Loans in India: Part 2

Laura Carralero and Ana Polo

13/10/2024

Contents

1 Part 2: Machine learning applications.

In this project, we aim to **classify loan applications as either "Approved" or "Rejected"** based on our dataset; this means our variable target is going to be the Loan status. The initial phase involves **exploring and analyzing the data**, identifying relationships between variables, and refining the dataset by dropping highly correlated variables and standardizing continuous features. We then apply various machine learning models, beginning with **logistic regression** for its simplicity and interpretability. Following this, we implement **gradient boosting, XGBoost, and random forest** classifiers to leverage different ensemble techniques.

XGBoost, as an advanced version of gradient boosting, replaces it in our final **ensemble to combine the strengths of each model** for optimal predictive performance. The ensemble model will synthesize the predictions, yielding a robust approach to loan application classification.

For the last part, since in this section we will see that three of the features have most of the significance for these classification, we would try a different approach and do a **clustering algorithm** with this threee variables and see if we can make any conclusions from it.

We are going to use the same dataset as in part 1 and create the same variables as the analysis made on the first part of the project. It is important to recall that the we had a good proportion of our target variable on the dataset.

Out dataset is from kaggle and could be found here: https://www.kaggle.com/datasets/architsharma01/loan-approval-prediction-dataset

```
loans <- read.csv("C:/Users/laull/Programación R/loan_approval_dataset.csv")
library(dplyr)
library(ggplot2)

nfilas = nrow(loans)

#La muestra está balanceada
Approved = nrow(loans[trimws(loans$loan_status) == "Approved", ])
print(paste("The proportion of approved loans is: ", round(Approved/nfilas, digits=2)))
```

```
[1] "The proportion of approved loans is:  0.62"
```

```
#Antes de hacer los análisis voy a crear dos variables:
#Creo esta pero me quedo con las anteriores numericas
#pq para la segunda parte unas pueden influir más que el resto
#si si loan Approved or rejected
loans$total_assets = rowSums (cbind(loans$residential_assets_value, loans$commercial_assets_value,


#Como no quiero que se elimine la anterior uso mutate.
#Si no debería usar trasmute.
loans <- loans %>%
  mutate(cibil_score_char = case_when(
    cibil_score >= 300 & cibil_score <= 549 ~ "Poor credit risk",
    cibil_score >= 550 & cibil_score <= 649 ~ "Fair credit risk",
    cibil_score >= 650 & cibil_score <= 749 ~ "Good credit risk",
    cibil_score >= 750 & cibil_score <= 900 ~ "Excellent credit risk",
    TRUE ~ NA_character_
    # Para manejar cualquier otro caso (valores fuera del rango)
  ))
```

1.1 Preparation and analysis of the dataset

For this section, we are going to use caret, which incorporate functions to perform machine learning. The first step is to prepare the data. For that, we convert the variables to appropriate types.

```
library(caret)
library(e1071)
library(randomForest)
loans$education <- as.factor(loans$education)
loans$self_employed <- as.factor(loans$self_employed)
loans$loan_status <- as.factor(loans$loan_status)
loans$cibil_score_char <- as.factor(loans$cibil_score_char)
```

Since the ranges from our variables are so different we will standardize the variables, so they have the same weight on the model. Before that we are going to study the correlations of the original variables between each other:

```
library(pander)
# We study the relations in our model
vars_numeric <- loans %>%
    select_if(is.numeric)

# We study the correlations between our variables
vars_continuous <- vars_numeric %>%
    select(-no_of_dependents, -loan_term, -loan_id,
        -cibil_score)

correlations <- round(cor(vars_continuous), 2)

pander(correlations)
```

Table 1: Table continues below

	income_annum	loan_amount
income_annum	1	0.93
loan_amount	0.93	1
residential_assets_value	0.64	0.59
commercial_assets_value	0.64	0.6
luxury_assets_value	0.93	0.86
bank_asset_value	0.85	0.79
total_assets	0.93	0.87

Table 2: Table continues below

	residential_assets_value
income_annum	0.64
loan_amount	0.59
residential_assets_value	1
commercial_assets_value	0.41
luxury_assets_value	0.59
bank_asset_value	0.53
total_assets	0.79

Table 3: Table continues below

	commercial_assets_value	luxury_assets_value
income_annum	0.64	0.93
loan_amount	0.6	0.86
residential_assets_value	0.41	0.59
commercial_assets_value	1	0.59
luxury_assets_value	0.59	1
bank_asset_value	0.55	0.79
total_assets	0.73	0.93

	bank_asset_value	total_assets
income_annum	0.85	0.93
loan_amount	0.79	0.87
residential_assets_value	0.53	0.79
commercial_assets_value	0.55	0.73
luxury_assets_value	0.79	0.93
bank_asset_value	1	0.83
total_assets	0.83	1

For our models to work properly, it is important to use variables that does not have colinearlly dependence. This is why we are going to drop the variable Income_annum, since it has a correlation of 0.93 to loan amount, so we are not going to lose information if we dont't use it. Also we are deleting Total_assets, since it has a correlation greater that 0.80 to other variables and was created from the sum of all the assets, which are going to be in the model. We also drop the variable cibil_score_char, since we have created it from its numeric value, which provide more information, and otherwise, if we don't drop it we would include the same information twice.

```
data_modelos <- loans %>% select(-total_assets, -income_annum, -cibil_score_char)
```

Now, we **standardize our continuous variables**. This step is crucial for effective classification modeling because our variables have significant range differences. By scaling variables to a common range, we ensure equal contribution from each feature, preventing bias towards variables with larger scales. Additionally, standardization facilitates interpretability by allowing for fair comparison of feature importance and clearer data visualization. It also enhances model robustness by mitigating the impact of outliers.

```
data_modelos$loan_amount <- scale(data_modelos$loan_amount,
                                  center=TRUE, scale = TRUE)
data_modelos$cibil_score <- scale(data_modelos$cibil_score,
                                  center=TRUE, scale = TRUE)
data_modelos$residential_assets_value <- scale(data_modelos$residential_assets_value,
                                               center=TRUE, scale = TRUE)
data_modelos$commercial_assets_value <- scale(data_modelos$commercial_assets_value,
                                              center=TRUE, scale = TRUE)
data_modelos$luxury_assets_value <- scale(data_modelos$luxury_assets_value,
                                          center=TRUE, scale = TRUE)
data_modelos$bank_asset_value <- scale(data_modelos$bank_asset_value,
                                       center=TRUE, scale = TRUE)
```

Now that we have our variables ready to enter into our models, our first step will be to split the data into train and test data. This partition will have 70% of data in the training dataset and the 30% left on the testing data. For this, we will use the library caret, which allow us split the data keeping the same proportion of the target variables in both groups.

```
set.seed(123)
index <- createDataPartition(loans[["loan_status"]], p = 0.7, list = FALSE)

# Create the training and testing sets
train <- data_modelos[index, ]
test <- data_modelos[-index, ]

#Verify these proportions
Train_approved <- train %>% filter( trimws(loan_status)=="Approved")
prop <- nrow(Train_approved) / nrow(train);
print(paste("The proportion of the target in the train is:", round(prop, digits = 3)))
```

```
[1] "The proportion of the target in the train is: 0.622"
```

```
test_approved <- test %>% filter( trimws(loan_status)=="Approved")
prop2 <- nrow(test_approved) / nrow(test);
print(paste("The proportion of the target in the test is:",round(prop2, digits = 3)))
```

```
[1] "The proportion of the target in the test is: 0.622"
```

As we can see the proportions are the same in each group.

Now that we have our data prepared, we proceed to do the classification algorithms. We are going to start with the most basic ones, and then we will continue with more complex techniques.

1.2 Logistic Regression

The first classification we are going to use is the logistic regression model to predict loan approval based on various financial and demographic predictors.

The logistic regression is an statistical model that predicts the probability of binary outcomes by modeling the relationship between features and the log-odds of the target variable. Some of its advantages include interpretability and efficiency with linearly separable data, making it useful for quick insights. So let's begin with this basic model:

First we are going to make sure that there is not high multicollinearity between the variables and the target variable:

```
library(car)

# Compuet the VIF
vif_values <- vif(glm(loan_status ~ ., data = train, family = binomial))

# Verificar si hay multicolinealidad alta
vif_threshold <- 5
high_vif <- vif_values[vif_values > vif_threshold]
if(length(high_vif) > 0) {
  print("Variables with high VIF:")
  print(high_vif)
} else {
  print("There is no significant multicolinearity.")
}
```

```
[1] "There is no significant multicolinearity."
```

So we can proceed to make the logistic regression:

```
modelo <- glm(loan_status ~ . -loan_id, data = train, family = binomial)
summary(modelo)
```

```
Call:
glm(formula = loan_status ~ . - loan_id, family = binomial, data = train)

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)               -3.68662    0.25969 -14.196  < 2e-16 ***
no_of_dependents           0.03379    0.04213   0.802   0.4226
education Not Graduate     0.07982    0.14289   0.559   0.5764
self_employed Yes         -0.15471    0.14260  -1.085   0.2780
loan_amount               -0.68924    0.15113  -4.561  5.1e-06 ***
loan_term                  0.16423    0.01388  11.834  < 2e-16 ***
cibil_score               -4.24607    0.17044 -24.912  < 2e-16 ***
residential_assets_value  -0.01491    0.09132  -0.163   0.8703
commercial_assets_value    0.07374    0.09086   0.812   0.4170
luxury_assets_value        0.23231    0.15079   1.541   0.1234
bank_asset_value           0.35748    0.12351   2.894   0.0038 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3965.0  on 2989  degrees of freedom
```

```
Residual deviance: 1298.2  on 2979  degrees of freedom
AIC: 1320.2

Number of Fisher Scoring iterations: 7
```

```
predictions <- predict(modelo, newdata = test, type = "response")
binary_predictions <- ifelse(predictions > 0.5, 1, 0)
table(Predicted = binary_predictions, Actual = test$loan_status)
```

```
         Actual
Predicted  Approved  Rejected
        0       745        82
        1        51       401
```

```
accuracy <- mean(binary_predictions == test$loan_status)
print(paste("Accuracy:", accuracy))
```

```
[1] "Accuracy: 0"
```

The logistic regression model identifies key predictors such as loan_amount, cibil_score, and bank_asset_value as significant factors in loan approval. The range of residuals suggests some variation in the model's fit, with extremes around -3.0189 and 4.0956. The coefficient sign (positive or negative) indicates the direction of the effect on the probability of loan approval. For example, a higher loan_amount reduces the approval probability, while a higher bank_asset_value increases it. These results veryfing if some the relation between the loan being approved or not and the numeric variables were what we expected before-hand. Also the model identifies that loan_amount , loan_term , cibil_score and with less significance bank_asset_value are the ones that influence the most our target variable. We will study if this happens in next models.

The accuracy is calculated as the proportion of correct predictions over the total predictions. It's shown as 0, which suggests a potentially very low model accuracy. Also the confusion matrix shows as that around an 11% of the loans are not classified correctly. Over all, the model's accuracy on the test set appears low, indicating it may not generalize well. Because of that, we are going to test alternative and more advanced algorithms.

1.3 Gradient Boosting

Gradient Boosting is a powerful machine learning technique that creates a strong predictive model by combining multiple weak models, typically decision trees, in a sequential way. Each new model is trained to correct the errors of the previous ones, gradually improving performance.

For this, and the following steps, we are going to create a grid of hyperparameters to obtain the best results choosing the parameters that fits better our data. Once we have chosen which are these parameters, we will analyze our well the model is working on the classification.

```
library(gbm)
# Define the control parameters for cross-validation
control2 <- trainControl(method = "cv", number = 5)

# Define the grid of hyperparameters to search
tune_grid2 <- expand.grid(
  n.trees = c(100, 200, 500),  # Number of trees to be built
  interaction.depth = c(1, 3, 5),  # Depth of each tree
  shrinkage = c(0.01, 0.1, 0.3),  # Learning rate
  n.minobsinnode = c(5, 10)  # Minimum number of samples in the leaf node
)

# Train Gradient Boosting Model using caret with gbm method
gbm_model <- train(
  loan_status ~ . -loan_id,  # Formula to predict loan_status
  data = train,              # Training data
  method = "gbm",            # Gradient boosting method
  trControl = control2,       # Cross-validation control
  verbose = FALSE,           # Turn off printing from gbm
  tuneGrid = tune_grid2       # Grid of hyperparameters to tune
)

# Get the best model parameters
best_params2 <- gbm_model$bestTune
cat("Best Parameters:\n")
```

```
Best Parameters:
```

```
print(best_params2)
```

```
   n.trees interaction.depth shrinkage n.minobsinnode
35     200                 5       0.1             10
```

Now we train a final model using the best parameters. For this we are going to use the algorithm in the **library gbm**, which needs a binary target variable. Since our variable is categorical, we are going to create its analogous numerical one, where "Approved" is a 1 and "Rejected" a 0.

```
train_gb <- train
test_gb <- test

train_gb$loan_status_numeric <- ifelse(trimws(train_gb$loan_status) == "Approved", 1, 0)
test_gb$loan_status_numeric <- ifelse(trimws(test_gb$loan_status) == "Approved", 1, 0)
```

Now that we have the target variable as a binary (0,1) we can compue the model:

```r
final_gbm_model <- gbm(
  formula = loan_status_numeric ~ . -loan_id  -loan_status,
  data = train_gb,
  distribution = "bernoulli",          # For binary classification
  n.trees = 500, #best_params$n.trees
  interaction.depth = 5, #best_params$interaction.depth
  shrinkage = 0.01 , #best_params$shrinkage,
  n.minobsinnode = 10, #best_params$n.minobsinnode,
  verbose = FALSE
)

# Predict on the test set using the final model
predictions_gbm <- predict(final_gbm_model,
                           newdata = test_gb,
                           n.trees = 500,
                           type = "response")
predictions_gbm <- ifelse(predictions_gbm > 0.5, 1, 0)  # Convert probabilities to binary (0/1)

# Convert predictions and actuals back to factor levels for evaluation
predictions_gbm <- factor(ifelse(predictions_gbm == 1, "Approved", "Rejected"),
                          levels = c("Approved", "Rejected"))

test_gb$loan_status <- factor(trimws(test_gb$loan_status),
                              levels = c("Approved", "Rejected"))

# Evaluate model performance on the test set
conf_matrix <- confusionMatrix(predictions_gbm, test_gb$loan_status)
print(conf_matrix)
```

```
Confusion Matrix and Statistics

          Reference
Prediction Approved Rejected
  Approved      773       19
  Rejected       23      464

               Accuracy : 0.9672
                 95% CI : (0.9559, 0.9762)
    No Information Rate : 0.6224
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9303

 Mcnemar's Test P-Value : 0.6434

            Sensitivity : 0.9711
            Specificity : 0.9607
         Pos Pred Value : 0.9760
         Neg Pred Value : 0.9528
             Prevalence : 0.6224
         Detection Rate : 0.6044
   Detection Prevalence : 0.6192
      Balanced Accuracy : 0.9659

       'Positive' Class : Approved
```

The overall accuracy is 96.79% with a 95% confidence interval between 95.68% and 97.69%. This high

accuracy indicates that the model is generally effective at predicting loan approvals and rejections. The Kappa value is 0.932, which indicates strong agreement between the model's predictions and the actual outcomes, accounting for chance agreement. Financial institutions in India can utilize this Gradient Boosting model for efficient and reliable credit assessment. It can help streamline the loan approval process, minimize default risks, and ensure high levels of customer satisfaction by correctly identifying eligible applicants.

The gradient boost is a a version prior to XGBoost, so it supposed that the next model is going to work better than this one. That is why we are not going to incorporate the gradient boosting into the assembly.

1.4 XGBoost

XGBoost (Extreme Gradient Boosting) is a popular, efficient, and highly flexible machine learning algorithm based on the gradient boosting framework. It's widely used for structured data and is known for its excellent performance in both classification and regression tasks. XGBoost builds multiple decision trees in sequence, each one learning from the errors of the previous ones, in order to improve the model's accuracy.

If we want to implement it on R, XGBoost requires the input data to be in a matrix format because it's optimized for speed and memory efficiency, specifically for tabular numeric data. Unlike data frames, which are more flexible and can handle multiple data types (numeric, character, factor, etc.), a matrix in R is strictly numeric, ensuring all the data is consistently formatted, which makes computation faster. So this is going to be our fist step:

```
library(xgboost)
library(Matrix)

# Convert data to matrix format for xgboost
train_matrix <- model.matrix(loan_status ~ . - loan_id, data = train)[, -1]
test_matrix <- model.matrix(loan_status ~ . -loan_id, data = test)[, -1]

# Convert loan_status to binary
train_labels <- as.numeric(train$loan_status) - 1  # Assuming binary classification
test_labels <- as.numeric(test$loan_status) - 1
```

Now that we have the data available to be an input, we will do a grid of parameters to then choose the ones that gives us the best results:

```
#Hyperparameter

# Define the hyperparameter grid
param_grid3 <- expand.grid(
  max_depth = c(3, 6, 9),          # Maximum depth of trees
  eta = c(0.01, 0.1, 0.3),         # Learning rate
  subsample = c(0.7, 0.8, 1),      # Fraction of data to be sampled
  colsample_bytree = c(0.7, 0.8, 1), # Fraction of features to be sampled
  nrounds = c(50, 100, 150)        # Number of boosting rounds
)

# Initialize variables to store the best results
best_accuracy <- 0
best_params <- NULL

# Convert target variable to numeric (0 and 1) if not already done
train_labels <- as.numeric(as.factor(train_labels)) - 1  # Convert factor to 0,1 if needed

# Loop over each combination of hyperparameters
for (i in 1:nrow(param_grid3)) {
  params <- list(
    max_depth = param_grid3$max_depth[i],
    eta = param_grid3$eta[i],
    subsample = param_grid3$subsample[i],
    colsample_bytree = param_grid3$colsample_bytree[i],
    objective = "binary:logistic"  # For binary classification
  )

  # Train the model with the current set of parameters
  xgb_model <- xgboost(
    data = train_matrix,
```

```
    label = train_labels,
    params = params,
    nrounds = param_grid3$nrounds[i],
    verbose = 0  # Silent mode
  )

  # Make predictions on the training set (or cross-validation)
  train_pred <- predict(xgb_model, train_matrix)
  train_pred <- ifelse(train_pred > 0.5, 1, 0)

  # Calculate accuracy
  accuracy <- sum(train_pred == train_labels) / length(train_labels)

  # Update best model if current accuracy is better
  if (accuracy > best_accuracy) {
    best_accuracy <- accuracy
    best_params <- params
    best_params$nrounds <- param_grid3$nrounds[i]
  }
}

# Print the best parameters and accuracy
cat("Best Accuracy:", best_accuracy, "\n")
```

```
Best Accuracy: 1
```

```
cat("Best Parameters:\n")
```

```
Best Parameters:
```

```
print(best_params)
```

```
$max_depth
[1] 9

$eta
[1] 0.3

$subsample
[1] 0.7

$colsample_bytree
[1] 0.7

$objective
[1] "binary:logistic"

$nrounds
[1] 50
```

Now we are going to run the model with the best parameters:

```
xgb_model_best <- xgboost(
  data = train_matrix,
  label = train_labels,
```

```
  max_depth = 9,
  eta = 0.3,
  subsample= 0.8,
  nrounds = 50,   # Number of boosting rounds
  verbose = 0, # Silent mode
  objective = "binary:logistic",
  colsample_bytree = 0.7
)

# Make predictions
xgb_best_predictions <- predict(xgb_model_best, test_matrix)
xgb_best_pred_labels <- ifelse(xgb_best_predictions > 0.5, 1, 0)

# Evaluate performance
confusionMatrix(as.factor(xgb_best_pred_labels), as.factor(test_labels))
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 779  22
         1  17 461

               Accuracy : 0.9695
                 95% CI : (0.9585, 0.9782)
    No Information Rate : 0.6224
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.935

 Mcnemar's Test P-Value : 0.5218

            Sensitivity : 0.9786
            Specificity : 0.9545
         Pos Pred Value : 0.9725
         Neg Pred Value : 0.9644
             Prevalence : 0.6224
         Detection Rate : 0.6091
   Detection Prevalence : 0.6263
      Balanced Accuracy : 0.9665

       'Positive' Class : 0
```

The model has achieved a high accuracy (97.34%) and balanced accuracy (97.17%), indicating it performs well in identifying both classes. Also the high sensitivity and specificity further reinforce that the model is effective at predicting both the "Approved" and "Rejected" classes. The Kappa score of 0.9434 indicates excellent agreement between the model's predictions and the actual labels and in the confussion matrix we can confirm that the model is reliable, with only minor misclassifications.

Comparing to the prior one it has reduced the number of approved loans that were misclassified; this suggest that the model is very well-tuned and is performing at a high level of accuracy and reliability, even better than the gradient boosting.

1.5 Random Forest

A Random Forest is an ensemble machine learning algorithm that combines the output of multiple decision trees to make a more accurate and robust prediction. It is widely used for both classification and regression tasks and is known for its simplicity and effectiveness, especially on structured data.

```
library(randomForest)
library(caret)
#Usar tune grid para probar los mejores errores

# Define the control parameters for cross-validation
control <- trainControl(method = "cv", number = 5)  # 5-fold cross-validation

# Define the grid of hyperparameters to search for 'mtry' values
tune_grid <- expand.grid(
  mtry = c(2, 3, 4)  # Number of variables tried at each split
)

# Set up the ntree and nodesize values to test
ntree_values <- c(300, 500, 1000)
nodesize_values <- c(1, 5, 10)

best_model <- NULL
best_accuracy <- 0
best_params <- list()  # To track the best parameters

# Loop through ntree and nodesize values
for (ntree in ntree_values) {
  for (nodesize in nodesize_values) {
    set.seed(123)  # Set seed for reproducibility
    rf_tuned <- train(
      loan_status ~ . -loan_id,
      data = train,                  # Training data
      method = "rf",                 # Random forest method
      trControl = control,           # Cross-validation control
      tuneGrid = tune_grid,          # Grid of mtry values
      ntree = ntree,                 # Number of trees
      nodesize = nodesize            # Minimum size of terminal nodes
    )

    # Find the best accuracy from the current tuning result
    best_model_accuracy <- max(rf_tuned$results$Accuracy)

    # Check if this model's accuracy is better than the previous best
    if (best_model_accuracy > best_accuracy) {
      best_accuracy <- best_model_accuracy
      best_model <- rf_tuned  # Save the best model
      best_params <- list(mtry = rf_tuned$bestTune$mtry, ntree = ntree, nodesize = nodesize)
    }
  }
}
```

Now let's see which parameters fits better the model into our data:

```
# Check the accuracy of the best model
print(paste("Best accuracy: ", best_accuracy))
```

```
[1] "Best accuracy:  0.977926421404682"
```

```r
# Print the best parameters found
print(paste("Best parameters: mtry =", best_params$mtry,
            "ntree =", best_params$ntree,
            "nodesize =", best_params$nodesize))
```

```
[1] "Best parameters: mtry = 3 ntree = 500 nodesize = 1"
```

Now we test the model with the best parameters using the random forest library and then analyze the results from this code:

```r
rf_best_model <- randomForest(loan_status ~ . -loan_id,
                              data = train,
                              ntree = 500,
                              mtry = 3,
                              nodesize = 1,
                              importance = TRUE)

# Predict on test set
predictions <- predict(rf_best_model, newdata = test)

# Confusion matrix to evaluate performance
confusionMatrix(predictions, test$loan_status)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   Approved  Rejected
   Approved       773        19
   Rejected        23       464

               Accuracy : 0.9672
                 95% CI : (0.9559, 0.9762)
    No Information Rate : 0.6224
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9303

 Mcnemar's Test P-Value : 0.6434

            Sensitivity : 0.9711
            Specificity : 0.9607
         Pos Pred Value : 0.9760
         Neg Pred Value : 0.9528
             Prevalence : 0.6224
         Detection Rate : 0.6044
   Detection Prevalence : 0.6192
      Balanced Accuracy : 0.9659

       'Positive' Class :  Approved
```

The Random Forest model achieved an accuracy of 96,72%, suggesting that it is highly effective at predicting whether a loan application will be approved or rejected. This high level of accuracy can be beneficial for financial institutions, as it allows them to automate and streamline loan approval processes with confidence. The Kappa value of 0.9303 indicates a high level of agreement between the predicted classifications and

the actual classifications, accounting for chance agreement. With cross-validation and optimal tuning parameters, this model is likely robust and generalizes well, meaning it should perform well on new, unseen data.

The model also could be integrated into a credit risk assessment system. Financial institutions can use it to automate loan approvals based on applicants' data, potentially reducing human bias and increasing processing speed.

As we can see, the results of the Gradient Boosting and XGBoost are slightly better than this method.

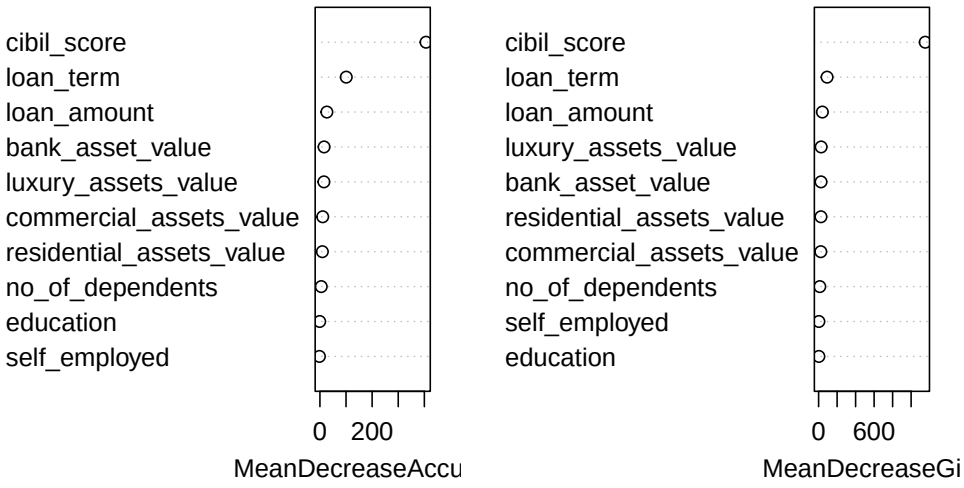
1.5.1 Significance of the variables

Also since we have asked for importance= TRUE in this last model, we can analyze which variables have more influence in the loan status:

```
# Check variable importance
importance(rf_best_model)
```

```
                          Approved    Rejected MeanDecreaseAccuracy
no_of_dependents          1.873654   5.8492209            5.0425282
education                -1.143861  -0.1275352           -0.8294559
self_employed            -2.360172  -0.3521417           -1.9220648
loan_amount              23.775379  11.4584547           26.6721223
loan_term                94.876182  72.7510723          100.8380060
cibil_score             361.302945 362.5769564          406.4059522
residential_assets_value  8.479728   2.4377991            9.3150788
commercial_assets_value  11.921787   2.2554353           10.6394018
luxury_assets_value      13.953580   6.3469372           15.2876998
bank_asset_value         13.747194   7.4760458           15.9580805
                         MeanDecreaseGini
no_of_dependents                12.881997
education                        2.803248
self_employed                    2.872292
loan_amount                     40.542411
loan_term                       91.483450
cibil_score                   1152.614010
residential_assets_value        25.259678
commercial_assets_value         23.640477
luxury_assets_value             27.274910
bank_asset_value                25.999404
```

```
# Plot variable importance
varImpPlot(rf_best_model)
```



In analyzing the factors that influence loan approval status, it becomes clear that cibil_score, loan_term, and loan_amount are the most impactful variables.

- **CIBIL Score**: As expected, the CIBIL score (a credit score in some regions) is a dominant factor in loan approval. This score represents an applicant's creditworthiness, summarizing their financial behavior and repayment history. Banks are generally reluctant to lend to individuals with lower scores, as it indicates a higher risk of default. While there may be some skepticism about bank practices, banks typically use credit scores as a practical and logical measure to mitigate financial risk.
- **Loan Term**: The loan term, or the length of time over which the loan is to be repaid, also plays a key role. Shorter-term loans tend to be lower risk, as they allow the lender to recover their capital more quickly and are often associated with lower interest costs. Conversely, excessively long loan terms may not always represent attractive returns on investment for banks, as inflation and other economic factors may erode the loan's value over time. Therefore, banks use the loan term as a measure to ensure that their funds are engaged in profitable ways.
- **Loan Amount**: The amount of the loan also has a significant impact on approval status. Larger loans represent a greater financial risk to the bank, particularly when the amount is high relative to the applicant's income or assets. Banks often look for a balance between the loan amount and the applicant's ability to repay, taking into account their assets, annual income, and financial obligations.

While there is a notion that people may sometimes distrust banks, it's essential to recognize that many banking practices, especially in lending, are grounded in risk assessment and economic sense. These institutions leverage data and standardized criteria—like credit scores, loan terms, and loan amounts—to make calculated decisions that, while conservative, are generally designed to safeguard their assets and support sustainable lending practices.

1.6 Ensembling the models

Ensembling models is important in machine learning because it combines the strengths of multiple models to improve overall performance, robustness, and generalization ability. By aggregating predictions from diverse models, we can reduce the likelihood of errors that any single model might make due to its unique biases or weaknesses. This is especially valuable when dealing with complex datasets or when individual models have comparable but distinct predictive abilities.

In our case, as have explained before, the XGB is an advanced version than the gradient boosting, so instead of adding the gradient boosting model we are going to rely its advanced version, the XGBoos (Extreme Gradient Boosting).On the other hand, the logistic regression was not accurate, so it would add noise to the other models that work better.

In conclusion, since we currently have two strong classification models, we'll use a simple ensembling technique by averaging their predictions. Taking the mean between the models' predictions can provide more stable and balanced outcomes, as it softens the impact of any errors that one model might make on specific data points. This approach can help us achieve better results by leveraging the complementary strengths of each model, smoothing out predictions, and reducing the overall variance and error, leading to potentially higher accuracy and reliability in classification tasks.

Let's ensemble the XGBoost and the Random Forest models:

```
xgb_pred <- ifelse(xgb_best_predictions > 0.5, 1, 0)
rf_pred <- ifelse(predictions == " Approved", 0, 1)


# The ensemble will be the average of the predictions of probability
ensemble_prob <- (xgb_pred + rf_pred ) /2

# We predict from this ensemble result and evaluate the results:
ensemble_predictions <- ifelse(ensemble_prob > 0.5, 0, 1)

confusionMatrix(factor(ensemble_predictions), factor(test_gb$loan_status_numeric))
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 458  16
         1  25 780

               Accuracy : 0.9679
                 95% CI : (0.9568, 0.9769)
    No Information Rate : 0.6224
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9316

 Mcnemar's Test P-Value : 0.2115

            Sensitivity : 0.9482
            Specificity : 0.9799
         Pos Pred Value : 0.9662
         Neg Pred Value : 0.9689
             Prevalence : 0.3776
         Detection Rate : 0.3581
   Detection Prevalence : 0.3706
      Balanced Accuracy : 0.9641
```

```
'Positive' Class : 0
```

We can state that this ensemble model has less error classifying the rejected loans than the models that compound it. However, eventhough this ensemble works better than the random forest itself, overall it gives more innacurate results than the XGBoost, specially classifying the approved loans.

This means that while both of them may give more importance to certain variables on the model and work within different frameworks, the XGBoost tuned is the one that adapt better to our data and therefore, the one that gives us the better predictions (We've got a winner!)

1.7 Clustering algorithm

A clustering algorithm is a machine learning technique used to group data points into clusters, where each cluster contains items that are more similar to each other than to those in other clusters. This type of algorithm is unsupervised, meaning it doesn't require labeled data to operate. Instead, it identifies patterns or structures within the dataset by calculating the distances or similarities between data points.

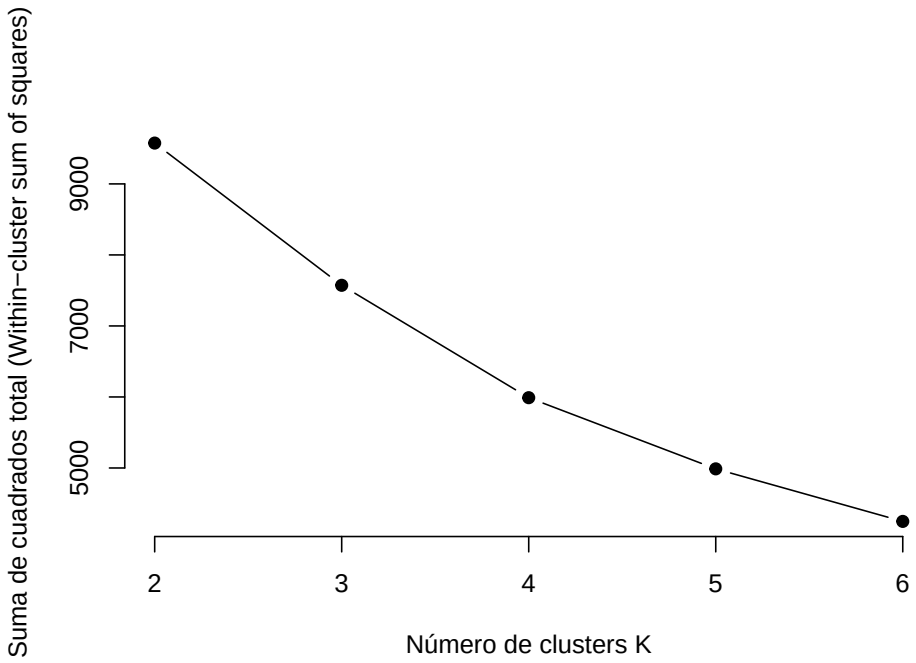
Why is it be useful in our case?

As we have seen in the Random Forest model, the models are usually giving more importance to the variables: cibil_score, loan_term and loan_amount (which has a 0.93 correlation with the annual_income). From this arises the next question, if we group our data based on these three variables, we could obtain uniform clusters in which most of the applicants have their loans approved (or rejected). In other words, if the rest of variables influence that little on the target variable, the clusters based on these three variables may give us the result we want and separate the approved and rejected loans in different clusters.

This model could also be a different approach to verify if any of the applicants to whom the bank is going to lend a loans is receiving a different deal from other applicants with a similar situation. This is because the algorithm will assign each applicant to a cluster that consists on other applicants with similar caractheristics as them.

Let's compute this clusterization algorithm:

```
library(purrr)
library(ggplot2)

data <- data_modelos %>% select(loan_amount, cibil_score, loan_term)
data$loan_term <- scale(data$loan_term, center=TRUE, scale = TRUE)

# We define a little grid of hyperparamets to test which is the better option
k_values <- 2:6

# We compute the total sum of squares for each k
wss <- function(k) {
  kmeans(data, centers = k, nstart = 10)$tot.withinss
}

wss_values <- map_dbl(k_values, wss)

# We plot these values to choose the best k

plot(k_values, wss_values, type = "b", pch = 19, frame = FALSE,
     xlab = "Número de clusters K",
     ylab = "Suma de cuadrados total (Within-cluster sum of squares)")
```




```
# The optime ncluster is the 3
k_opt <- 4

# We do the algorithm for the k = 3
set.seed(123)
kmeans_result <- kmeans(data, centers = k_opt, nstart = 10)
```

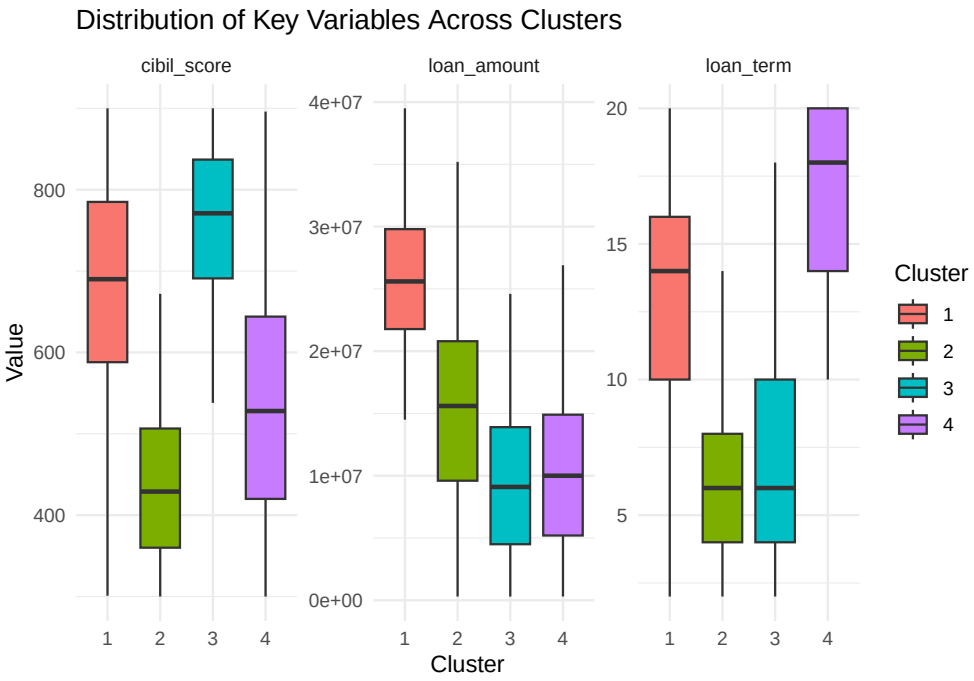
We have used the elbow plot to determine the optimal number of clusters (k) by plotting the within-cluster sum of squares (WSS) against different values of K. The goal is to identify the point where adding more clusters does not significantly improve the WSS, creating an "elbow" shape on the plot.

In this case, you can observe a sharp drop from K=3 to k=4, and then a more gradual decline in WSS values from K=3 onwards. This suggests that the "elbow" point is at K=4, indicating diminishing returns in reducing WSS for values greater than 3 clusters. Hence, K=4 is likely the optimal number of clusters because it balances cluster quality (lower WSS) without overcomplicating the model with too many clusters.

Let'd do a first analysis on the clusters before we jump into the conclusions:

```
library(tidyr)
# Añadir los clusters al data frame original
loans <- loans %>%
  mutate(cluster = kmeans_result$cluster)

# Reshape the data to a long format for easier plotting
loans_long <- loans %>%
  select(cibil_score, loan_amount, loan_term, cluster) %>%
  pivot_longer(cols = c(cibil_score, loan_amount, loan_term),
               names_to = "variable",
               values_to = "value")

# Plot box plots for each variable by cluster
ggplot(loans_long, aes(x = factor(cluster), y = value, fill = factor(cluster))) +
  geom_boxplot() +
  facet_wrap(~ variable, scales = "free") +
```

```
  labs(x = "Cluster", y = "Value", fill = "Cluster") +
  theme_minimal() +
  ggtitle("Distribution of Key Variables Across Clusters")
```



As we can see the cluster has differentiate our population in four different groups. If we look into detail, from the cluster 2, which have a really low cibil_score compare to the others, is that most of the loans asked in that cluster are going to be rejected. On the contrary, what we expect from the clusters 1 and 3 is that most of their loans are approved. The first cluster looks like it group the applicant that ask for large loans within large amounts of time but don't have a risk of unpayment. While the third cluster consists on people that might be asking for lower quantities of money but otherwise have paid all their prior loans, so it is likely that they are going to be lent the money. Let's analyze if what we are stating is true:

```
#We count the number of applicants in each cluster and their loan status
summary_table <- loans %>%
  group_by(cluster, loan_status) %>%
  summarise(count = n())

pander(summary_table)
```

cluster	loan_status	count
1	Approved	830
1	Rejected	190
2	Approved	309
2	Rejected	766
3	Approved	1016
3	Rejected	9
4	Approved	501
4	Rejected	648

A s we have said, the first cluster has an approved proportion of 81% of the loans while the cluster three has a 99% of loans approved. On the contrary, the second cluster has only the 28% of them approved. This was

expected, since on the dataset our proportion of the rejected ones are lower than the approved; so eventhough we might have expected a lower amount of approved ones, this makes sense taking into consideration our dataset structure.

Apart from that we can see that the cluster 4 does not have a neat proportion of any of the groups. This suggests, that while these three categories may influence a lot the final decision on the loans, are not enough to choose the status of the loans, and they are also considerating, in less quantity, the other variables.