Machine Learning with caret in R

for Exercises

Eszter Katalin Bognar

02-06-2020

Regression models: fitting them and evaluating their performance

In-sample RMSE

```r
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```r
# Fit lm model: model
model <- lm(price ~., diamonds)

# Predict on full data: p
p <- predict(model)

# Compute errors: error
error <- p - diamonds$price

# Calculate RMSE
RMSE <- sqrt(mean(error^2))
RMSE
```

```
## [1] 1129.843
```

Out-of-sample RMSE

Randomly order the data frame

```r
# Set seed
set.seed(42)

# Shuffle row indices: rows
```

```r
rows <- sample(nrow(diamonds))

# Randomly order data
shuffled_diamonds <- diamonds[rows, ]
```

80/20 split

```r
# Determine row to split on: split
split <- round(nrow(diamonds) * 0.80)

# Create train
train <- diamonds[1:split,]

# Create test
test <- diamonds[(split + 1):nrow(diamonds), ]
```

Predict on test set

```r
# Fit lm model on train: model
model <- lm(price ~ ., train)

# Predict on test: p
p <- predict(model,test)
```

Calculate test set RMSE

```r
# Compute errors: error
error <- p-test$price

# Calculate RMSE
print(sqrt(mean(error^2)))
```

```
## [1] 796.8922
```

Cross-validation

10-fold cross-validation

```r
# Fit lm model using 10-fold CV: model
model <- train(
  price ~.,
  diamonds,
  method = "lm",
  trControl = trainControl(
    method = "cv",
    number = 10,
    verboseIter = FALSE
  )
)
```

5 x 5-fold cross-validation

```r
# Fit lm model using 5 x 5-fold CV: model
model <- train(
  price ~.,
  diamonds,
  method = "lm",
  trControl = trainControl(
    method = "repeatedcv",
    number = 5,
    repeats = 5,
    verboseIter = FALSE
  )
)
```

train/test split

60/40 split on Sonar dataset

```r
library(mlbench)
data(Sonar)
# Get the number of observations
n_obs <- nrow(Sonar)

# Shuffle row indices: permuted_rows
permuted_rows <- sample(n_obs)

# Randomly order data: Sonar
Sonar_shuffled <- Sonar[permuted_rows, ]

# Identify row to split on: split
split <- round(nrow(Sonar_shuffled) * 0.60)

# Create train
train <- Sonar_shuffled[1:split,]

# Create test
test <- Sonar_shuffled[(split + 1):nrow(Sonar_shuffled), ]
```

model fit

```r
model<-glm(Class~.,family="binomial", train)
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
# Predict on test: p
p<-predict(model,test,type="response")
```

Confusion matrix

```r
# If p exceeds threshold of 0.5, M else R: m_or_r
m_or_r <- ifelse(p > 0.5, "M", "R")

# Convert to factor: p_class
p_class <- factor(m_or_r, levels = levels(test[["Class"]]))

# Create confusion matrix
confusionMatrix(p_class, test[["Class"]])
```

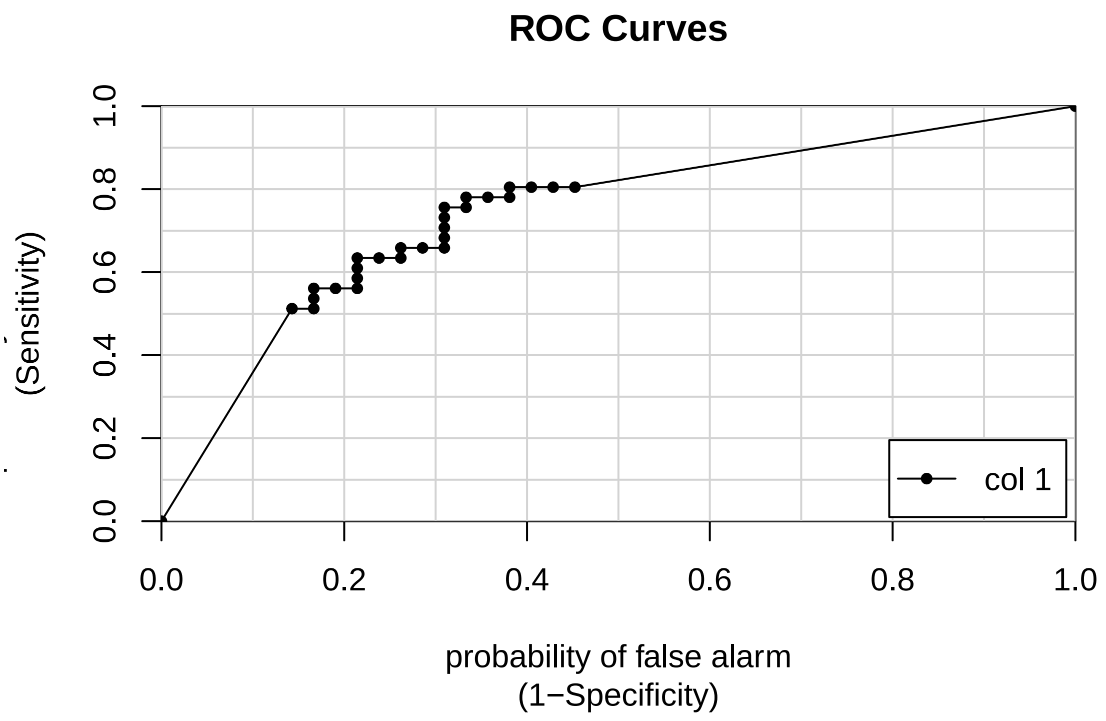
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  M  R
##          M 13 27
##          R 29 14
##
##                Accuracy : 0.3253
##                  95% CI : (0.2265, 0.437)
##     No Information Rate : 0.506
##     P-Value [Acc > NIR] : 0.9997
##
##                   Kappa : -0.3488
##
##  Mcnemar's Test P-Value : 0.8937
##
##             Sensitivity : 0.3095
##             Specificity : 0.3415
##          Pos Pred Value : 0.3250
##          Neg Pred Value : 0.3256
##              Prevalence : 0.5060
##          Detection Rate : 0.1566
##    Detection Prevalence : 0.4819
##       Balanced Accuracy : 0.3255
##
##        'Positive' Class : M
##
```

Class probabilities and predictions - Evaluating classification thresholds

ROC curve

```r
library(caTools)
# Predict on test: p
p<-predict(model,test,type="response")

# Make ROC curve
colAUC(p, test[["Class"]], plotROC = TRUE)
```



```
##                [,1]
## M vs. R 0.7439024
```

Area under the curve (AUC)

Customizing and using trainControl

```r
# Create trainControl object: myControl
myControl <- trainControl(
  method = "cv",
  number = 10,
  summaryFunction = twoClassSummary,
  classProbs = TRUE, # IMPORTANT!
  verboseIter = FALSE
)
# Train glm with custom trainControl: model
model<-train(method="glm",data=Sonar,Class~.,trControl=myControl)

# Print model to console
print(model)
```

Random forest model

fitting RF

```r
# obtain the dataset
url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv"
wine <- read.csv(url, header = TRUE, sep = ";")
# Fit random forest: model
model <- train(
  quality~.,
  tuneLength = 1,
  data = wine,
  method = "ranger",
  trControl = trainControl(
    method = "cv",
    number = 5,
    verboseIter = FALSE
  )
)
```

Hyperparameter tuning

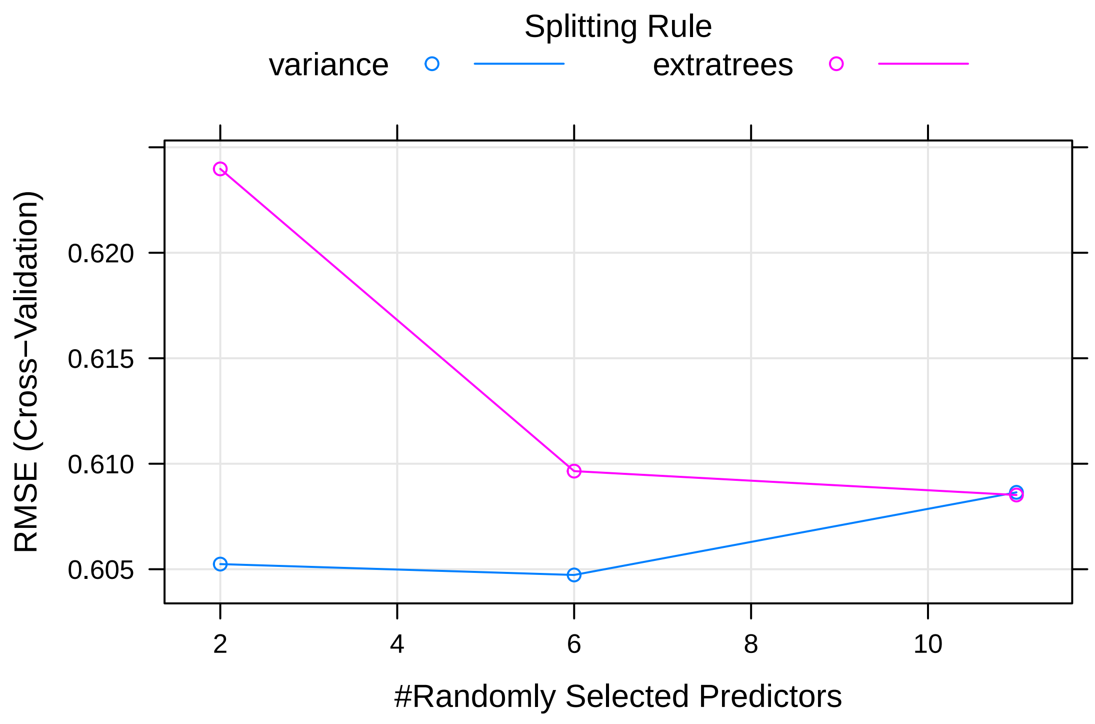
Try a longer tune length

```r
# Fit random forest: model
model <- train(
  quality~.,
  tuneLength = 3,
  data = wine,
  method = "ranger",
  trControl = trainControl(
    method = "cv",
    number = 5,
    verboseIter = FALSE
  )
)

# Print model to console
print(model)
```

```
## Random Forest
##
## 4898 samples
##   11 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 3918, 3919, 3918, 3919, 3918
## Resampling results across tuning parameters:
##
##   mtry  splitrule   RMSE       Rsquared   MAE
##    2    variance    0.6052432  0.5442810  0.4397588
##    2    extratrees  0.6239763  0.5281805  0.4661522
##    6    variance    0.6047288  0.5385476  0.4347356
##    6    extratrees  0.6096502  0.5377340  0.4472939
##   11    variance    0.6086439  0.5303983  0.4368120
##   11    extratrees  0.6085183  0.5354760  0.4427934
##
## Tuning parameter 'min.node.size' was held constant at a value of 5
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were mtry = 6, splitrule = variance
##  and min.node.size = 5.
```

```r
# Plot model
plot(model)
```



Custom tuning using tuneGrid

```r
tuneGrid <- data.frame(
  .mtry = c(2, 3, 7),
  .splitrule = "variance",
  .min.node.size = 5
)
```

Print maximum ROC statistic

```r
max(model[["results"]][["ROC"]])
```

```
## Warning in max(model[["results"]][["ROC"]]): no non-missing arguments to max;
## returning -Inf

## [1] -Inf
```

glmnet with custom trainControl and tuning

```r
# Train glmnet with custom trainControl and tuning: model
model <- train(
  y~.,
  data = overfit,
  tuneGrid = expand.grid(
    alpha = 0:1,
    lambda = seq(0.0001,1,length=20)
  ),
  method = "glmnet",
  trControl = myControl
)
```

Handling missing values

Median inputation

```r
library(OneR)
data(breastcancer)
breast_cancer_y <- breastcancer$Class
breast_cancer_x <- breastcancer[,-10]
str(breast_cancer_y)
model <- train(
  x = breast_cancer_x, y = breast_cancer_y,
  method = "glm",
  trControl = myControl,
  preProcess = "medianImpute"
)
```

KNN inputation

```r
# Apply KNN imputation: knn_model
library(RANN)
knn_model <- train(
  x = breast_cancer_x,
  y = breast_cancer_y,
  method = "glm",
  trControl = myControl,
  preProcess = "knnImpute"
)
```

Other preprocessing steps

Combining preprocessing methods

```r
# Update model with standardization
model <- train(
  x = breast_cancer_x,
  y = breast_cancer_y,
  method = "glm",
  trControl = myControl,
  preProcess = c("medianImpute", "center", "scale")
)
```

Handling low-information predictors

Remove near zero variance predictors

```r
url <- "https://assets.datacamp.com/production/course_1048/datasets/BloodBrain.RData"
download.file(url, "./BloodBrain.RData")
load("./BloodBrain.RData")
# Identify near zero variance predictors: remove_cols
remove_cols <- nearZeroVar(bloodbrain_x, names = TRUE,
                           freqCut = 2, uniqueCut = 20)

# Get all column names from bloodbrain_x: all_cols
all_cols <- names(bloodbrain_x)

# Remove from data: bloodbrain_x_small
bloodbrain_x_small <- bloodbrain_x[ , setdiff(all_cols, remove_cols)]
print(model)
```

```
## Random Forest
##
## 4898 samples
##   11 predictor
```

```
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 3918, 3919, 3918, 3919, 3918
## Resampling results across tuning parameters:
##
##   mtry  splitrule   RMSE       Rsquared   MAE
##    2    variance    0.6052432  0.5442810  0.4397588
##    2    extratrees  0.6239763  0.5281805  0.4661522
##    6    variance    0.6047288  0.5385476  0.4347356
##    6    extratrees  0.6096502  0.5377340  0.4472939
##   11    variance    0.6086439  0.5303983  0.4368120
##   11    extratrees  0.6085183  0.5354760  0.4427934
##
## Tuning parameter 'min.node.size' was held constant at a value of 5
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were mtry = 6, splitrule = variance
##  and min.node.size = 5.
```

Principle components analysis (PCA)

```r
# Fit glm model using PCA: model
model <- train(
  x = bloodbrain_x,
  y = bloodbrain_y,
  method = "glm",
  preProcess = "pca"
)
print(model)
```

```
## Generalized Linear Model
##
## 208 samples
## 132 predictors
##
## Pre-processing: principal component signal extraction (132), centered
##  (132), scaled (132)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 208, 208, 208, 208, 208, 208, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.5896973  0.4358796  0.4468568
```

Applying trainControl

Custom train/test split

```r
url <- "https://assets.datacamp.com/production/course_1048/datasets/Churn.RData"
download.file(url, "./Churn.RData")
load("./Churn.RData")
# Create custom indices: myFolds
myFolds <- createFolds(churn_y, k = 5)

# Create reusable trainControl object: myControl
myControl <- trainControl(
  summaryFunction = twoClassSummary,
  classProbs = TRUE, # IMPORTANT!
  verboseIter = FALSE,
  savePredictions = TRUE,
  index = myFolds
)
```

glmnet and rf models

```r
model_glmnet <- train(
x = churn_x, y = churn_y,
metric = "ROC",
method = "glmnet",
trControl = myControl
)

model_rf <- train(
x = churn_x, y = churn_y,
metric = "ROC",
method = "ranger",
trControl = myControl
)
```

Comparing models

```r
# Create model_list
model_list <- list(item1 = model_glmnet, item2 = model_rf)

# Pass model_list to resamples(): resamples
resamples <- resamples(model_list)

# Summarize the results
summary(resamples)
```

box-and-whisker plot

```r
# Create bwplot
bwplot(resamples,metric = "ROC")
```

Scatterplot

```r
# Create xyplot
xyplot(resamples,metric = "ROC")
```

Ensembling models

```r
library(caretEnsemble)
# Create ensemble model: stack
stack <- caretStack(all.models = model_list, method = "glm")
# Look at summary
summary(stack)
```