How can trainControl be used to control training parameters in R?

Authored by

stats writer

June 23, 2024

TrainControl is a function in R that allows for the control of training parameters in machine learning models. It provides a flexible and efficient way to specify various settings for the training process, such as cross-validation, resampling methods, and model performance metrics. By using trainControl, users can easily customize their training process to achieve optimal results for their specific data set and analytical goals. This function is a valuable tool for data scientists and researchers seeking to improve the accuracy and efficiency of their machine learning models in R.

R: Use trainControl to Control Training Parameters

To evaluate how well a model is able to fit a dataset, we must analyze how the model performs on observations it has never seen before.

One of the most common ways to do this is by using k-fold cross-validation, which uses the following approach:

1. Randomly divide a dataset into *k* groups, or "folds", of roughly equal size.

2. Choose one of the folds to be the holdout set. Fit the model on the remaining k-1 folds. Calculate the test MSE on the observations in the fold that was held out.

3. Repeat this process *k* times, using a different set each time as the holdout set.

4. Calculate the overall test MSE to be the average of

the *k* test MSE's.

The easiest way to perform k-fold cross-validation in R is by using the trainControl() and train() functions from the caret library in R.

The trainControl() function is used to specify the parameters for training (e.g. the type of cross-validation to use, the number of folds to use, etc.) and the train() function is used to actually fit the model to the data.

The following example shows how to use the trainControl() and train() functions in practice.

Example: How to Use trainControl() in R

Suppose we have the following dataset in R:

```
#create data frame
df <- data.frame(y=c(6, 8, 12, 14, 14, 15, 17, 22, 24, 23),
x1=c(2, 5, 4, 3, 4, 6, 7, 5, 8, 9),
x2=c(14, 12, 12, 13, 7, 8, 7, 4, 6, 5))

#view data frame
df

y x1 x2
```

6 2 14

8 5 12

12 4 12

14 3 13

14 4 7

15 6 8

17 7 7

22 5 4

24 8 6

23 9 5

Now suppose we use the function to fit a to this dataset, using x1 and x2 as the predictor variables and y as the response variable:

```
#fit multiple linear regression model to data
fit <- lm(y ~ x1 + x2, data=df)

#view model summary
summary(fit)
```

Call:

lm(formula = y ~ x1 + x2, data = df)

Residuals:

Min 1Q Median 3Q Max
-3.6650 -1.9228 -0.3684 1.2783 5.0208

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 21.2672 6.9927 3.041 0.0188 *
x1 0.7803 0.6942 1.124 0.2981
x2 -1.1253 0.4251 -2.647 0.0331 *

Signif. codes: 0 '*' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1**

Residual standard error: 3.093 on 7 degrees of freedom
Multiple R-squared: 0.801, Adjusted R-squared: 0.7441
F-statistic: 14.09 on 2 and 7 DF, p-value: 0.003516

Using the coefficients in the model output, we can write the fitted regression model:

y = 21.2672 + 0.7803*(x1) - 1.1253(x2)

To get an idea of how well this model would perform on unseen , we can use k-fold cross validation.

We then pass this trainControl() function to the train() function to actually perform the k-fold cross validation:

```
library(caret)

#specify the cross-validation method
ctrl <- trainControl(method = "cv", number = 5)

#fit a regression model and use k-fold CV to evaluate performance
model <- train(y ~ x1 + x2, data = df, method = "lm", trControl = ctrl)

#view summary of k-fold CV
print(model)

Linear Regression

10 samples
2 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 8, 8, 8, 8, 8
Resampling results:

RMSE Rsquared MAE
3.612302 1 3.232153
```

Tuning parameter 'intercept' was held constant at a value of TRUE

From the output we can see that the model was fit 5 times using a sample size of 8 observations each time.

Each time the model was then used to predict the values of the 2 observations that were held out and the following metrics were calculated each time:

RMSE: The root mean squared error. This measures the average difference between the predictions made by the model and the actual observations. The lower the RMSE, the more closely a model can predict the actual observations.MAE: The mean absolute error. This is the average absolute difference between the predictions made by the model and the actual observations. The lower the MAE, the more closely a model can predict the actual observations.

The average of the RMSE and MAE values for the five folds is shown in the output:

RMSE: 3.612302MAE: 3.232153

These metrics give us an idea of how well the model performs on previously unseen data.

In practice, we typically fit several different models and compare these metrics to determine which model performs best on unseen data.

For example, we might proceed to fit a and perform K-fold cross validation on it to see how the RMSE and MAE metrics compare to the multiple linear regression model.

Note #1: In this example we chose to use k=5 folds, but you can choose however many folds you'd like. In practice, we typically choose between 5 and 10 folds because this turns out to be the optimal number of folds that produce reliable test error rates.

Note #2: The trainControl() function accepts many potential arguments. You can find the complete documentation for this function .

The following tutorials provide additional information about model training: