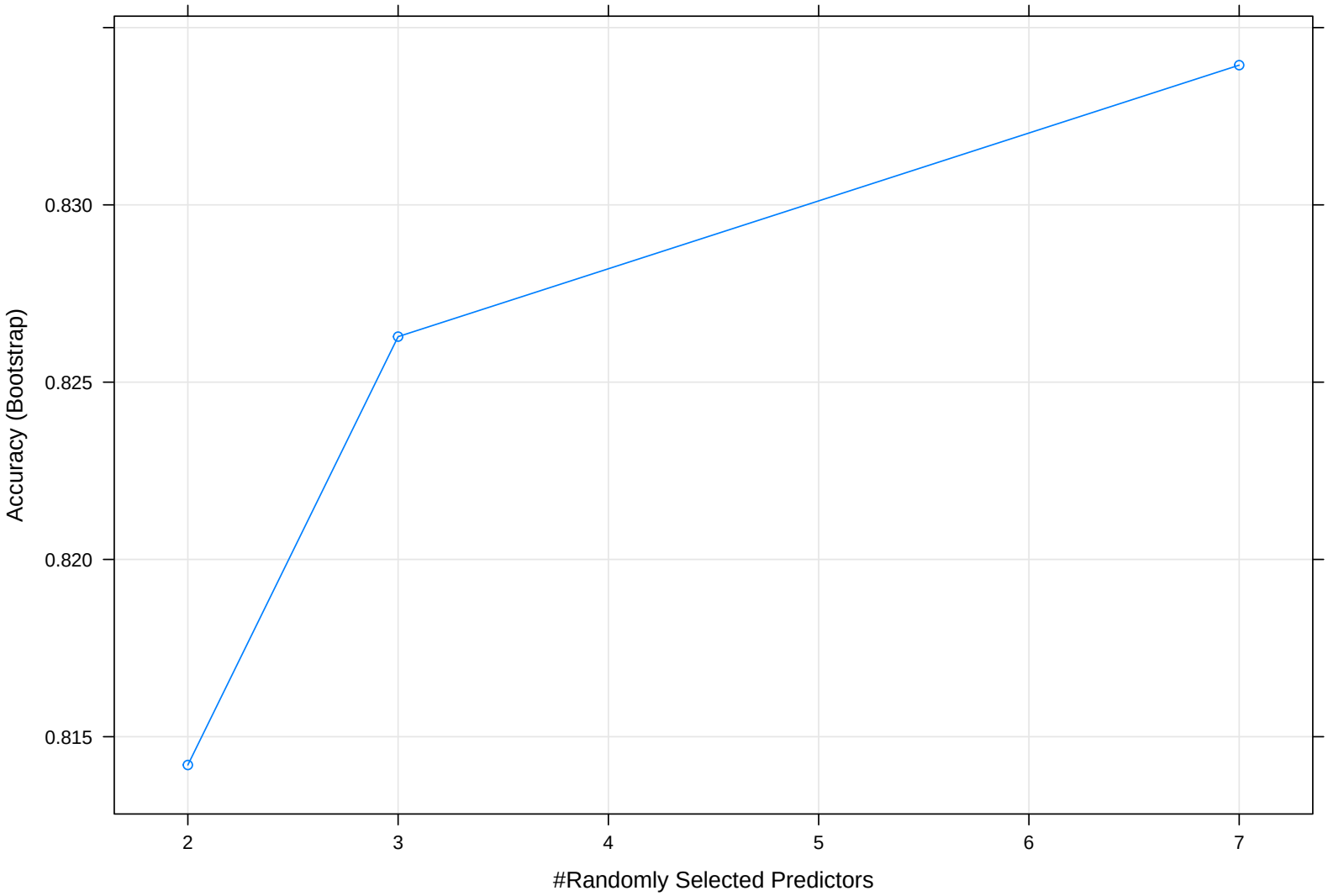
Pros and cons of custom tuning

- 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(2,3,7),
   .splitrule = "extratrees",
   .min.node.size = 5
)
# Fit a model with a custom tuning grid
 set.seed(42)
 model <- train(Class ~ ., data = Sonar, method = "ranger",
tuneGrid = myGrid, verbose = FALSE)
# Plot the results
 plot(model)
```

Custom tuning



Introduction to 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

A Hard dataset to model

```
# 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
)
```

Defatults values

```
# 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

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

```
+ Fold01: alpha=0.10, lambda=0.01013
- Fold01: alpha=0.10, lambda=0.01013
+ Fold01: alpha=0.55, lambda=0.01013
- Fold01: alpha=0.55, lambda=0.01013
+ Fold01: alpha=1.00, lambda=0.01013
- Fold01: alpha=1.00, lambda=0.01013
+ Fold02: alpha=0.10, lambda=0.01013
- Fold02: alpha=0.10, lambda=0.01013
+ Fold02: alpha=0.55, lambda=0.01013
- Fold02: alpha=0.55, lambda=0.01013
+ Fold02: alpha=1.00, lambda=0.01013
- Fold02: alpha=1.00, lambda=0.01013
```

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

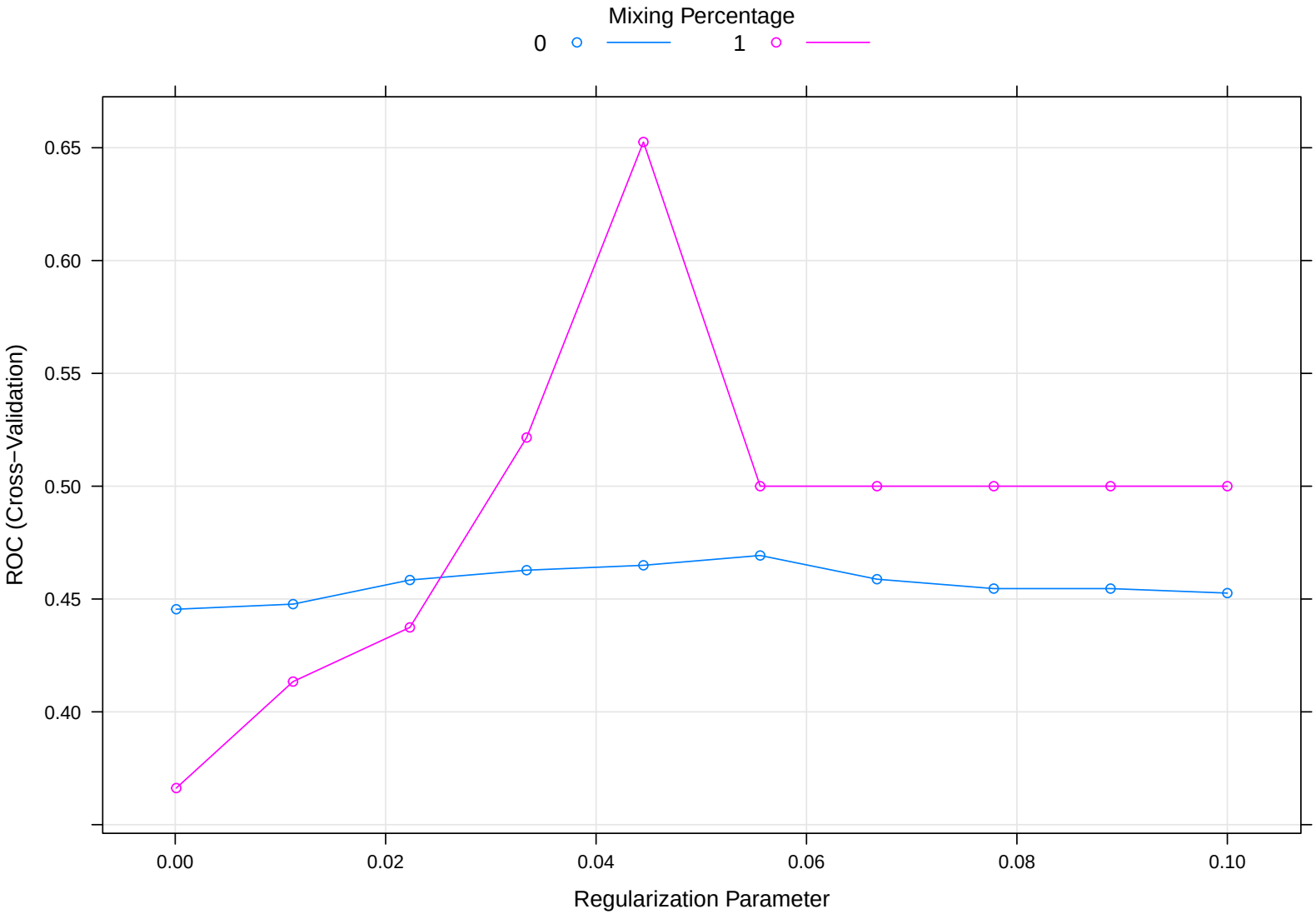
Example: glmnet tuning

```
# 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)
```

```
+ Fold01: alpha=0, lambda=0.1
- Fold01: alpha=0, lambda=0.1
+ Fold01: alpha=1, lambda=0.1
- Fold01: alpha=1, lambda=0.1
+ Fold02: alpha=0, lambda=0.1
- Fold02: alpha=0, lambda=0.1
+ Fold02: alpha=1, lambda=0.1
```

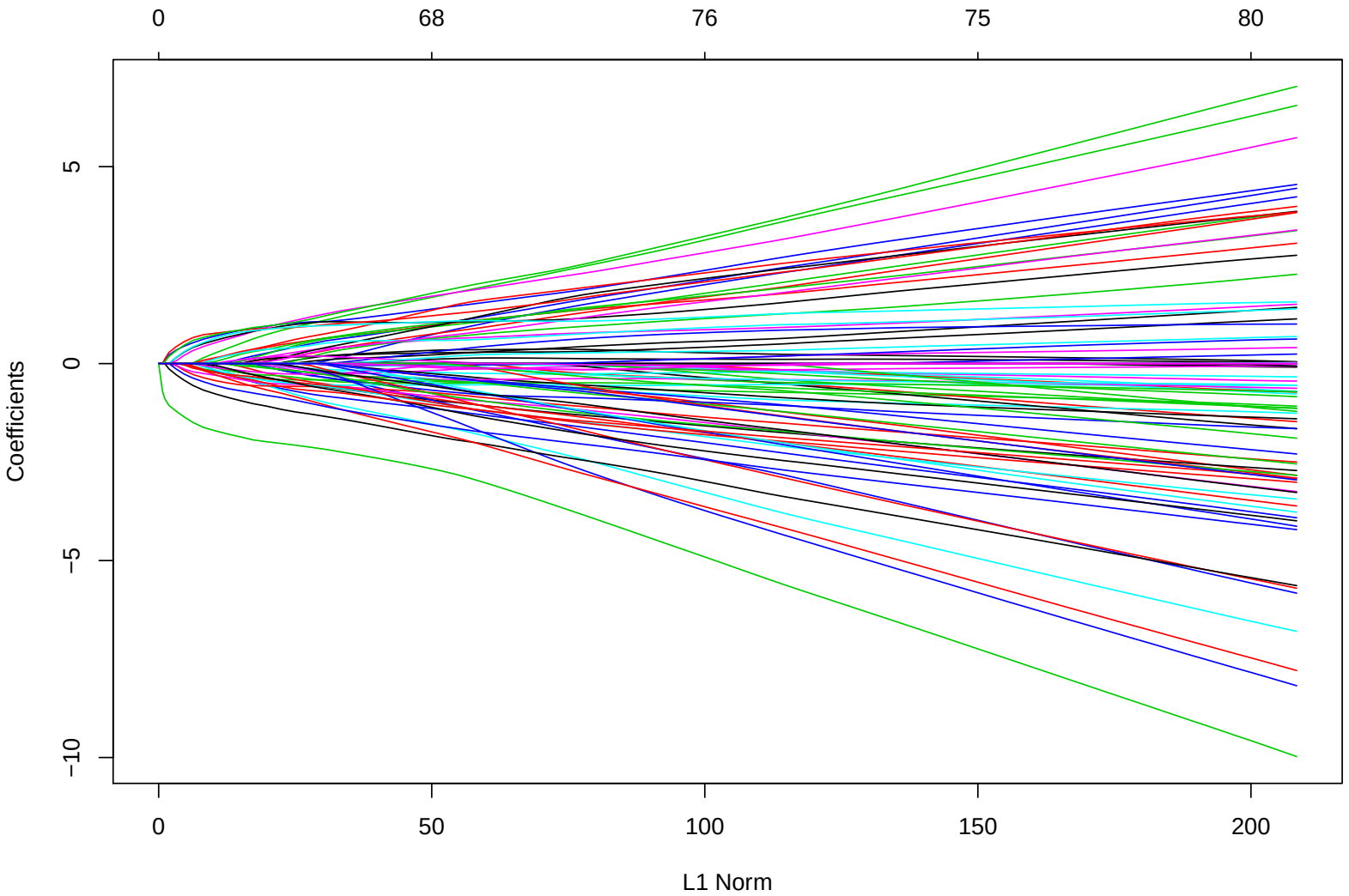
Compare Models visually

```
plot(model)
```



Full regularization path

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



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
- Be!er strategy: median imputation!
- Replace missing values with medians
- Works well if data missing at random (MAR)

Example: mtcars

```r
# 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)
```

```
note: only 2 unique complexity parameters in default grid. Tru

Something is wrong; all the RMSE metric values are missing:
      RMSE        Rsquared         MAE
 Min.   : NA   Min.   : NA   Min.   : NA
 1st Qu.: NA   1st Qu.: NA   1st Qu.: NA
```

Median Imputation Solution

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

```
note: only 2 unique complexity parameters in default grid. Tru
```

```
 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:
```

KNN Imputation

- 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

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

```
set.seed(42)
 model <- train(x = X, y = Y,
method = "glm",
preProcess = "knnImpute"
)
print(min(model$results$RMSE))
```

```
[1] 3.26861
```

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!)
- See ?preProcess for more detail
- 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] 3.332758
```

Example: preprocessing mtcars

```
# Generate some data with missing values
# 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] 4.080328
```

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

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$mpg
 X <- mtcars[, 2:4]
# Add constant-valued column to
X$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")
)
```

```
Something is wrong; all the RMSE metric values are missing:
      RMSE          Rsquared          MAE
 Min.   : NA   Min.   : NA   Min.   : NA
 1st Qu.: NA   1st Qu.: NA   1st Qu.: NA
 Median : NA   Median : NA   Median : NA
 Mean   :NaN   Mean   :NaN   Mean   :NaN
 3rd Qu.: NA   3rd Qu.: NA   3rd Qu.: NA
 Max.   : NA   Max.   : NA   Max.   : NA
 NA's   :1     NA's   :1     NA's   :1

Error: Stopping
```

Removing constant variables

- "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] 3.25045
```

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 = FALSE),
preProcess = c("zv", "center", "scale")
)
 min(model$results$RMSE)
```

```
[1] 1.094783
```

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 = FALSE),
preProcess = c("zv", "center", "scale", "pca")
)
 min(model$results$RMSE)
```

```
[1] 0.5601395
```

Applied Case: 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
```

Example: customer churn data

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

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

GLMNET Review

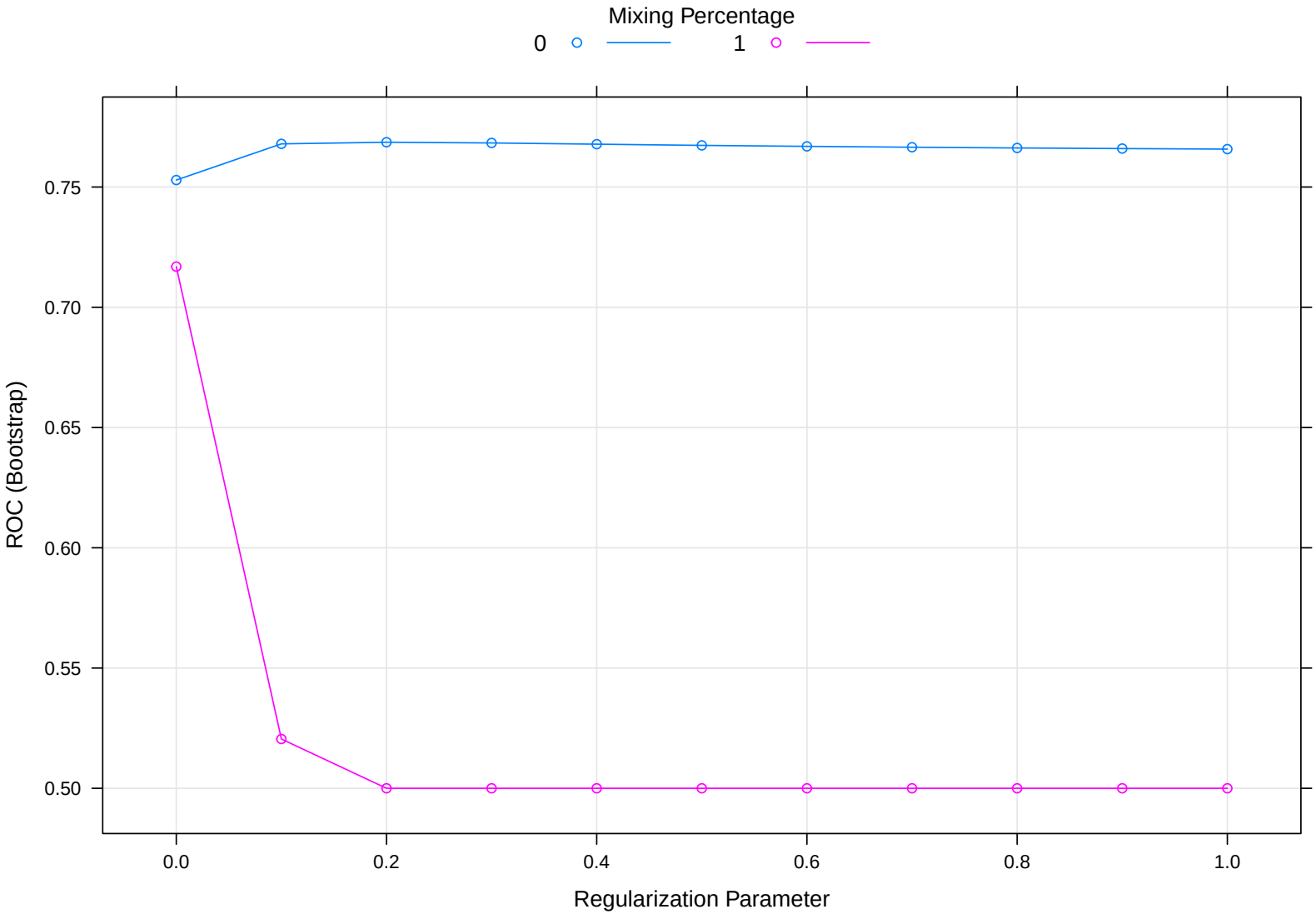
Linear model with built-in variable selection

- Great baseline model
- Advantages
  - Fits quickly
  - Ignores noisy variables
  - Provides interpretable coefficients

Example: glmnet on churn data

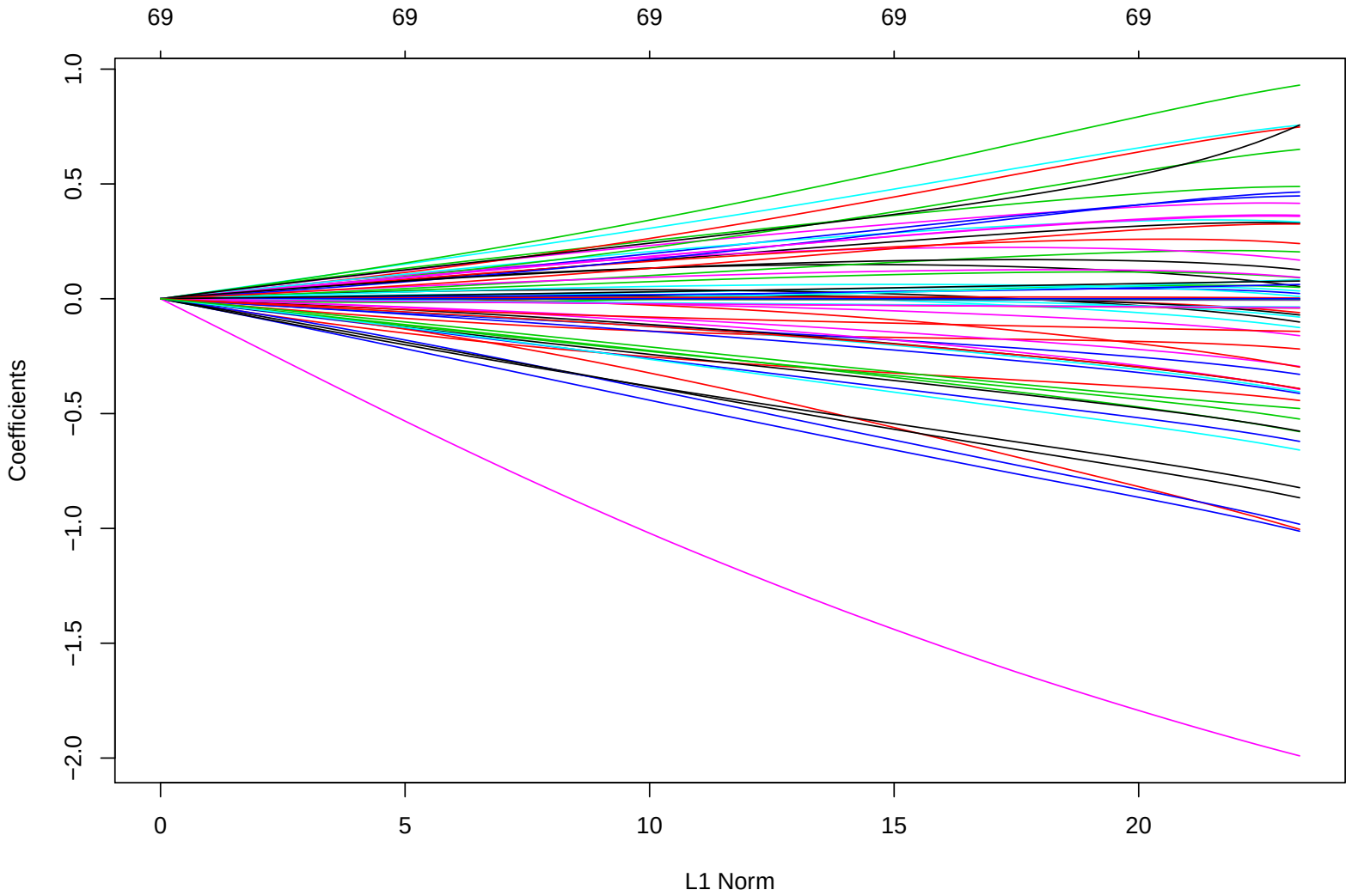
```
# Fit the model
set.seed(42)
model_glmnet <- train(
churn ~ ., churnTrain,
metric = "ROC",
method = "glmnet",
tuneGrid = expand.grid(
alpha = 0:1,
lambda = 0:10/10
),
trControl = myControl
)
# Plot the results
plot(model_glmnet)
```

Visual Inspection



Plot the coefficients

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



Random forest review

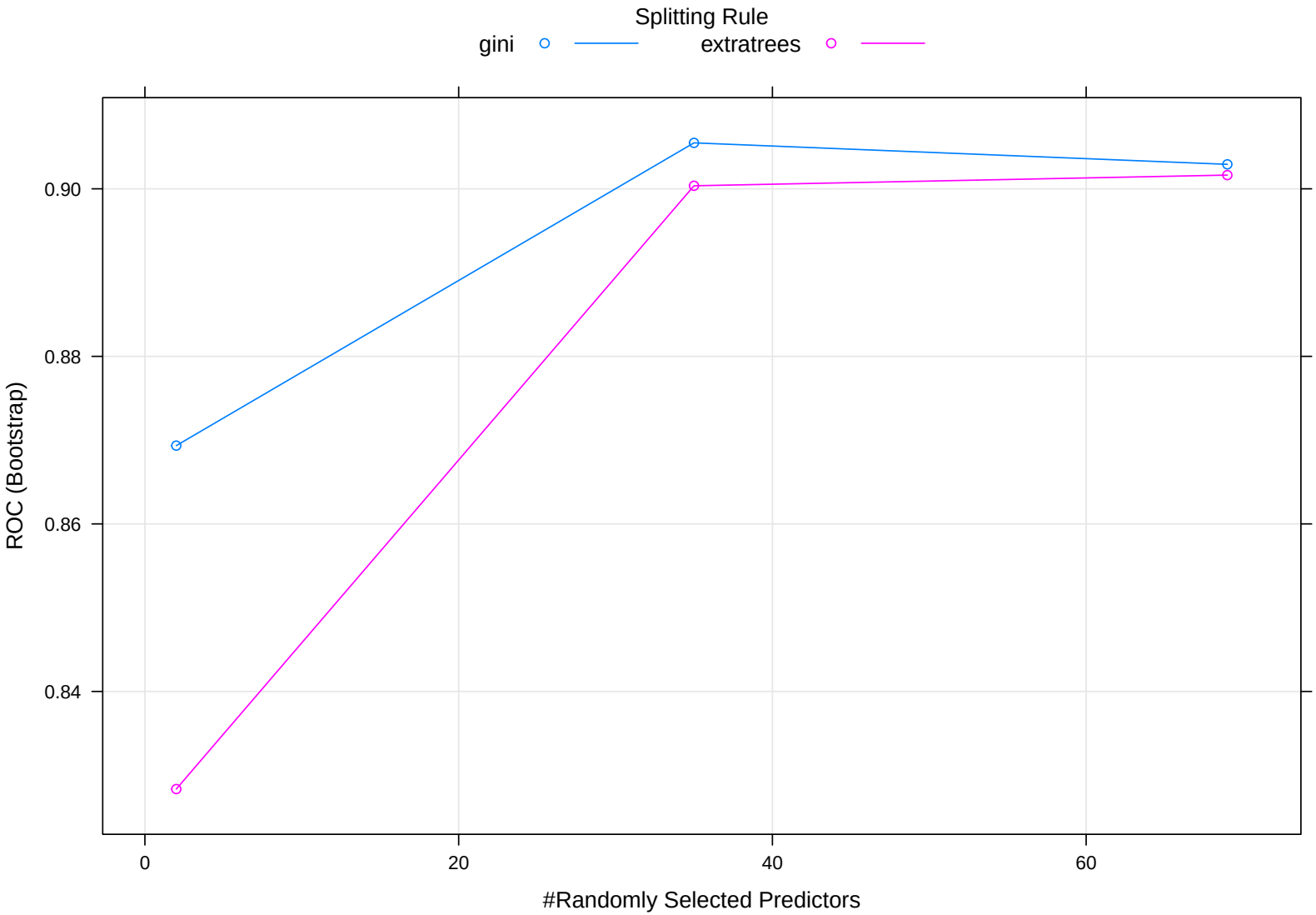
- Slower to fit than glmnet
- Less interpretable
- O#en (but not always) more accurate than glmnet
- Easier to tune
- Require li$le preprocessing
- Capture threshold effects and variable interactions

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)
```



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

Comparing models

```
# 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)
```

Summarize the results

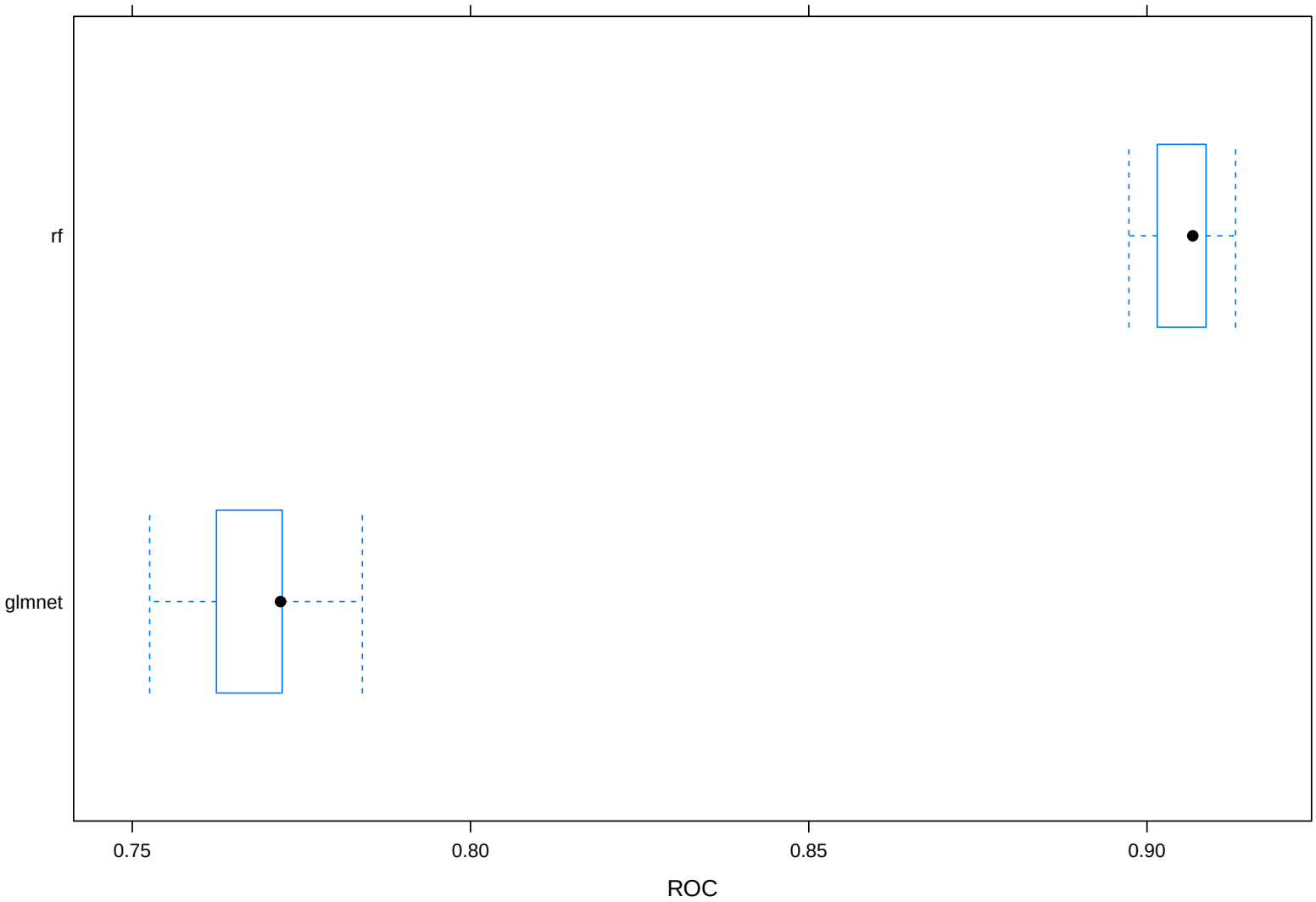
```
Call:
summary.resamples(object = resamps)

Models: glmnet, rf
Number of resamples: 5

ROC
            Min.   1st Qu.    Median      Mean   3rd Qu.
glmnet 0.7525668 0.7624405 0.7719151 0.7686177 0.7721599 0.784
rf     0.8973281 0.9015118 0.9067637 0.9054843 0.9087305 0.913

Sens
             Min.    1st Qu.     Median       Mean    3rd Qu.
glmnet 0.01036269 0.01295337 0.01808786 0.01759248 0.01813472
rf     0.98157895 0.98245614 0.98508772 0.98482456 0.98596491
       NA's
```

Comparing models

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

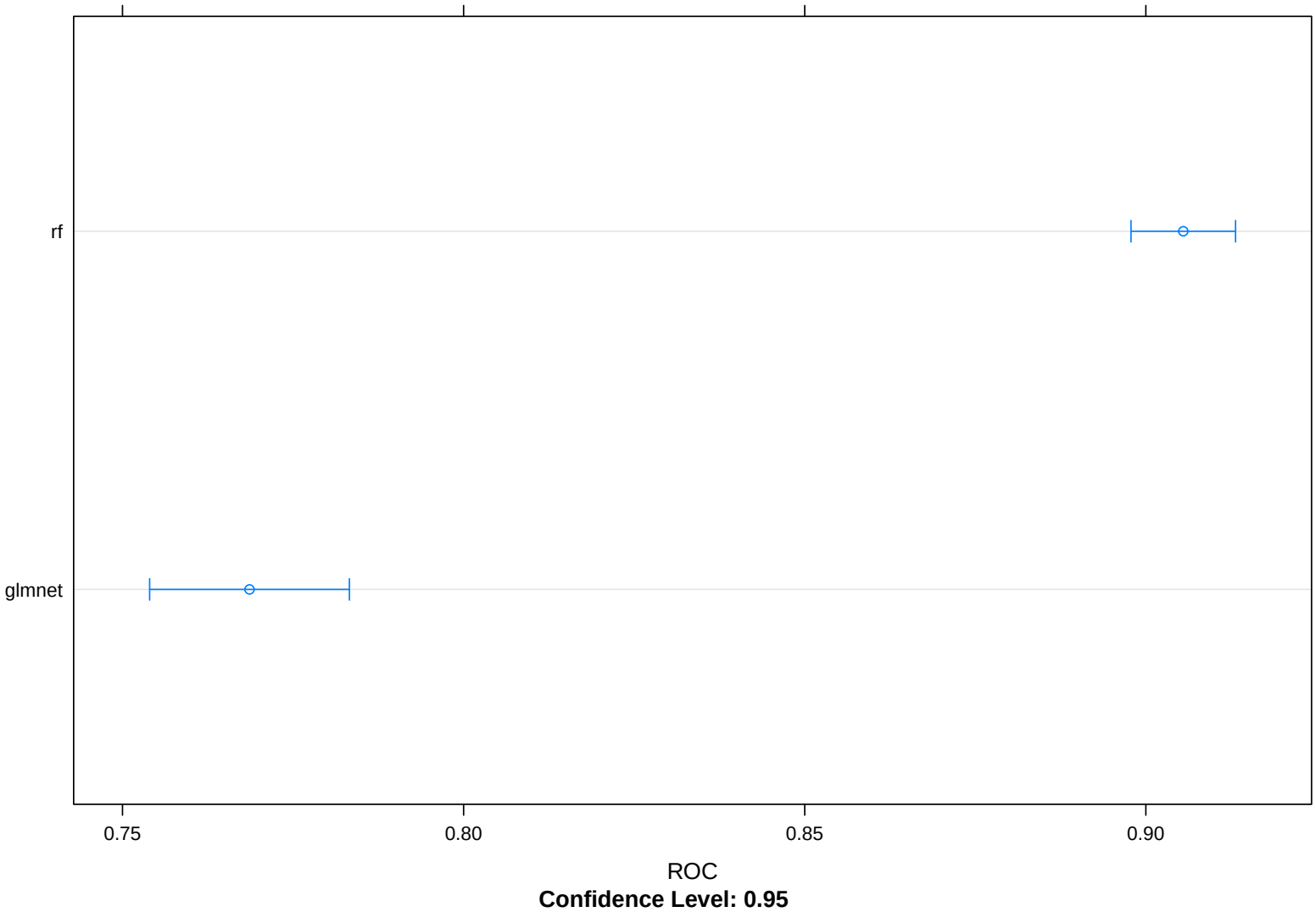
Box-and-whisker

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



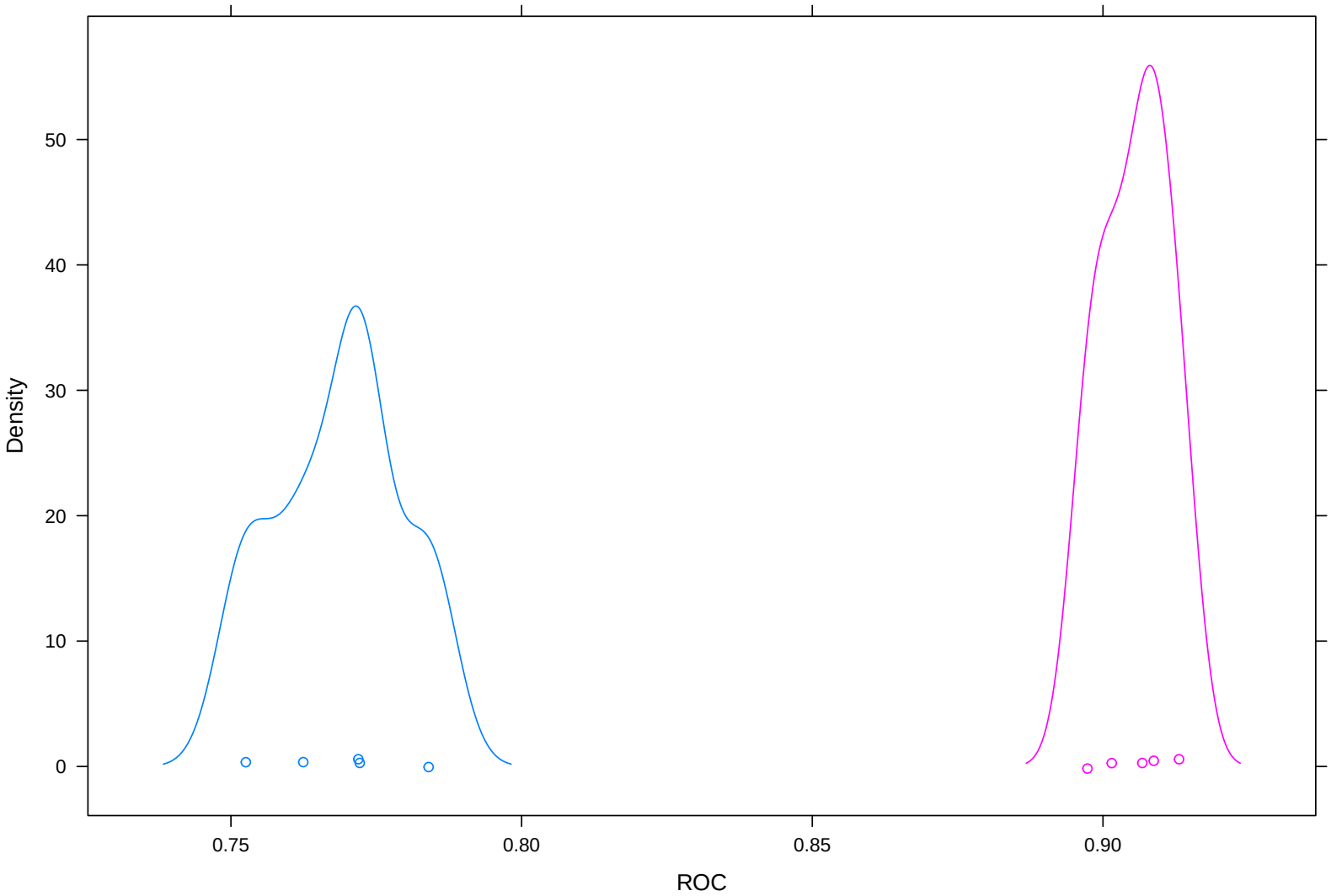
Dot plot

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



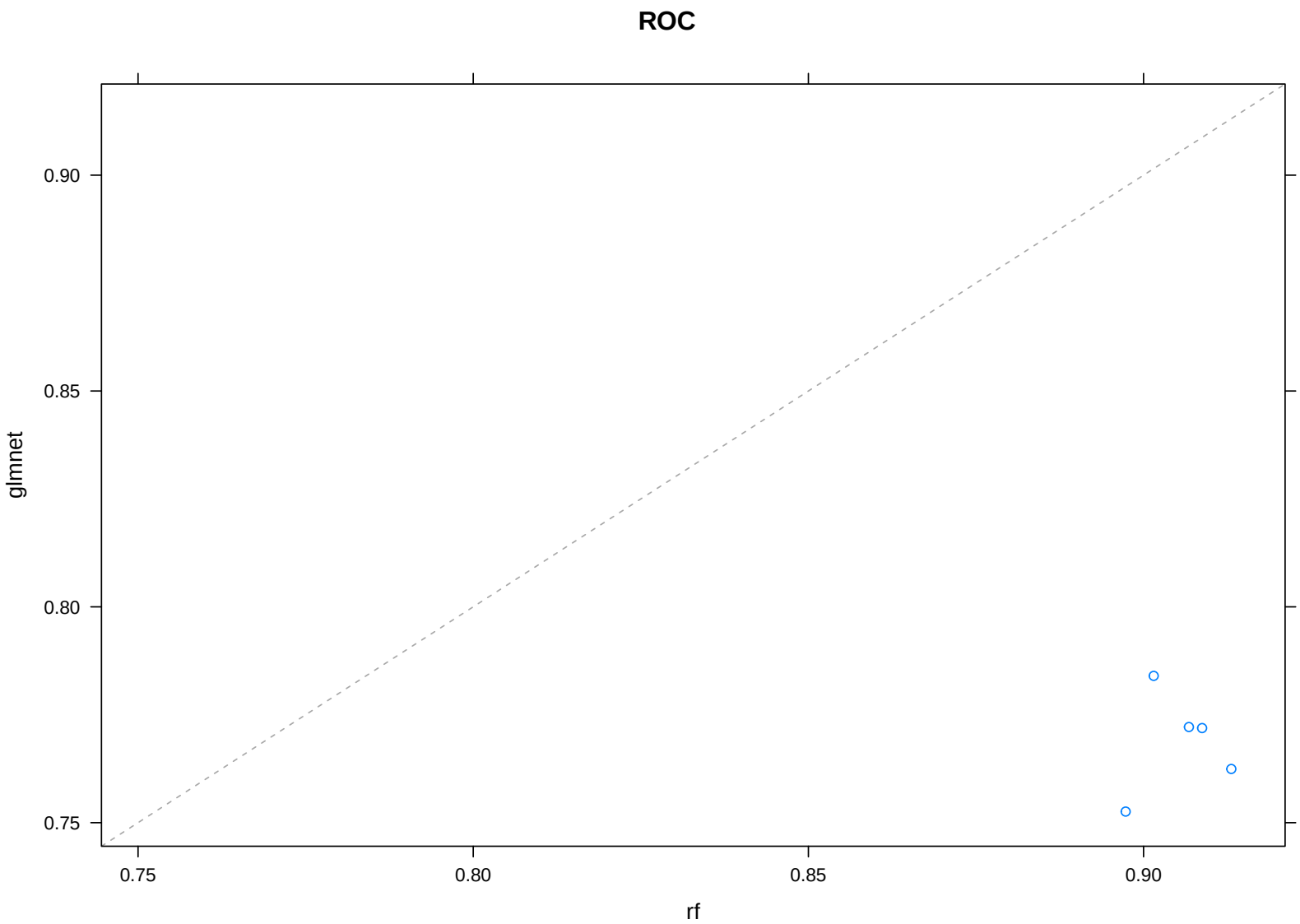
Density plot

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



Scatter plot

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



What can you do with Caret

- Model fitting and evaluation
- Parameter tuning for better results
- Data preprocessing
- Reproducible Research
- Paralell Computing
- Create a common interface for several packages

And a little bit of Advanced Ensemble Models

```
models <- caretList(iris[1:50,1:2], iris[1:50,3],
                    methodList=c("glmnet", "ranger"))
```

```
note: only 1 unique complexity parameters in default grid. Tru
```

```
ens <- caretEnsemble(models)
summary(ens)
```

```
The following models were ensembled: glmnet, ranger
They were weighted:
1.39 -0.3199 0.362
The resulting RMSE is: 0.1652
The fit for each individual model on the RMSE is:
 method      RMSE     RMSESD
 glmnet 0.1703281 0.03446505
 ranger 0.1754519 0.02561593
```