Learning Logistic Regression with R: A Step-by-Step Guide

Authored by

Mohammed looti

November 6, 2025

Logistic regression is a fundamental and widely used statistical technique, especially prevalent in fields like machine learning, finance, and epidemiology. Its primary purpose is to model the probability of a discrete outcome, making it distinct from linear regression, which predicts continuous variables. Specifically, logistic regression excels when the outcome, or **response variable**, is categorical and most commonly **binary** (e.g., success/failure, 0/1, default/non-default). This model estimates the likelihood that an observation belongs to a specific category based on a set of predictor variables.

The mathematical foundation of this approach relies on transforming the linear combination of predictor variables into a probability bounded between 0 and 1. This transformation is achieved using the logit function (the natural logarithm of the odds), which ensures that the predictions remain within the valid probability range. The coefficients derived from the model are typically estimated using the principle of **maximum likelihood estimation** (MLE), which finds the coefficient values that maximize the likelihood of observing the actual data.

The core objective of the model is to link the predictor variables to the **log odds** of the event occurring. The structural equation fitted by logistic regression takes the following form, representing the linear relationship on the logit scale:

log = β0 + β1X1 + β2X2 + ... + βpXp

In this equation, the right-hand side is a standard linear predictor. The left-hand side, log, represents the log odds of the event (p(X)) occurring versus the event not occurring (1-p(X)).

Xj: Represents the jth **predictor variable** used in the model.

βj: Represents the coefficient estimate corresponding to the jth predictor variable, quantifying its impact on the log odds of the outcome.

To convert the predicted log odds back into an interpretable probability (p(X)), we apply the inverse of the logit function (the logistic function). This resulting probability calculation allows us to estimate the chance that a given observation will result in the value of 1:

p(X) = eβ0 + β1X1 + β2X2 + ... + βpXp / (1 + eβ0 + β1X1 + β2X2 + ... + βpXp)

After calculating the probability, classification requires a predefined **probability threshold** (often set at 0.5). Any observation whose calculated probability meets or exceeds this threshold is classified as the positive outcome ("1"), while those falling below it are classified as the negative outcome ("0"). This tutorial provides a meticulous, step-by-step guide on how to implement and rigorously interpret a logistic regression model using the R programming language.

Step 1: Preparing the Environment and Loading the Dataset

To demonstrate the practical application of logistic regression, we will use the well-known **Default** dataset, which is conveniently housed within the ISLR package in R. This dataset is commonly used for predictive modeling exercises. Our goal is to develop a robust model that utilizes customer financial and demographic data to predict the likelihood of financial default. The process begins with setting up the R environment, loading the necessary data, and performing an initial exploratory review to fully grasp the structure and underlying distributions of the variables.

The following R code snippet executes the necessary steps: loading the data into a variable named `data` and generating crucial summary statistics. This initial summary output provides valuable insights into the data's characteristics, including the range and distribution of numerical variables (like income and balance) and the frequency counts for categorical variables (like default status and student status).

```
#load dataset
data <- ISLR::Default

#view summary of dataset
summary(data)

 default    student       balance        income
 No :9667   No :7056   Min.   :   0.0   Min.   :  772
 Yes: 333   Yes:2944   1st Qu.: 481.7   1st Qu.:21340
                       Median : 823.6   Median :34553
                       Mean   : 835.4   Mean   :33517
                       3rd Qu.:1166.3   3rd Qu.:43808
                       Max.   :2654.3   Max.   :73554

#find total observations in dataset
nrow(data)

10000
```

The dataset comprises exactly 10,000 observations. It is immediately clear that the outcome variable, `default`, is highly imbalanced, with 9,667 non-defaults and only 333 defaults. This imbalance is a critical consideration for model evaluation later on. Each observation is characterized by four principal variables:

default: The binary response variable indicating whether the individual defaulted (Yes/No). This is our target variable.

student: A categorical predictor indicating student status (Yes/No).

balance: The average monthly credit card balance carried by the individual (a numerical predictor).

income: The annual income of the individual (a numerical predictor).

Our primary analytical objective is to construct a predictive logistic regression model using `student` status, credit card `balance`, and `income` as predictors to accurately estimate the probability that a specific individual will experience a financial default.

Step 2: Partitioning Data into Training and Test Sets

Before any statistical modeling commences, standard practice dictates splitting the available data into two mutually exclusive subsets: the training set and the testing set. This critical step ensures that the model's performance can be assessed on data it has never encountered, thereby providing an unbiased estimate of its generalization ability. The **training set** is used exclusively for estimating the model's coefficients (the β values), while the independent **testing set** is reserved solely for evaluating prediction accuracy and overall model diagnostics.

For this demonstration, we adopt a common allocation ratio, designating 70% of the observations to the training set for model development and reserving the remaining 30% for validation and performance assessment. Crucially, we utilize the `set.seed()` function to fix the random number generator's state. This practice guarantees that the random sampling process is perfectly reproducible, meaning that anyone running the exact same code, or running it again in the future, will obtain the identical data split, which is essential for transparent research.

The R code below performs the required stratified sampling, creating the `train` and `test` data frames based on the 70/30 split. The logical vector `sample` determines which observations are allocated to the training set using a probability weighting function.

#make this example reproducible
set.seed(1)

#Use 70% of dataset as training set and remaining 30% as testing set
sample <- sample(c(TRUE, FALSE), nrow(data), replace=TRUE, prob=c(0.7,0.3))
train <- data
test <- data

By carefully separating the data, we mitigate the risk of **overfitting**--a situation where the model performs excellently on the training data but fails dramatically when applied to new, unseen data. The testing set serves as the final, objective checkpoint for model validity.

Step 3: Fitting the Logistic Regression Model in R

The fitting of a logistic regression model in R is accomplished using the `glm()` function, which stands for Generalized Linear Model. To ensure that R uses the appropriate mathematical framework--the logit link function--for our binary outcome, we must explicitly specify the argument `family="binomial"`. Our full model formula includes all three available predictors from the dataset: `student`, `balance`, and `income`, aiming to predict the `default` status.

Prior to viewing the statistical summary, it is often useful to temporarily disable scientific notation in R (using `options(scipen=999)`). This minor modification ensures that the coefficient estimates, which are often very small numerical values in logistic regression, are displayed in a clear, easily readable decimal format, facilitating direct interpretation.

#fit logistic regression model
model <- glm(default~student+balance+income, family="binomial", data=train)

#disable scientific notation for model summary
options(scipen=999)

#view model summary
summary(model)

Call:
glm(formula = default ~ student + balance + income, family = "binomial",
data = train)

Deviance Residuals:
Min 1Q Median 3Q Max
-2.5586 -0.1353 -0.0519 -0.0177 3.7973

Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) -11.478101194 0.623409555 -18.412 <0.0000000000000002 ***
studentYes -0.493292438 0.285735949 -1.726 0.0843 .
balance 0.005988059 0.000293765 20.384 <0.0000000000000002 ***
income 0.000007857 0.000009965 0.788 0.4304

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

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 2021.1 on 6963 degrees of freedom

Residual deviance: 1065.4 on 6960 degrees of freedom
AIC: 1073.4

Number of Fisher Scoring iterations: 8

Interpreting the **Coefficients** table is crucial. The Estimate column provides the change in the log odds of the outcome (default=Yes) associated with a one-unit increase in the predictor variable, assuming all other variables are held constant. For instance, the positive coefficient for `balance` (approximately 0.006) indicates that a one-dollar increase in a customer's credit card balance is linked to an increase of 0.006 in the log odds of defaulting. Conversely, the negative coefficient for `studentYes` suggests that, holding balance and income constant, being a student slightly decreases the log odds of default compared to a non-student baseline, though this effect needs further significance testing.

The **p-values** (listed under Pr(>|z|)) are used to determine the statistical significance of each predictor. Very low p-values (typically below 0.05) indicate that the predictor significantly influences the log odds of default. Based on the model summary: `balance` is highly significant (p-value < 0.0000), suggesting it is a crucial determinant of default probability. The `student` status is marginally significant (p-value = 0.0843), while `income` is clearly not statistically significant (p-value = 0.4304) within this specific model structure. Consequently, we can infer that the credit card balance is the dominant factor driving the predicted likelihood of default.

Step 4: Assessing Model Fit and Multicollinearity

Unlike linear regression, where R-squared provides a clear measure of variance explained, logistic models require alternative metrics to assess overall fit. We utilize **Pseudo R^2** statistics for this purpose. A widely accepted and interpretable metric is McFadden's R^2. This statistic ranges from 0 to a maximum value just under 1, and values typically exceeding 0.20 or 0.40 are often cited as indicative of an excellent model fit, especially in social science applications.

We calculate McFadden's R^2 using the `pR2` function from the `pscl` package, applying it directly to our fitted model object:

pscl::pR2(model)

McFadden
0.4728807

The calculated value of **0.4728807** is quite high, providing strong evidence that our logistic regression model captures a substantial amount of the variability in the default outcome. This

suggests the model has strong predictive capabilities when applied to the training data.

Furthermore, we quantify the relative contribution of each predictor using the `varImp` function provided by the `caret` package. This metric provides a standardized measure of variable importance, where higher values denote a greater influence on the predicted outcome. The results below confirm our initial interpretation of the p-values from the GLM summary:

caret::varImp(model)

Overall
studentYes 1.726393
balance 20.383812
income 0.788449

The overwhelming difference in the 'Overall' score clearly establishes `balance` as the most dominant factor in predicting default, followed distantly by `student` status. `income` shows negligible importance, consistent with its non-significant p-value.

Finally, a critical step in model validation is checking for **multicollinearity**--a condition where predictor variables are highly correlated with each other, which can inflate standard errors and destabilize coefficient estimates. We assess this using the Variance Inflation Factor (VIF) for each predictor via the `car::vif` function.

#calculate VIF values for each predictor variable in our model
car::vif(model)

student balance income
2.754926 1.073785 2.694039

A VIF value exceeding 5 or 10 is typically considered problematic, indicating severe multicollinearity. Since all VIF values in our model are low (below 3), we can confidently conclude that multicollinearity is not an issue among these predictors. This guarantees that the standard errors and coefficient estimates we derived are reliable.

Step 5: Generating Probabilistic Predictions

Once the logistic regression model has been rigorously fitted and validated on the training set, its core utility lies in its ability to predict the probability of the outcome for new, unseen data points. We utilize the `predict()` function in R for this purpose. It is essential to include the argument `type="response"`; this tells R to output the predicted probability (p(X))--a value between 0 and 1--rather than the raw log odds.

To illustrate how the model differentiates individuals, consider two hypothetical cases: a student and a non-student, both sharing identical high balance and low income characteristics. We input these profiles into the model to see their respective predicted default probabilities:

```
#define two individuals
new <- data.frame(balance = 1400, income = 2000, student = c("Yes", "No"))

#predict probability of defaulting
predict(model, new, type="response")
```

```
1 2
0.02732106 0.04397747
```

The results demonstrate a subtle but important finding: the student (Individual 1) has a predicted default probability of **2.73%**, which is lower than the non-student (Individual 2) at **4.40%**, despite identical balance and income. This difference directly reflects the negative coefficient associated with the `studentYes` variable we observed earlier, suggesting that, holding finances constant, being a student slightly reduces the risk profile in this population.

Moving beyond hypothetical cases, the next crucial step is to generate predictions for the entire reserved test dataset. These calculated probabilities form the foundation for evaluating the model's accuracy, sensitivity, and specificity on independent data in the final diagnostic stage.

```
#calculate probability of default for each individual in test dataset
predicted <- predict(model, test, type="response")
```

Step 6: Optimizing Thresholds and Evaluating Performance

The final and most critical phase of the modeling process involves a rigorous evaluation of the model's performance on the independent test data. While 0.5 is commonly used as the default classification threshold, models often benefit from an optimized threshold that maximizes overall accuracy or balances sensitivity and specificity based on the specific business need.

We begin by determining this optimal probability threshold using the `optimalCutoff()` function from the `InformationValue` package. Note that R requires the categorical "Yes"/"No" factor levels in the test data to be converted into numerical 1s and 0s (where 1 represents default) before comparison with the predicted probabilities.

```
library(InformationValue)

#convert defaults from "Yes" and "No" to 1's and 0's
```

```
test$default <- ifelse(test$default=="Yes", 1, 0)

#find optimal cutoff probability to use to maximize accuracy
optimal <- optimalCutoff(test$default, predicted)
optimal
```

0.5451712

The calculated optimal threshold is **0.5451712**. This value should now be used to classify test observations: any predicted probability equal to or above 0.5451712 is classified as a default (1); anything below is classified as a non-default (0). Using this optimized cutoff, we generate a confusion matrix, which provides a detailed breakdown of the model's predictive successes and failures against the true outcomes:

```
confusionMatrix(test$default, predicted)
```

0 1
0 2912 64
1 21 39

The confusion matrix allows us to calculate key performance metrics. Given that our dataset is highly imbalanced, relying solely on accuracy can be misleading. Therefore, we calculate sensitivity and specificity:

Sensitivity (True Positive Rate): Measures the proportion of actual positive cases (defaults) that were correctly identified by the model.

Specificity (True Negative Rate): Measures the proportion of actual negative cases (non-defaults) that were correctly identified.

Misclassification Error Rate: The percentage of total observations that were incorrectly classified.

```
#calculate sensitivity
sensitivity(test$default, predicted)
```

0.3786408

```
#calculate specificity
specificity(test$default, predicted)
```

0.9928401

```
#calculate total misclassification error rate
misClassError(test$default, predicted, threshold=optimal)
```

```
0.027
```

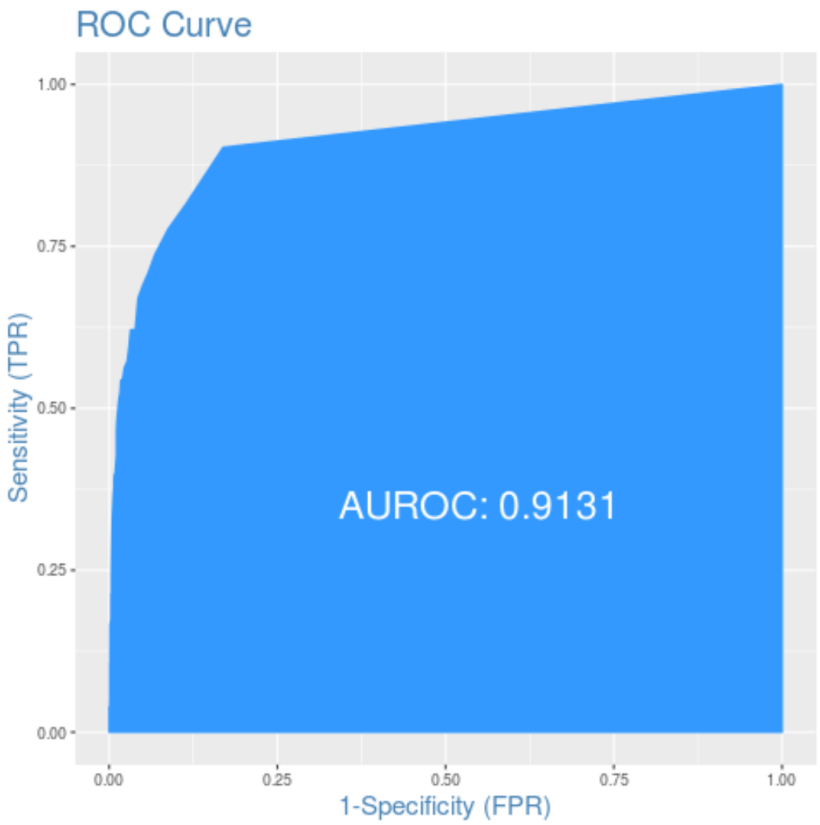
The results show exceptionally high specificity (99.28%), indicating that the model is extremely adept at correctly identifying individuals who are safe credit risks. However, the sensitivity (37.86%) is significantly lower, meaning the model misses approximately two-thirds of the actual default cases (false negatives). Despite this trade-off, the overall **misclassification error rate** remains very low at **2.7%**, suggesting strong overall performance in this skewed dataset.

Step 7: Visualizing Predictive Power with the ROC Curve

To gain a comprehensive view of the model's performance across all possible classification thresholds, we rely on the ROC (Receiver Operating Characteristic) Curve. This powerful diagnostic tool plots the True Positive Rate (Sensitivity) against the False Positive Rate (1 - Specificity).

The most critical metric derived from the ROC curve is the **AUC (Area Under the Curve)**. The AUC quantifies the model's ability to discriminate between the two classes (default vs. non-default). An AUC value of 0.5 suggests the model is no better than random chance, while an AUC of 1.0 represents a perfect classifier. Generally, an AUC above 0.8 is considered excellent.

```
#plot the ROC curve
plotROC(test$default, predicted)
```



The AUC value displayed on the plot is **0.9131**. Since this value is remarkably close to 1, it provides definitive evidence that our logistic regression model is highly effective and accurate in predicting the likelihood of individual financial default based on the provided predictor variables. This strong result confirms the model's viability for deployment in a real-world predictive scenario.

The complete R code used in this tutorial can be found here.

The complete R code used in this tutorial can be found here.

The complete R code used in this tutorial can be found here.