How to Easily Create a Confusion Matrix in R

Authored by

stats writer

December 6, 2025

A confusion matrix is an indispensable tool in the field of statistical classification, providing a comprehensive visual summary of the performance of an algorithm. Specifically, in the environment of R, creating and interpreting a confusion matrix is a fundamental skill for evaluating predictive models, particularly those involving binary classification outcomes. This step-by-step guide details the process: starting with the construction of a matrix of predicted versus actual values, and culminating in the utilization of specialized functions--such as the **confusionMatrix()** function provided by the robust caret package--to compute the matrix and derive meaningful insights. The resulting confusion matrix provides critical statistics like **true positives**, **false positives**, **false negatives**, and **true negatives**, allowing for a thorough assessment of the model's accuracy, sensitivity, and specificity.

The evaluation of any statistical or machine learning model requires robust metrics beyond simple accuracy. For models dealing with dichotomous outcomes--where the response variable is **binary** (e.g., Yes/No, 1/0)--the confusion matrix offers a granular view of predictive successes and failures across all outcome categories. This detailed insight is crucial, as misclassification errors often carry different costs depending on whether they are false positives or false negatives. Understanding these trade-offs is paramount to deploying a reliable classifier in real-world applications.

Before diving into the R implementation, it is important to understand that the classification model we are evaluating here is typically logistic regression. Logistic regression is the standard type of generalized linear model employed when the dependent variable represents a probability or a categorical outcome. This method transforms the linear output into a probability score between zero and one, which must then be converted into a definitive class prediction (e.g., default or no default) using a predefined threshold or cutoff point. The decision regarding this cutoff significantly impacts the final structure of the confusion matrix and, consequently, the resulting performance metrics.

The core concept of evaluating predictive models rests on comparing what the model predicted against what actually occurred in the test dataset. A logistic regression model is inherently designed for situations where the response variable is binary, such as predicting loan default status or disease presence. The clarity offered by a **confusion matrix** in this context is unmatched, as it systematically breaks down the four possible prediction outcomes, providing the foundational counts necessary for calculating advanced performance statistics.

One highly effective and standard method for quantitatively assessing the quality and reliability of a classification model, like logistic regression, is the generation of a confusion matrix. This powerful evaluation tool is typically a 2x2 table (for binary classification) that explicitly cross-tabulates the predicted outcomes generated by the model against the actual, observed values present in the held-out test dataset. This structured layout makes it immediately clear where the model is succeeding and where it is failing, especially when dealing with imbalanced datasets where overall

accuracy can be misleading.

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

The following detailed, step-by-step example walks through the entire process, demonstrating how to properly initialize the environment, fit the necessary model, calculate the appropriate classification threshold, and finally, generate and evaluate a comprehensive confusion matrix within the R statistical environment. This practical guide ensures that users can confidently assess their own classification models.

Understanding Logistic Regression and Binary Classification

Before constructing the evaluation metrics, we must establish the predictive foundation. Logistic regression serves as the backbone for solving classification problems involving two outcomes. It is critical to understand that the model itself outputs a probability score--the likelihood that an observation belongs to the positive class. This raw probability score is not yet a classification; it is a measure of confidence. Therefore, to transform this probability into a definitive classification (which is required for the confusion matrix), a decision boundary or cutoff point must be applied.

If the calculated probability exceeds this specific cutoff, the observation is classified as positive; otherwise, it is classified as negative. The selection of this threshold is not trivial; it determines the balance between False Positives and False Negatives, reflecting the specific priorities of the modeling task. A threshold of 0.5 is common, but methods exist, such as maximizing the Youden Index or overall accuracy, to identify a more appropriate optimal cutoff tailored to the dataset's characteristics and class imbalance.

For our practical demonstration, we will utilize a widely recognized dataset to predict a real-world scenario: loan default. This requires careful data preparation, including loading the appropriate packages and partitioning the dataset into distinct training and testing subsets. The training set is used to fit the model parameters, learning the underlying relationships between the predictors and the outcome, while the unseen testing set is reserved exclusively for evaluating the model's performance rigorously, ensuring that the results are generalizable and not overly dependent on the data used during fitting.

The choice of packages is also vital in the R ecosystem. We rely heavily on the powerful

functionalities provided by the **caret** package for standardized model evaluation and the **ISLR** package, which provides the necessary sample data used in many introductory statistics courses. Additionally, the **InformationValue** package provides convenient tools for determining the optimal classification cutoff, which is a key step often overlooked when generating a confusion matrix.

Step 1: Fitting the Logistic Regression Model in R

This initial step focuses on defining and training the predictive model using the available data. For this illustrative example, we will be utilizing the well-known **Default** dataset, which is conveniently housed within the standard ISLR package. This dataset contains crucial variables related to credit card holders, allowing us to build a model that predicts the probability that a specific individual defaults on their loan obligations. Our chosen predictors include the individual's **student status** (a categorical variable), their current **bank balance**, and their annual **income**. These variables are hypothesized to significantly influence the default status, which is our binary response variable.

The process begins with loading the required statistical libraries, including **caret**, which handles the evaluation, and **InformationValue**, which assists in finding the optimal classification cutoff, along with **ISLR** for the data access. Following library initialization, the raw data is loaded and then carefully partitioned. We employ a common practice of splitting the data into a training set (70% of observations) and a testing set (30% of observations). This split is crucial for ensuring unbiased evaluation: the model learns patterns from the training data and is then tested on completely fresh, unseen data, mimicking how the model would perform in a production environment.

The final part of this step involves calling the **glm()** function (Generalized Linear Model) to fit the logistic regression. We specify the formula predicting `default` based on our three chosen predictors and importantly set the `family` argument to `"binomial"`. This specification tells R to use the logistic link function, appropriate for converting the linear combination of predictors into a probability suitable for our binary dependent variable. We also use `set.seed()` to ensure that the random sampling used for the training/testing split is reproducible, guaranteeing consistency across different runs of the script. The following code demonstrates the necessary setup and fitting procedure:

```
# Load necessary statistical packages for modeling and evaluation
library(caret)
library(InformationValue)
library(ISLR)

# Load the target dataset
data <- Default

# Split dataset into training (70%) and testing (30%) subsets to ensure unbiased evaluation
```

```
set.seed(1) # Set seed for reproducibility
sample <- sample(c(TRUE, FALSE), nrow(data), replace=TRUE, prob=c(0.7,0.3))
train <- data
test <- data

# Fit the logistic regression model using the binomial family for binary outcome
model <- glm(default~student+balance+income, family="binomial", data=train)
```

Step 2: Generating Predictions and Determining the Optimal Cutoff

Once the logistic regression model, named `model`, has been successfully fitted to the training data, the next critical step is to apply this model to the unseen testing data (`test`) to generate predicted probabilities. We use the `predict()` function for this purpose, specifying `type="response"` to ensure the output consists of predicted probabilities--the likelihood of default--rather than the default log-odds output. These probabilities represent the model's confidence that an observation belongs to the positive class (i.e., defaults on the loan).

However, a confusion matrix requires categorical predictions (Yes/No, 1/0), not continuous probabilities. This necessitates the crucial process of defining an **optimal cutoff threshold**. While 0.5 is often used as a default cutoff, selecting an optimal threshold tailored to the specific dataset can significantly maximize performance metrics, such as overall accuracy, or adjust the balance between sensitivity and specificity based on business needs. We utilize the `optimalCutoff()` function from the **InformationValue** package to programmatically determine the best threshold that balances these performance metrics, often by maximizing overall classification accuracy.

Furthermore, before calculating the cutoff and the matrix, we must ensure that the response variable in the test set is numerically encoded (1s and 0s) if it was originally factor data ("Yes" and "No"). This standardization is necessary because most R functions dealing with binary classification metrics expect numerical inputs (0 for negative, 1 for positive). This data manipulation step ensures compatibility with the classification functions and guarantees that the comparison between predicted probabilities and true outcomes is accurate. This entire process culminates in providing the necessary inputs for constructing the actual confusion matrix, which compares these thresholded predictions against the true outcomes.

Step 3: Constructing the Confusion Matrix using caret

With the predictions generated and the optimal classification threshold established, we are now ready to construct the confusion matrix itself. This is achieved using the powerful **confusionMatrix()** function, which is a staple of the caret package ecosystem. This function takes the true, known outcomes (e.g., `test$default`, after conversion to 1s and 0s) and the model's

thresholded predictions as input, performing the essential cross-tabulation and generating a standardized matrix structure. This matrix is the definitive tool for evaluating classification performance.

The resulting matrix is a 2x2 table that visually and quantitatively summarizes the performance of the classification algorithm. The rows typically represent the actual classes (the ground truth), while the columns represent the predicted classes (the model output). Interpreting this matrix is key to model evaluation, as it directly reveals the counts of the four fundamental outcomes:

True Negatives (TN): The count of observations where the actual class was negative, and the model correctly predicted negative (e.g., correctly predicting 'No Default').
False Positives (FP): The count of observations where the actual class was negative, but the model incorrectly predicted positive (Type I error, e.g., predicting 'Default' when the person did not default).
False Negatives (FN): The count of observations where the actual class was positive, but the model incorrectly predicted negative (Type II error, e.g., predicting 'No Default' when the person actually defaulted).
True Positives (TP): The count of observations where the actual class was positive, and the model correctly predicted positive (e.g., correctly predicting 'Default').

The execution of the following code sequence first generates the predictions, transforms the factor outcomes to numerical values, calculates the optimal threshold for classification, and finally runs the **confusionMatrix()** function to display the core performance summary. Note that the output shown here provides the fundamental count matrix, essential for subsequent metric calculations:

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

Convert the actual default status from factor levels ("Yes" and "No") to binary outcomes (1's and 0's)
test$default <- ifelse(test$default=="Yes", 1, 0)

# Find the optimal cutoff probability (threshold) to use, typically aimed at maximizing accuracy
optimal <- optimalCutoff(test$default, predicted)

# Create the confusion matrix using the predicted probabilities and true values
confusionMatrix(test$default, predicted)

0 1
0 2912 64
1 21 39

Analyzing the resulting matrix counts: The row labeled '0' represents actual non-defaulters (Negative class), and the row labeled '1' represents actual defaulters (Positive class). From the output, we observe 2912 **True Negatives** (TN) and 39 **True Positives** (TP). The classification errors include 64 **False Positives** (FP) and 21 **False Negatives** (FN). This count distribution immediately indicates that the model is highly effective at identifying the negative class, but its performance drops significantly when attempting to correctly identify the positive class.

Evaluating Model Performance: Key Classification Metrics

The raw counts within the confusion matrix are crucial, but they must be translated into actionable percentage metrics to truly understand the model's effectiveness in a normalized context. These calculated metrics provide context, normalize performance across different dataset sizes, and allow for objective comparisons between different classification models or algorithms. The most commonly evaluated metrics derived directly from the matrix counts are Sensitivity, Specificity, and the overall Misclassification Error Rate.

Sensitivity and Specificity are often viewed as inverse metrics, representing a fundamental trade-off in classification systems. Adjusting the classification cutoff threshold to capture more True Positives (increasing Sensitivity) will almost inevitably result in capturing more False Positives (decreasing Specificity), and vice versa. The optimal balance depends entirely on the domain and the relative costs associated with Type I (FP) and Type II (FN) errors.

For example, in a financial lending context, the cost of a False Negative (failing to predict a default that occurs) results in direct monetary loss, whereas the cost of a False Positive (denying a loan to a reliable customer) results in lost business opportunity. These metrics help articulate the exact nature of these trade-offs.

Sensitivity (Recall): Also known as the **True Positive Rate**. This metric quantifies the model's ability to correctly identify all relevant positive cases. It is calculated as True Positives divided by the total number of actual positive cases (TP / (TP + FN)). In our loan example, it is the percentage of individuals who actually defaulted that the model correctly predicted would default.
Specificity: Known as the **True Negative Rate**. This metric measures the proportion of actual negative cases that were correctly identified. It is calculated as True Negatives divided by the total number of actual negative cases (TN / (TN + FP)). For our model, this represents the percentage of individuals who would *not* default that the model correctly predicted as non-defaulters.
Total Misclassification Error Rate: This is the simplest measure of overall error, providing the inverse of overall accuracy. It represents the percentage of total observations that were incorrectly classified by the model, regardless of whether they were False Positives or False Negatives. It is calculated as the sum of all errors divided by the total number of observations ((FP + FN) / Total Observations). A lower rate indicates higher overall accuracy, but must be viewed critically if the

data is highly imbalanced.

By calculating these critical ratios, we transition from raw counts of hits and misses to quantifiable, normalized measures of model reliability, which are essential for comparing performance across different datasets or algorithms.

Calculating and Interpreting Performance Statistics

To quantify the performance of our logistic regression model accurately, we use specialized functions, specifically `sensitivity()`, `specificity()`, and `misClassError()`, which are conveniently accessible through the **InformationValue** package when working with predicted probabilities. These functions simplify the calculation process, allowing analysts to quickly extract the ratios corresponding to the raw counts derived from the cross-tabulation without manual formula input.

The calculation of **Sensitivity** reveals how effective the model is at catching the events of interest--in this case, loan defaults. Given the high consequence of missing a default (a False Negative), this value is often a key focus in financial modeling. Conversely, the calculation of **Specificity** gauges the model's ability to correctly rule out the event of interest, minimizing unnecessary interventions or incorrect classifications for the majority class.

Finally, the **Total Misclassification Error Rate** provides a single, easy-to-digest figure representing the overall proportion of mistakes. Since the optimal cutoff was used in Step 2 to maximize accuracy, this error rate provides the best possible overall performance measure achievable with this particular model structure and dataset split. It is absolutely crucial to pass the calculated `optimal` threshold into the `misClassError()` function to ensure the error is calculated based on the boundary that optimizes model performance.

 # Calculate Sensitivity (True Positive Rate) using the test data and predicted probabilities
sensitivity(test$default, predicted)

0.3786408

# Calculate Specificity (True Negative Rate)
specificity(test$default, predicted)

0.9928401

# Calculate total misclassification error rate, applying the previously determined optimal threshold
misClassError(test$default, predicted, threshold=optimal)

0.027

Conclusion: Interpreting the Model's Performance Profile

The statistical output confirms the model's performance profile based on the chosen optimal cutoff. The **Specificity** value of approximately 99.28% is exceptionally high. This outstanding result indicates that the logistic regression model is extremely reliable in predicting individuals who will **not** default on their loans, resulting in a minimal number of False Positives. This strength is vital in applications where minimizing incorrect flagging of the negative class is paramount.

Conversely, the **Sensitivity** value of 37.86% is relatively low. This indicates a significant weakness in identifying the positive class: the model only correctly identifies about 38% of the individuals who actually defaulted. The implication is a high rate of False Negatives (21 in the matrix), meaning the model failed to predict default for a large segment of individuals who eventually defaulted. Depending on the business objective--if minimizing loan loss (i.e., maximizing sensitivity) is the priority--this model might require further refinement, feature engineering, or a deliberate adjustment of the cutoff threshold to favor catching more positive cases, even if it slightly lowers specificity.

Overall, the **total misclassification error rate** is remarkably low, standing at only **2.7%** for this specific model configuration and cutoff. While this headline accuracy figure looks excellent, the detailed metrics provided by the binary classification confusion matrix reveal the nuanced trade-offs. The model's high overall accuracy is primarily driven by the large number of correctly identified negative cases (non-defaulters), reflecting the imbalance inherent in the dataset. Consequently, while the model is accurate in general, it demonstrates clear challenges and potential limitations in successfully identifying and predicting the minority, positive class (defaulters).