Identifying risky bank loans

Nithya Sarabudla

02-25-2024

Step 1 -Collecting data

Step 2 – exploring and preparing the data

The credit dataset is a collection of data encompassing 1,000 loan records. It integrates a mix of numerical and nominal attributes that detail both the characteristics of the loans and the profiles of the applicants. Among these attributes are factors such as checking_balance, months_loan_duration, credit_history, purpose, and amount, which vary across several levels and categories. The dataset also includes a crucial class variable indicating whether a loan defaulted, making it a valuable resource for analyzing and predicting loan default risks.

```
# Load the credit dataset and convert character variables to factors
credit <- read.csv("/Users/nithyasarabudla/Downloads/credit.csv", stringsAsFactors = TRUE)
# Display the structure of the credit dataset to check variable types and data frame structure
str(credit)
```

```
## 'data.frame':    1000 obs. of  21 variables:
##  $ checking_balance    : Factor w/ 4 levels "< 0 DM","> 200 DM",..: 1 3 4 1 1 4 4 3 4 3 ...
##  $ months_loan_duration: int  6 48 12 42 24 36 24 36 12 30 ...
##  $ credit_history      : Factor w/ 5 levels "critical","delayed",..: 1 5 1 5 2 5 5 5 5 1 ...
##  $ purpose             : Factor w/ 10 levels "business","car (new)",..: 8 8 5 6 2 5 6 3 8 2 ...
##  $ amount              : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ savings_balance     : Factor w/ 5 levels "< 100 DM","> 1000 DM",..: 5 1 1 1 1 5 4 1 2 1 ...
##  $ employment_length   : Factor w/ 5 levels "> 7 yrs","0 - 1 yrs",..: 1 3 4 4 3 3 1 3 4 5 ...
##  $ installment_rate    : int  4 2 2 2 3 2 3 2 2 4 ...
##  $ personal_status     : Factor w/ 4 levels "divorced male",..: 4 2 4 4 4 4 4 4 1 3 ...
##  $ other_debtors       : Factor w/ 3 levels "co-applicant",..: 3 3 3 2 3 3 3 3 3 3 ...
##  $ residence_history   : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ property            : Factor w/ 4 levels "building society savings",..: 3 3 3 1 4 4 1 2 3 2 ...
##  $ age                 : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ installment_plan    : Factor w/ 3 levels "bank","none",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ housing             : Factor w/ 3 levels "for free","own",..: 2 2 2 1 1 1 2 3 2 2 ...
##  $ existing_credits    : int  2 1 1 1 2 1 1 1 1 2 ...
##  $ default             : int  1 2 1 1 2 1 1 1 1 2 ...
##  $ dependents          : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ telephone           : Factor w/ 2 levels "none","yes": 2 1 1 1 1 2 1 2 1 1 ...
##  $ foreign_worker      : Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...
##  $ job                 : Factor w/ 4 levels "mangement self-employed",..: 2 2 4 2 2 4 2 1 4 1 ...
```

This chunk of code is responsible for importing the credit dataset from a CSV file. By setting stringsAsFactors = TRUE, all character columns in the dataset are automatically converted to factors, which is suitable for the categorical data present in the dataset.

```r
# Display the distribution of checking account balances among loan applicants
table(credit$checking_balance)
```

```
##
##     < 0 DM   > 200 DM 1 - 200 DM    unknown
##        274         63        269        394
```

```r
# Display the distribution of savings account balances among loan applicants
table(credit$savings_balance)
```

```
##
##      < 100 DM     > 1000 DM  101 - 500 DM 501 - 1000 DM       unknown
##           603            48           103            63           183
```

The code analyzes the checking account balances of loan applicants, categorizing them into four groups: "< 0 DM", "> 200 DM", "1 - 200 DM", and "unknown". It's a preliminary step to identify patterns that might correlate with loan default risks. The savings account balances are categorized and counted, similar to the checking balances. This distribution might also serve as a predictor for loan defaulting, considering that savings account balances could reflect an applicant's financial stability.

```r
# Summarize the distribution of loan duration in months
summary(credit$months_loan_duration)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     4.0    12.0    18.0    20.9    24.0    72.0
```

```r
# Summarize the distribution of loan amounts
summary(credit$amount)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     250    1366    2320    3271    3972   18424
```

The statistical summary of the loan duration, offering insights into the typical length of loans and the range applicants are requesting. Understanding loan duration is important for assessing the risk profile of the loan portfolio. The summary of loan amounts requested by applicants highlights the financial magnitude involved in the loans. It shows the range, mean, and median amounts, which can be critical in determining the bank's exposure to risk.

```r
# Display the distribution of checking account balances among loan applicants
table(credit$default)
```

```
##
##   1   2
## 700 300
```

Counts how many loans ended in default versus those that did not, giving an idea of the default rate.

Data preparation – creating random training and test datasets

```
# Ensures reproducibility of random sample
set.seed(9829)
# Randomly selects 900 samples for training
train_sample <- sample(1000, 900)
str(train_sample)
```

```
##  int [1:900] 653 866 119 152 6 617 250 343 367 138 ...
```

This code sets up the foundation for splitting the dataset by randomly selecting 900 out of 1,000 records for training. The set.seed function ensures that the random selection is reproducible.

```
# Creates the training dataset
credit_train <- credit[train_sample, ]
 # Creates the test dataset by excluding the training samples
credit_test  <- credit[-train_sample, ]
```

These lines split the credit dataset into two parts: 90% for training and 10% for testing, using the indices generated by the sample function.

```
# Checks the proportion of defaults in the training set
prop.table(table(credit_train$default))
```

```
##
##         1         2
## 0.7055556 0.2944444
```

```
# Checks the proportion of defaults in the test set
prop.table(table(credit_test$default))
```

```
##
##    1    2
## 0.65 0.35
```

These commands assess whether the split between training and testing datasets maintains a similar proportion of defaults, ensuring that both datasets are representative.

Step 3 – training a model on the data

```
# Loads the C50 package for decision tree models
library(C50)
# Trains a decision tree model on the training dataset
credit_model <- C5.0(factor(default) ~ ., data = credit_train)
credit_model
```

```
##
## Call:
## C5.0.formula(formula = factor(default) ~ ., data = credit_train)
##
```

```
## Classification Tree
## Number of samples: 900
## Number of predictors: 20
##
## Tree size: 73
##
## Non-standard options: attempt to group attributes
```

This section loads the necessary library for decision tree modeling and trains a C5.0 model using all predictors in the credit_train dataset to predict the default status.

Step 4 – evaluating model performance

```r
# Generates predictions for the test dataset
credit_pred <- predict(credit_model, credit_test)
```

This line uses the trained model to predict default status on the unseen credit_test dataset.

```r
library(gmodels)
CrossTable(credit_test$default, credit_pred,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
           dnn = c('actual default', 'predicted default'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  100
##
##
##                | predicted default
## actual default |         1 |         2 | Row Total |
## ---------------|-----------|-----------|-----------|
##              1 |        56 |         9 |        65 |
##                |     0.560 |     0.090 |           |
## ---------------|-----------|-----------|-----------|
##              2 |        27 |         8 |        35 |
##                |     0.270 |     0.080 |           |
## ---------------|-----------|-----------|-----------|
##   Column Total |        83 |        17 |       100 |
## ---------------|-----------|-----------|-----------|
##
##
```

After loading the gmodels package for enhanced table functionalities, this code evaluates the model's performance by comparing the predicted default status against the actual default status in the test dataset, offering a detailed breakdown of the model's accuracy and misclassifications.

Table presents the results of a model's predictions compared to actual outcomes for loan defaults, based on a test set of 100 observations. The "actual default" rows indicate whether a loan actually defaulted (1 for no default, 2 for default), while the "predicted default" columns show the model's predictions (1 for predicted no default, 2 for predicted default). Out of 65 loans that did not default, the model correctly predicted 56 as non-defaults (a 56% accuracy rate for non-defaults) and incorrectly predicted 9 as defaults. For the 35 loans that did default, the model only correctly identified 8 (an 8% accuracy rate for defaults) and incorrectly labeled 27 as non-defaults. Overall, the model predicted non-defaults with higher accuracy, evidenced by 83 predictions as non-defaults, but struggled to accurately identify actual defaults, with a significant number of false negatives (27 out of 35 actual defaults were missed).

Step 5 – improving model performance

Boosting the accuracy of decision trees

```
# Trains a boosted model with 10 iterations
credit_boost10 <- C5.0(factor(default) ~ ., data = credit_train,
                       trials = 10)
```

This command retrains the model using boosting to improve accuracy, specifying 10 boosting iterations to enhance the model's performance by aggregating the predictions of multiple trees.

```
# Predicts using the boosted model
credit_boost_pred10 <- predict(credit_boost10, credit_test)
CrossTable(credit_test$default, credit_boost_pred10,
          prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
          dnn = c('actual default', 'predicted default'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  100
##
##
##                | predicted default
## actual default |         1 |         2 | Row Total |
## ---------------|-----------|-----------|-----------|
##              1 |        57 |         8 |        65 |
##                |     0.570 |     0.080 |           |
## ---------------|-----------|-----------|-----------|
##              2 |        23 |        12 |        35 |
##                |     0.230 |     0.120 |           |
## ---------------|-----------|-----------|-----------|
##   Column Total |        80 |        20 |       100 |
## ---------------|-----------|-----------|-----------|
##
##
```

Similar to the earlier evaluation, this code predicts default status using the boosted model and evaluates its performance, aiming to see if boosting has improved the model's ability to accurately classify default cases.

Making some mistakes cost more than others

```
# Define the dimensions of the cost matrix with two vectors indicating the possible values for predicti
matrix_dimensions <- list(c("1", "2"), c("1", "2"))
# Name the dimensions to clarify their roles in the matrix.
names(matrix_dimensions) <- c("predicted", "actual")
```

In this section, we're preparing to construct a cost matrix for use in a decision tree model, specifically a C5.0 model. The dimensions are defined to represent the possible outcomes of the model's predictions (1 for "no", 2 for "yes") against the actual outcomes. Naming the dimensions helps clarify their roles and ensures we accurately assign costs to the correct type of error later on.

```
# Initialize the error cost matrix with specific penalties for different types of prediction errors.
error_cost <- matrix(c(0, 1, 4, 0), nrow = 2)
error_cost
```

```
##      [,1] [,2]
## [1,]    0    4
## [2,]    1    0
```

Here, we define the cost matrix, assigning penalties for the different types of errors a model can make. A penalty of 0 means no cost for correct classifications. A penalty of 1 is assigned for false positives (predicting "yes" when the actual outcome is "no"), and a penalty of 4 for false negatives (predicting "no" when the actual outcome is "yes"). This reflects the idea that missing a default (false negative) is considered more costly than incorrectly identifying a non-default as a default (false positive).

```
# Build the C5.0 model with the specified cost matrix.
credit_cost <- C5.0(factor(default) ~ ., data = credit_train,
                        costs = error_cost)
```

```
## Warning: no dimnames were given for the cost matrix; the factor levels will be
## used
```

```
# Generate predictions from the model using the test dataset.
credit_cost_pred <- predict(credit_cost, credit_test)
# Evaluate the model's predictions against the actual outcomes.
CrossTable(credit_test$default, credit_cost_pred,
             prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
             dnn = c('actual default', 'predicted default'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
```

```
##
##
## Total Observations in Table:  100
##
##
##                 | predicted default
## actual default |         1 |         2 | Row Total |
## ---------------|-----------|-----------|-----------|
##              1 |        37 |        28 |        65 |
##                |     0.370 |     0.280 |           |
## ---------------|-----------|-----------|-----------|
##              2 |         4 |        31 |        35 |
##                |     0.040 |     0.310 |           |
## ---------------|-----------|-----------|-----------|
##   Column Total |        41 |        59 |       100 |
## ---------------|-----------|-----------|-----------|
##
##
```

In this final section, the C5.0 model is trained using the credit_train dataset, incorporating the cost matrix defined earlier. This influences the model to weigh certain types of errors more heavily during training, aiming to minimize the overall cost associated with misclassifications. After training, the model is evaluated using the credit_test dataset. The CrossTable function is used to generate a confusion matrix, allowing us to examine the model's performance in detail, including the number of true positives, true negatives, false positives, and false negatives.

The table displays the results of a confusion matrix for a model predicting defaults, based on a test dataset of 100 observations. The matrix categorizes predictions into two groups: "1" for no default and "2" for default. Observations are split into actual outcomes (actual default) and model predictions (predicted default).

Of the 65 actual non-default cases (labelled as "1"), the model correctly predicted 37 as non-defaults but misclassified 28 as defaults. In contrast, of the 35 actual default cases (labelled as "2"), the model successfully identified 31 as defaults while incorrectly predicting 4 as non-defaults.

This results in a 37% accuracy rate for predicting non-defaults and a 31% accuracy rate for predicting defaults, relative to the total observations. The model's overall performance shows a preference for correctly identifying defaults (88.6% accuracy in predicting defaults) over avoiding false positives, which may align with the goal of minimizing the more costly error of failing to predict a default.