How to Evaluate Your R Model with Cross Validation

Authored by

stats writer

December 30, 2025

Cross validation is a fundamental statistical technique essential for rigorously evaluating the predictive performance of a statistical or machine learning model, particularly within the R environment. This methodology operates by partitioning the dataset used to construct the model into separate subsets: a training set, utilized for model fitting, and a testing set, reserved for unbiased performance assessment. By iterating this split-train-evaluate procedure multiple times, we generate robust estimates of the model's generalization ability to new, unseen data, effectively minimizing the risk of overfitting.

In the field of statistics and machine learning, modeling serves two primary objectives:

To establish and quantify the relationship between one or more predictor variables and a corresponding response variable.
To leverage the fitted model for accurate forecasting or classification of future, unobserved observations.

While traditional methods focus on inference, **Cross validation** is explicitly designed to quantify a model's robustness and accuracy when predicting new data points.

Consider a scenario involving a multiple linear regression model built to assess loan risk. This model might utilize demographic factors like *age* and *income* (the predictor variables) to forecast *loan default status* (the response variable). The ultimate goal is to apply this trained model to new loan applicants, providing a probabilistic prediction of their default risk based on their supplied attributes.

Assessing true predictive strength requires the model to operate on data outside of its training regimen. By testing the model on novel data, we can accurately estimate the crucial metric of prediction error, which measures how well the model generalizes beyond the sample data.

Using Cross Validation to Estimate Prediction Error

While there are multiple variants, the core methodology of **cross-validation** remains consistent. The objective is to cyclically partition data to ensure the performance evaluation is independent of the training process.

Data Partitioning: A specific subset of observations--typically 15% to 25% of the total dataset--is randomly set aside to serve as the temporary hold-out or testing data.
Model Training: The statistical model is fitted, or "trained," exclusively using the remaining majority of the data (the training set).
Performance Evaluation: The trained model is then applied to make predictions on the set-aside observations, allowing for an accurate assessment of its predictive capacity on novel data.

Measuring the Quality of a Model

To quantify the predictive quality of a model when tested against new observations, several key metrics are employed. These indicators help practitioners choose the optimal model architecture by minimizing error or maximizing explained variance.

Multiple R-squared: This metric quantifies the proportion of the variance in the response variable that is predictable from the predictor variables. A value of 1 signifies a perfect linear relationship where the model explains all variability, while a value of 0 suggests no linear relationship exists. Generally, higher R-squared values indicate a superior fit and greater predictive power.

Root Mean Squared Error (RMSE): RMSE represents the average magnitude of the prediction error in the same units as the response variable. It calculates the average distance between the observed true values and the values predicted by the model. Because errors are squared before averaging, RMSE penalizes large errors heavily, making lower values indicative of a more accurate model fit.

Mean Absolute Error (MAE): MAE measures the average of the absolute differences between the actual observations and the predicted values. Unlike RMSE, MAE uses absolute values, making it less sensitive to extreme outliers. As with RMSE, a lower MAE score corresponds to a better model fit.

Implementing Four Different Cross-Validation Techniques in R

We will now demonstrate the practical implementation of four distinct **cross validation** strategies within the R environment. Understanding these variations is crucial for selecting the most appropriate evaluation method for a given modeling task.

The **Validation Set Approach**
The **k-fold Cross Validation**
The **Leave One Out Cross Validation (LOOCV)**
The **Repeated k-fold Cross Validation**

To provide concrete examples, we will utilize a specific subset of the widely recognized, built-in R dataset, *mtcars*. This data allows us to focus on predicting fuel efficiency (miles per gallon) based on engine characteristics.

```
#define dataset
data <- mtcars

#view first six rows of new data
head(data)
```

```
# mpg disp hp drat
#Mazda RX4 21.0 160 110 3.90
#Mazda RX4 Wag 21.0 160 110 3.90
#Datsun 710 22.8 108 93 3.85
#Hornet 4 Drive 21.4 258 110 3.08
#Hornet Sportabout 18.7 360 175 3.15
#Valiant 18.1 225 105 2.76
```

Our objective is to construct a **multiple linear regression** model using displacement (*disp*), horsepower (*hp*), and rear axle ratio (*drat*) as the predictor variables, with miles per gallon (*mpg*) serving as the response variable.

The Validation Set Approach

The **validation set approach** is the simplest form of cross-validation, requiring only a single, explicit split of the data. This technique relies on randomly dividing the available observations into two distinct, non-overlapping subsets: a training set and a test (or validation) set.

The data is permanently split into two major portions. Typically, 70%-80% of the data is allocated for training, leaving the remaining 20%-30% for testing.
The model parameters are estimated solely using the designated training data set.
The trained model is then used to generate predictions exclusively on the observations contained within the hold-out test set.
Model quality is assessed using performance metrics such as R-squared, RMSE, and MAE, providing a single estimate of the generalization error.

R Implementation Example

The following R example demonstrates the implementation of the validation set approach. We use the previously defined *mtcars* subset, splitting it 80/20 into training and testing partitions, respectively. After fitting the linear model, we assess its performance metrics on the unseen test data.

```
#load dplyr library used for data manipulation
library(dplyr)

#load caret library used for partitioning data into training and test set
library(caret)

#make this example reproducible
set.seed(0)
```

```
#define the dataset
data <- mtcars

#split the dataset into a training set (80%) and test set (20%).
training_obs <- data$mpg %>% createDataPartition(p = 0.8, list = FALSE)

train <- data
test <- data

# Build the linear regression model on the training set
model <- lm(mpg ~ ., data = train)

# Use the model to make predictions on the test set
predictions <- model %>% predict(test)

#Examine R-squared, RMSE, and MAE of predictions
data.frame(R_squared = R2(predictions, test$mpg),
RMSE = RMSE(predictions, test$mpg),
MAE = MAE(predictions, test$mpg))

# R_squared RMSE MAE
#1 0.9213066 1.876038 1.66614
```

In practical model selection, the model that yields the lowest RMSE value on the independent test set is generally considered the superior choice for prediction.

Pros and Cons of this Approach

The primary advantage of the validation set approach is its operational simplicity and high computational efficiency, requiring the model to be trained only once. However, a significant drawback is the potential for bias: since the model is built on only a fraction of the total data, if the hold-out test set happens to contain critical, unique information, the final model's performance estimate may be unreliable or underestimate the true prediction error.

k-fold Cross Validation Approach

The **k-fold cross validation approach** addresses the data usage limitations inherent in the simple validation set method. Instead of a single split, k-fold CV systematically rotates through all data points, ensuring every observation is used exactly once in the validation process. The process is defined by the parameter *k*, which specifies the number of partitions (folds) the data is divided into.

The entire dataset is randomly partitioned into **k** equal-sized subsets, known as folds (e.g., k=5 or

k=10).
The model is trained on the data derived from **k-1** folds, combining the vast majority of the data.
The model is then tested exclusively on the one remaining, untouched fold (the validation set), and the prediction error is recorded.
This training and testing cycle is repeated **k** times, ensuring that each of the k folds serves as the validation set exactly once.
The final measure of model quality, or the **cross-validation error**, is the aggregated average of the k individual test error estimates.

R Implementation Example (k=5)

In this demonstration, we perform 5-fold cross validation. The *caret* package handles the resampling setup, iteratively training the model on 4 folds and testing on the remaining 1. The output summarizes the averaged metrics (R-squared, RMSE, and MAE) across the five repetitions.

#load *dplyr* library used for data manipulation
library(dplyr)

#load *caret* library used for partitioning data into training and test set
library(caret)

#make this example reproducible
set.seed(0)

#define the dataset
data <- mtcars

#define the number of subsets (or "folds") to use
train_control <- trainControl(method = "cv", number = 5)

#train the model
model <- train(mpg ~ ., data = data, method = "lm", trControl = train_control)

#Summarize the results
print(model)

#Linear Regression
#
#32 samples
3 predictor
#
#No pre-processing

```
#Resampling: Cross-Validated (5 fold)
#Summary of sample sizes: 26, 25, 26, 25, 26
#Resampling results:
#
# RMSE Rsquared MAE
# 3.095501 0.7661981 2.467427
#
#Tuning parameter 'intercept' was held constant at a value of TRUE
```

Pros and Cons of this Approach

The key advantage of k-fold cross validation over the validation set method is the comprehensive use of the data; since the model is built multiple times using different data combinations, the risk of omitting vital information is drastically reduced, leading to a more reliable estimate of model performance.

The primary decision point in k-fold CV is selecting the optimal value for *k*. A low *k* (e.g., k=3) results in higher bias but lower variance in the error estimate, as the training sets are smaller than the full dataset. Conversely, a high *k* (e.g., k=100) reduces bias (as training sets are larger) but increases the variance of the estimate. In common practice, choosing **k=5** or **k=10** is generally recommended, as these values offer a balanced trade-off between bias and variance while maintaining reasonable computational speed.

Leave One Out Cross Validation (LOOCV) Approach

The **Leave One Out Cross Validation (LOOCV) approach** is an extreme case of k-fold CV where the number of folds, *k*, is set equal to the total number of observations, *N*. In this method, the model is trained on almost the entire dataset, maximizing the training sample size in each iteration.

A model is constructed using all available data points except for a single observation (N-1 data points).
This trained model is used to predict the value of the single, excluded observation. The error for this specific prediction is recorded.
This entire sequence (train on N-1, test on 1) is repeated N times, once for every observation in the dataset.
The overall performance is determined by calculating the average of all N recorded prediction errors.

R Implementation Example

The R code below demonstrates LOOCV using the same *mtcars* subset. Since there are 32 observations in our data, the training process will fit 32 separate models, each leaving one unique observation out for testing.

#load *dplyr* library used for data manipulation
library(dplyr)

#load *caret* library used for partitioning data into training and test set
library(caret)

#make this example reproducible
set.seed(0)

#define the dataset
data <- mtcars

#specify that we want to use LOOCV
train_control <- trainControl(method = "LOOCV")

#train the model
model <- train(mpg ~ ., data = data, method = "lm", trControl = train_control)

#summarize the results
print(model)

#Linear Regression
#
#32 samples
3 predictor
#
#No pre-processing
#Resampling: Leave-One-Out Cross-Validation
#Summary of sample sizes: 31, 31, 31, 31, 31, 31, ...
#Resampling results:
#
RMSE Rsquared MAE
3.168763 0.7170704 2.503544
#
#Tuning parameter 'intercept' was held constant at a value of TRUE

Pros and Cons of this Approach

LOOCV offers the distinct benefit of maximum data utilization, as the training set size (N-1) closely approximates the full dataset size, which generally leads to a low-bias estimate of the prediction error. However, this method suffers from two major disadvantages: first, since the training sets are nearly identical across iterations, the error estimates tend to be highly correlated, leading to high variability in the final error estimate. Second, the requirement to fit N models makes LOOCV significantly more computationally demanding and cumbersome, especially for large datasets.

Repeated k-fold Cross Validation Approach

The **repeated k-fold cross validation** method is a refinement of the standard k-fold technique designed to provide a more stable and robust estimate of model performance. It achieves this by performing the k-fold process multiple times (e.g., 4 or 10 repeats), with the data being randomly reshuffled before each repetition begins.

The final aggregated measure of performance is calculated as the mean error across all repeats (R repetitions * k folds). This iterative shuffling and re-evaluation smooths out any potential bias introduced by a single random split in the standard k-fold process, thereby yielding a highly reliable assessment of generalization capability.

The subsequent example demonstrates a setup where 5-fold cross validation is executed, and this entire procedure is repeated 4 times.

```
#load dplyr library used for data manipulation
library(dplyr)

#load caret library used for partitioning data into training and test set
library(caret)

#make this example reproducible
set.seed(0)

#define the dataset
data <- mtcars

#define the number of subsets to use and number of times to repeat k-fold CV
train_control <- trainControl(method = "repeatedcv", number = 5, repeats = 4)

#train the model
model <- train(mpg ~ ., data = data, method = "lm", trControl = train_control)
```

```
#summarize the results
print(model)

#Linear Regression
#
#32 samples
# 3 predictor
#
#No pre-processing
#Resampling: Cross-Validated (5 fold, repeated 4 times)
#Summary of sample sizes: 26, 25, 26, 25, 26, 25, ...
#Resampling results:
#
# RMSE Rsquared MAE
# 3.176339 0.7909337 2.559131
#
#Tuning parameter 'intercept' was held constant at a value of TRUE
```

Pros and Cons of this Approach

The chief benefit of repeated k-fold cross validation is that the repeated shufflings and subsequent model fitting provide an even more accurate and unbiased estimate of the true prediction error compared to a single run. The drawback, naturally, is the increased computational burden: since the process involves R repeats of k-fold CV, the total number of models fitted is R * k, making it significantly slower than standard k-fold CV or the validation set approach.

Balancing Bias and Variance: Choosing the Number of Folds (k)

The most critical and subjective aspect of performing k-fold cross validation is determining the optimal number of folds, *k*. This choice directly influences the trade-off between bias and variance in the resulting error estimate. Generally, a smaller number of folds (low *k*, e.g., k=3) results in smaller training sets, leading to a higher bias in the estimate but lower variance. Conversely, a larger number of folds (high *k*, approaching LOOCV) reduces bias because the training sets are larger, but it increases the variability of the estimate due to the high correlation between the resulting models.

Computational cost is another practical constraint. Since a new model must be trained for every fold, selecting a high *k* significantly increases the time required for evaluation, which can be prohibitive for complex models or very large datasets.

Consequently, in standard practice, **k=5** or **k=10** folds are most commonly used. This range is widely accepted because it successfully strikes a balance, offering a reasonable compromise between low bias, manageable variance, and acceptable computational efficiency.

How to Choose and Finalize a Model After Cross Validation

The primary utility of cross validation is its ability to objectively assess the predictive performance of candidate models. By evaluating multiple models (e.g., a simple linear model versus a complex polynomial model) using consistent metrics like RMSE or MAE, we can confidently identify the architecture that exhibits the lowest true prediction error.

It is crucial to understand that the model instances trained during the cross-validation cycles are purely for evaluation purposes. Once cross validation identifies the optimal model structure, the final, production-ready model must be refitted using **all** of the available data.

For instance, if 5-fold cross validation confirms that Model A is superior to Model B, we discard the five instances of Model A created during CV. We then utilize **100%** of the dataset to train the final, robust version of Model A, ensuring the ultimate model leverages the maximum possible information for deployment.