Create a Confusion Matrix in R (Step-by-Step)

Authored by

Mohammed looti

November 5, 2025

Logistic Regression stands as a cornerstone in statistical modeling, particularly essential when dealing with scenarios where the response variable falls into a binary classification (such as Yes/No, 1/0, or Default/No Default). Diverging significantly from standard linear regression, this powerful technique employs a sophisticated logit function to meticulously estimate the probability of a specific outcome occurring. Its utility spans a vast landscape, proving indispensable across various domains, including predictive finance, clinical diagnostics, and risk assessment. Given its crucial role in high-stakes decision-making, the rigorous evaluation of how accurately these models perform is not just recommended, but absolutely mandatory prior to deployment and operational use.

The most reliable and universally accepted method for assessing the quality and predictive efficacy of any classification model--including models built using Logistic Regression--is by constructing and analyzing a **confusion matrix**. This fundamental diagnostic instrument offers a straightforward, tabular representation that contrasts the model's predicted classifications against the actual outcomes observed in a dedicated test dataset. By utilizing this matrix, data scientists gain clear insight into precisely where the model exhibited successful predictive power and, critically, where it failed or introduced errors.

For binary classification challenges, the **confusion matrix** is conventionally structured as a 2x2 table, providing a concise summary of four primary results: True Positives, True Negatives, False Positives, and False Negatives. This structured reporting mechanism is pivotal because it allows for the calculation of a diverse array of performance metrics. These derived statistics collectively offer a holistic perspective on the model's performance, encompassing not only simple accuracy but also nuanced measures such as precision, recall, and the precise trade-offs inherent in its predictions.

The Foundation of Classification Evaluation

A machine learning model's performance cannot be judged by overall accuracy alone, especially when dealing with imbalanced datasets where one outcome class is significantly more common than the other. The **confusion matrix** moves beyond this simple measure by providing a granular breakdown of the model's predictive success across every possible outcome category. This transparency is vital for understanding the true cost and benefit of a model in a real-world scenario, where different types of errors carry different levels of impact.

The table visually organizes the counts of correct and incorrect predictions made by the classifier relative to the actual class labels. For instance, in a medical context, a model predicting disease presence, this matrix would instantly reveal how often the model correctly identified sick patients versus how often it issued a false alarm or, worse, missed an actual case. This quantitative assessment guides the optimization process, ensuring that the final model alignment meets the

specific risk tolerance and business objectives of the project.

The visual representation below illustrates the canonical structure of this essential diagnostic tool, forming the basis for all subsequent performance calculations we will discuss.

		Predicted	
		0	**1**
Actual	**0**	30	12
	1	8	56

The subsequent sections will provide a highly detailed, step-by-step methodology for constructing and meticulously interpreting this **confusion matrix** specifically within the powerful R programming environment. We will leverage highly robust and specialized R packages, specifically engineered for standardized machine learning evaluation, thereby ensuring reliable and reproducible outcomes that adhere to industry best practices.

Deconstructing the Confusion Matrix Components

Before proceeding to the practical implementation in code, it is absolutely paramount to establish a clear understanding of the four fundamental components that constitute the confusion matrix. These four counts serve as the building blocks for all subsequent derived metrics and performance indicators:

True Positives (TP): These are instances where the model correctly identified the positive class. For example, the model correctly predicted that a customer would default on their loan, and the customer actually defaulted.
True Negatives (TN): These instances represent cases where the model correctly identified the negative class. For example, the model correctly predicted that a customer would not default, and the customer did not default.
False Positives (FP) - Type I Error: This scenario occurs when the model incorrectly predicts the positive class, but the actual outcome was negative. This is often termed a "false alarm." For instance, incorrectly predicting a non-defaulter will default, leading to unnecessary restrictions or intervention.
False Negatives (FN) - Type II Error: This is arguably one of the most critical errors, occurring when the model incorrectly predicts the negative class, but the actual outcome was positive. This represents a "miss." For instance, incorrectly predicting a defaulter will not default, leading to potential financial loss or failure to treat a disease.

In various real-world applications, the financial or societal cost associated with a Type I error (False Positive) versus a Type II error (False Negative) can differ substantially. Consider, for example, an airport security screening system: a False Negative (missing a threat) is drastically more dangerous than a False Positive (a false alarm). Conversely, in targeted marketing, a high rate of False Positives might simply lead to wasted marketing spend, while a high rate of False Negatives means missing out on valuable customer conversions. The quantitative output of the **confusion matrix** allows project stakeholders to explicitly quantify and manage these error types, thereby guiding the selection of the most suitable model for a specific operational context.

By providing these explicit counts, the matrix offers a transparent and actionable view of the model's ability to predict across both the minority and majority classes, ensuring model assessment moves beyond simplistic accuracy metrics that often mask poor performance in scenarios characterized by class imbalance.

Step 1: Preparing the Data and Fitting the Model in R

The initial phase of any robust machine learning evaluation involves meticulous setup of the R programming environment and preparation of the raw data for modeling. For the purpose of this comprehensive tutorial, we will utilize the well-established **Default** dataset, which is conveniently packaged within the **ISLR** library. Our analytical objective is to develop a predictive model that estimates the probability of an individual defaulting on their loan, based on three crucial predictor variables: their **student status**, their outstanding **bank balance**, and their recorded **annual income**.

Successful implementation requires loading several specialized packages. The caret package (Classification and Regression Training) is indispensable for its capabilities in streamlining both model training and structured evaluation. Additionally, the `InformationValue` package offers specialized functions crucial for determining the optimal classification cutoff and deriving performance metrics. Most importantly, before any model fitting occurs, we must perform rigorous data splitting. This process involves partitioning the dataset into two distinct subsets: a larger training set (typically 70% of the data) used to teach the model, and a smaller, reserved test set (30%) used exclusively to evaluate the model's performance on previously unseen data. This critical step ensures the prevention of overfitting, confirming the model's true generalization ability.

The core modeling action in this step involves invoking the standard `glm()` function (Generalized Linear Model) available in R. By setting the parameter `family="binomial"`, we explicitly instruct R to fit a Logistic Regression model, which is the mathematically appropriate choice for our binary outcome variable (Default/No Default).

#load necessary packages
library(caret)

```
library(InformationValue)
library(ISLR)

#load dataset
data <- Default

#split dataset into training and testing set (70% train, 30% test)
set.seed(1)
sample <- sample(c(TRUE, FALSE), nrow(data), replace=TRUE, prob=c(0.7,0.3))
train <- data
test <- data

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

Once the model successfully completes its training phase using the 70% training subset, it possesses the optimized coefficient values necessary to calculate the predicted probability of default for any new observation. The subsequent critical step is to deploy this newly trained model against the reserved, untouched test set to accurately gauge its genuine predictive effectiveness in a simulated deployment environment.

Step 2: Optimizing the Cutoff and Generating the Matrix

A Logistic Regression model, by design, produces a continuous probability value ranging from 0 to 1. To translate this probability into a definitive binary classification (either 'Default' or 'No Default'), a critical decision point must be established: the **cutoff threshold**. If the calculated predicted probability exceeds this predetermined threshold, the observation is rigorously classified as 'Positive' (1); conversely, if it falls below the threshold, it is classified as 'Negative' (0).

While 0.5 is the statistically neutral, standard default threshold, relying on it often proves suboptimal, especially in applications where class imbalance or differential error costs exist. To address this, we leverage the specialized `optimalCutoff()` function from the `InformationValue` package. This function systematically scans potential thresholds to identify the specific value that maximizes a chosen performance metric, typically overall accuracy or the Youden index. Defining this optimal cutoff threshold ensures that our classification balance is the best possible compromise for the model structure. Once this optimal threshold is defined and applied, we can generate the final, detailed confusion matrix using the powerful `confusionMatrix()` function provided by the caret package.

It is important to note the preceding requirement: the initial step of the code block ensures that the actual response variable in the test set is properly encoded numerically (1s for 'Yes' and 0s for

'No') to facilitate accurate mathematical calculation of the subsequent metrics.

```
#use the trained model to predict probability of default on the test set
predicted <- predict(model, test, type="response")

#convert actual defaults from categorical ("Yes"/"No") to numeric (1/0)
test$default <- ifelse(test$default=="Yes", 1, 0)

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

#create confusion matrix using actual vs. predicted values
confusionMatrix(test$default, predicted)

0 1
0 2912 64
1 21 39
```

The interpretation of the resulting matrix is straightforward yet profound. In this standard output format, the rows consistently represent the Actual observed values, while the columns represent the Predicted classification values. We can now dissect the raw counts:

True Negatives (TN): 2,912 individuals were correctly identified as highly likely to have "No Default" (Actual 0, Predicted 0).
False Positives (FP): 64 individuals were incorrectly flagged as likely to "Default" (Actual 0, Predicted 1). These represent Type I errors, or false alarms.
False Negatives (FN): 21 individuals were incorrectly classified as "No Default" (Actual 1, Predicted 0). These represent Type II errors, or missed detections.
True Positives (TP): 39 individuals were correctly identified as likely to "Default" (Actual 1, Predicted 1).

This immediate quantitative breakdown immediately illuminates the significant class imbalance present within the test data (with 2,976 non-defaulters versus only 60 defaulters). It also clearly demonstrates that the model excels in identifying the majority class (non-defaulters) but has a much smaller absolute number of successful predictions for the crucial minority class (defaulters).

Step 3: Deriving and Interpreting Key Performance Metrics

While the raw counts within the confusion matrix are highly informative, the true measure of a model's effectiveness is derived from the calculated metrics. These quantitative measures are essential for objective comparison between competing models and for establishing clear

performance benchmarks required for production deployment. Our focus here is on the primary metrics that quantify the model's ability to correctly identify both positive and negative cases.

The most critical performance metrics extracted directly from the confusion matrix are:

Sensitivity (Recall or True Positive Rate): This metric is mathematically calculated as TP / (TP + FN). Sensitivity measures the proportion of all actual positive cases that were correctly identified by the model. In the context of our loan default analysis, it quantifies the percentage of customers who actually defaulted that the model successfully predicted would default. Achieving high **Sensitivity** is paramount in situations where the cost of missing a true positive case (a False Negative) is exceptionally high or catastrophic.

Specificity (True Negative Rate): Calculated as TN / (TN + FP), Specificity measures the proportion of all actual negative cases that were correctly identified. In our analysis, this is the percentage of individuals who genuinely did *not* default that the model correctly predicted would not default. High **Specificity** is critical when the business objective is to minimize false alarms (False Positives).

Total Misclassification Error Rate: This simple aggregate metric is calculated as (FP + FN) divided by the Total Observations. It provides the overall percentage of incorrect classifications made by the model, irrespective of whether the error was a Type I or a Type II mistake.

The following R code utilizes the streamlined functions available in the `InformationValue` package to rapidly calculate these indispensable performance indicators, ensuring that the optimal classification threshold determined in the preceding step is maintained for consistency.

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

0.3786408

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

0.9928401

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

0.027

A thorough analysis of these quantitative results immediately reveals an inherent trade-off within the current model configuration. The model demonstrates an exceptionally high **Specificity**, registering at approximately 99.3%. This confirms the model is highly effective and reliable at

correctly identifying customers who are genuinely low-risk and will *not* default. Conversely, the model's Sensitivity is remarkably low, standing at only about 37.9%. This low value indicates a serious limitation: while the model rarely issues a false alarm (low FP), it fails to successfully capture the majority (62.1%) of customers who actually default. These missed cases represent the costly False Negatives (FN).

Although the total misclassification error rate for this model appears impressively low at just **2.7%**, the low sensitivity serves as a crucial warning. Relying solely on the overall accuracy figure would be deeply misleading. The high percentage of False Negatives suggests that deploying this model without further refinement could lead to significant financial losses, as many genuinely high-risk individuals are incorrectly classified as safe. This comprehensive, error-type-specific evaluation facilitated by the **confusion matrix** is therefore absolutely essential for making strategically informed decisions regarding model tuning, selection, and eventual deployment.