"Enhancing Loan Approval Processes with SVM: A Machine Learning Approach to Financial Decision-Making"

Masters of Professional Studies in Informatics, Northeastern University

ALY 6040: Data Mining Applications

Mohammed Saif Wasay

NUID: 002815958

Prof: Harpreet Sharma

12th October 2024

Table of Contents

1. Introduction

This report aims to development a **Support Vector Machine (SVM)** predictive model to assess loan approval probabilities based on applicant data. Focused on enhancing decision-making for lenders, this model leverages SVM's strengths in binary classification and high-dimensional spaces. By analysing attributes like income, marital status, credit history, and loan amounts, it aims to improve risk management and operational efficiency. The report will cover data preparation, exploratory analysis, model training, and evaluation. The goal is to provide a reliable tool for fair and efficient borrower assessment, optimizing financial decision-making processes in lending institutions.

Methodology & Data Collection

This study employs a comprehensive dataset sourced from historical loan applications to build a Support Vector Machine (SVM) model for predicting loan approvals. The dataset comprises a rich array of applicant attributes including demographic information (e.g., gender, marital status), financial statistics (e.g., applicant income, loan amounts), and other relevant loan details. These variables are instrumental in understanding and predicting the financial behavior and creditworthiness of applicants. The data collection process ensured a robust representation of the population by including a diverse range of socioeconomic backgrounds, enabling a nuanced analysis of factors influencing loan approval decisions.

Data Preparation and code walkthrough

The initial phase of our analysis involved loading the dataset into the R environment from a designated CSV file. This was accomplished using the read.csv() function. Simultaneously, essential R libraries such as dplyr, tidyr, e1071 for SVM modeling, ggplot2 for data visualization, and gridExtra for arranging plots were loaded to enable effective data manipulation, analysis, and model training.

Code Walkthrough:

- **Library Imports**:

  The script begins by importing the required libraries:

  - **e1071:** Used for implementing the Support Vector Machine (SVM) model.
  - **dplyr:** Facilitates data manipulation tasks such as filtering, summarizing, and arranging data frames in R.
  - **ggplot2:** A data visualization library used to create various plots.
  - **tidyr:** Assists in tidying the data which includes reshaping, transforming, and cleaning data sets.

Loading the Dataset:

The dataset is loaded into the R environment for preprocessing and analysis:

- **Data Loading**: The read.csv() function is used to import data from a CSV file. This function converts the imported data into a dataframe, which is the standard data structure for statistical analysis in R.

```r
#Mohammed Saif Wasay
#NUID: 002815958
#ALY6040
#Data Mining Aplications

cat("\014") # clears console
rm(list = ls()) # clears global environment
try(dev.off(dev.list()["RStudioGD"]), silent = TRUE) # clears plots
try(p_unload(p_loaded(), character.only = TRUE), silent = TRUE) #clears packages
options(scipen = 100) # disables scientific notation for entire R session

# Load necessary libraries
library(dplyr)
library(tidyr)
library(e1071)
library(ggplot2)
library(gridExtra)

# Load the data
loan_data <- read.csv("C:/Users/Mohammed Saif Wasay/Documents/code/data/loan-train.csv")

```

Visualize Distributions of Numerical Features

To understand the distribution of key numerical attributes in the dataset, histograms are utilized. Histograms are effective in showing the frequency distribution of numerical data, helping to identify patterns such as skewness, outliers, or the central tendency.

- **Histogram of Applicant Income**:
    - ggplot(): This function initializes a ggplot object. It is the foundational step for creating plots with the ggplot2 package.
    - aes(x=ApplicantIncome): Sets the aesthetic mappings, in this case, mapping the ApplicantIncome variable to the x-axis.
    - geom_histogram(): Adds a histogram layer to the plot, with specified bins to control the number of bars and fill color to enhance visual distinction.
    - ggtitle(): Adds a descriptive title to the histogram for better readability and understanding of the plot content.

Histogram of Loan Amount:

- Similarly, this code snippet creates a histogram for the LoanAmount variable, following the same structure and methodological approach to highlight the distribution characteristics of loan amounts among applicants.

```
21
22 # Data Exploration: Visualize distributions of numerical features
23 p1 <- ggplot(loan_data, aes(x=ApplicantIncome)) + geom_histogram(bins=30, fill="blue", color="black") + ggtitle("Distribution of Applicant Income")
24 p2 <- ggplot(loan_data, aes(x=LoanAmount)) + geom_histogram(bins=30, fill="green", color="black") + ggtitle("Distribution of Loan Amount")
25
```

Visualizing Categorical Data Distribution

Bar plots are particularly useful for categorical data as they show the frequency of each category, making it easy to compare different groups.

- **Bar Plot of Gender Distribution**:
    - ggplot(): Initializes the plotting object.

- aes(x=Gender, fill=Gender): The aesthetic function here maps the Gender variable to the x-axis and also uses it to fill the bars, differentiating categories by color.
- geom_bar(): This geom creates a bar plot. By default, it counts the number of cases for each category unless specified otherwise.
- ggtitle(): Specifies the title of the bar plot.

Bar Plot of Marital Status Distribution:

- This plot follows the same structure as the gender distribution plot but maps and visualizes the Married variable, offering insights into the marital status distribution among the loan applicants.

```
# Visualize categorical data distribution
p3 <- ggplot(loan_data, aes(x=Gender, fill=Gender)) + geom_bar() + ggtitle("Gender Distribution")
p4 <- ggplot(loan_data, aes(x=Married, fill=Married)) + geom_bar() + ggtitle("Marital Status Distribution")

# Display plots
grid.arrange(p1, p2, p3, p4, nrow=2)

```

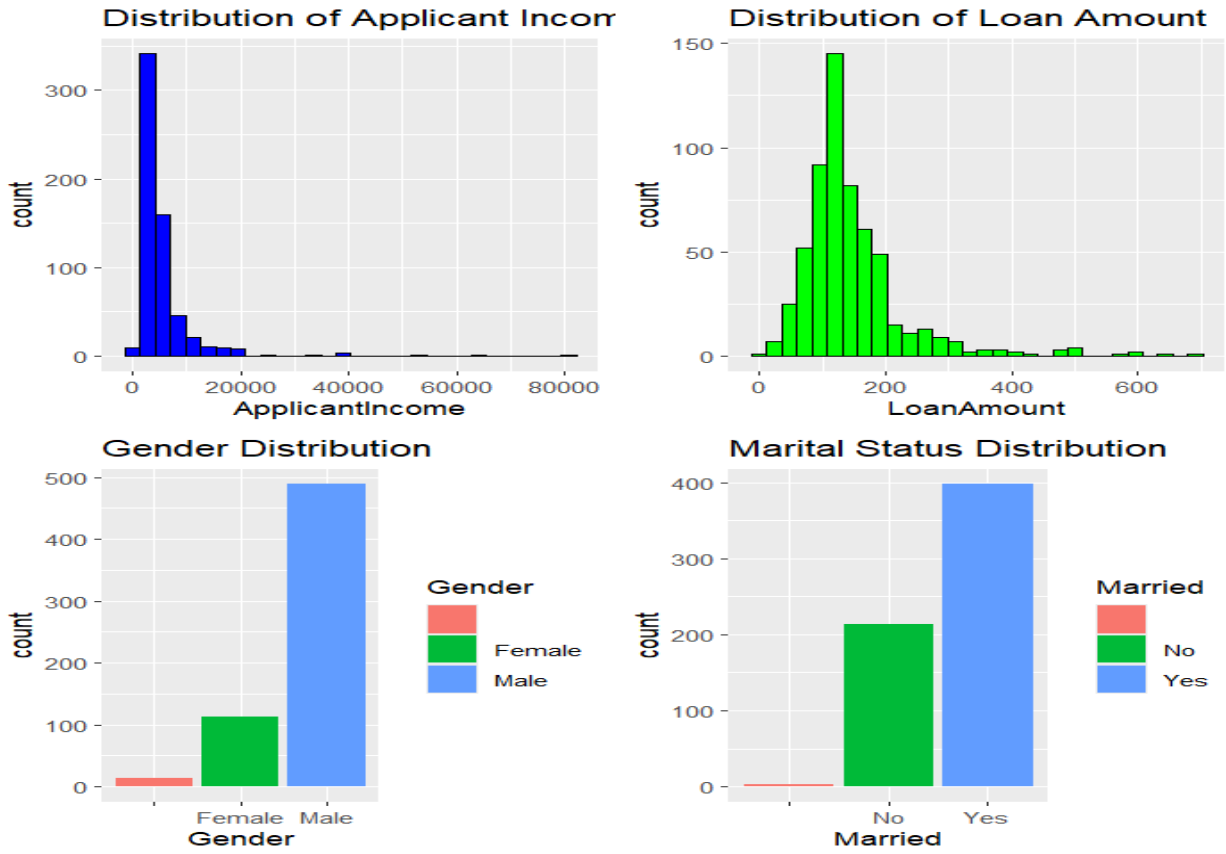

Figure 1: General Visualizations of both Numerical and Categorical Variables.

Summarizing Missing Values

Before addressing the missing data, it's essential to understand the extent and distribution of missingness across different variables in the dataset.

- **Summarizing Missing Data:**
  - **summarise():** This function from the dplyr package is used to apply summary operations to each variable in the dataset. Here, it is combined with across() to apply the summary operation across all columns.
  - **across(everything(), ~sum(is.na(.))):** The across() function is applied to every column (everything()) in the dataframe. The lambda function ~sum(is.na(.)) calculates the total number of missing (NA) values for each column.

```
33 # Handling missing values
34 missing_before <- loan_data %>% summarise(across(everything(), ~sum(is.na(.))))
```

Visualizing Missing Data:

To better understand which columns in the dataset have missing values and to what extent, a bar plot visualization is created using ggplot2.

```
35 p5 <- ggplot(missing_before %>% pivot_longer(everything(), names_to = "variable", values_to = "value"),
36              aes(x=variable, y=value)) +
37   geom_bar(stat="identity", fill="red") +
38   ggtitle("Missing Values Before Imputation")
39 print(p5)
```

- **Pivoting the Data:**
  - **pivot_longer():** Converts data from wide to long format, which is necessary for effective plotting with ggplot2. This function is part of the tidyr package.
  - **names_to** and **values_to:** Specify the names of the new key and value columns generated by pivot_longer(). Here, each column name from missing_before becomes a value under the variable column, and the count of missing values becomes a value under the value column.

- **Creating a Bar Plot:**
    - **ggplot():** Initializes a ggplot object for plotting.
    - **aes(x=variable, y=value):** Maps the variable column to the x-axis and the value column (count of missing values) to the y-axis.
    - **geom_bar(stat="identity", fill="red"):** Adds a bar plot layer to the ggplot object. stat="identity" tells ggplot that the y-values provided in the aes() are to be used as the heights of the bars. The bars are filled in red to highlight missing data.
    - **ggtitle("Missing Values Before Imputation"):** Adds a title to the plot for clarity.

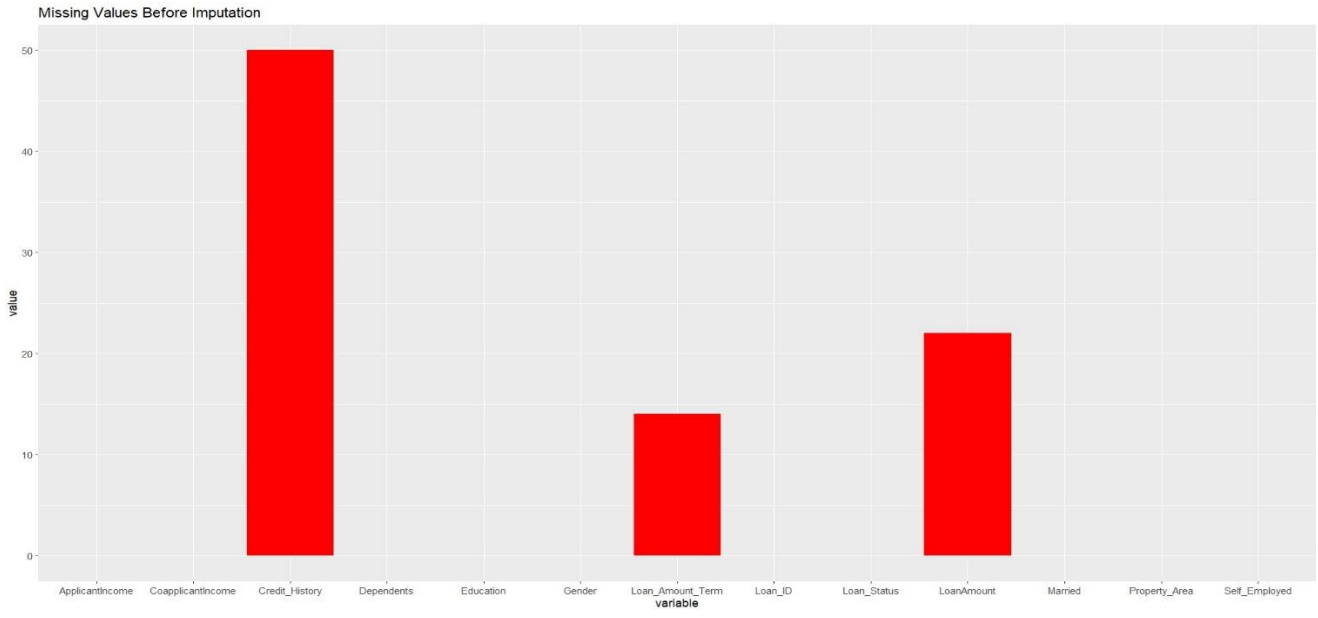

Figure 2: General Visualizations of Missing Data.

Missing Value Imputation

The goal of this step is to address missing entries in the dataset to ensure the integrity and usability of the data for further analysis and modeling. This process involves filling missing values using statistical methods or the most common value, depending on the nature of the data—categorical or numerical.

Categorical Variables For categorical variables, missing values are imputed with the most common category (mode). This approach is chosen because it maintains the distribution of data by filling in the missing entries with the most frequently occurring value.

- **Gender, Married, Dependents, Self_Employed:**
    - **names(which.max(table(loan_data$Variable, useNA = "no"))):** This line of code is used for each categorical variable. The function **table()** is used to create a frequency table of the variable, including handling NA values explicitly **(useNA = "no" ensures NAs are counted).** The which.max() function identifies the index of the maximum count in the table, essentially finding the mode. names() is then used to extract the category name corresponding to this mode.

Numerical Variables: For numerical variables, missing values are imputed with the median, which is more robust to outliers than the mean.

LoanAmount, Loan_Amount_Term, Credit_History:

- **median(loan_data$Variable, na.rm = TRUE):** Calculates the median of the variable while ignoring NA values **(na.rm = TRUE).** The median provides a central value that is less affected by outliers and skewed data.
- This value is then assigned to the missing positions in each respective numerical variable.

```
40
41 #Missing Value Imputation
42 loan_data$Gender[is.na(loan_data$Gender)] <- names(which.max(table(loan_data$Gender, useNA = "no")))
43 loan_data$Married[is.na(loan_data$Married)] <- names(which.max(table(loan_data$Married, useNA = "no")))
44 loan_data$Dependents[is.na(loan_data$Dependents)] <- names(which.max(table(loan_data$Dependents, useNA = "no")))
45 loan_data$Self_Employed[is.na(loan_data$Self_Employed)] <- names(which.max(table(loan_data$Self_Employed, useNA = "no")))
46 loan_data$LoanAmount[is.na(loan_data$LoanAmount)] <- median(loan_data$LoanAmount, na.rm = TRUE)
47 loan_data$Loan_Amount_Term[is.na(loan_data$Loan_Amount_Term)] <- median(loan_data$Loan_Amount_Term, na.rm = TRUE)
48 loan_data$Credit_History[is.na(loan_data$Credit_History)] <- median(loan_data$Credit_History, na.rm = TRUE)
49
```

Summarizing Missing Values Post-Imputation

After handling missing data through imputation, it is prudent to verify that all intended imputations were successful. This verification helps ensure data integrity before proceeding to further analysis or modeling.

- **Summarizing Missing Data:**
    - **summarise():** Part of the dplyr package, this function is used to apply summary operations across selected columns in the data frame.

- **across(everything(), ~sum(is.na(.))):** The across() function is applied to every column in the dataframe. Here, it calculates the sum of NA (missing) values for each column using sum(is.na(.)), which is a common pattern to identify missing data.

```
#After Imputation:
missing_after <- loan_data %>% summarise(across(everything(), ~sum(is.na(.))))
```

Visualizing Missing Data After Imputation

To ensure that all missing values have been addressed and to visually confirm the absence of NAs post-imputation, a bar plot is created. This visualization aids in quickly identifying any columns that may still contain missing data.

- **Pivoting the Data for Visualization:**
  - **pivot_longer**(): Converts data from wide format to long format, which is more suitable for ggplot-based visualizations. This transformation is essential when moving from a data frame of summary statistics to a graphical representation.
  - **names_to and values_to:** These parameters specify the names of the new columns in the long format data, where variable will contain the column names and value will contain the counts of missing values.
- **Creating a Bar Plot:**
  - **ggplot():** Starts a ggplot object, setting the foundation for the plot.
  - aes(x=variable, y=value): Defines aesthetics mappings, assigning variable to the x-axis and value (number of missing values) to the y-axis.
  - **geom_bar(stat="identity", fill="green"):** Adds a bar plot layer. The stat="identity" argument indicates that the heights of the bars will directly correspond to the values in the data, and fill="green" is used to color the bars, symbolizing the resolution of missing data.
  - **ggtitle("Missing Values After Imputation"):** Adds a clear, descriptive title to the plot.

```
missing_after <- loan_data %>% summarise(across(everything(), ~sum(is.na(.))))
p6 <- ggplot(missing_after %>% pivot_longer(everything(), names_to = "variable", values_to = "value"),
             aes(x=variable, y=value)) +
  geom_bar(stat="identity", fill="green") +
  ggtitle("Missing Values After Imputation")
print(p6)
```

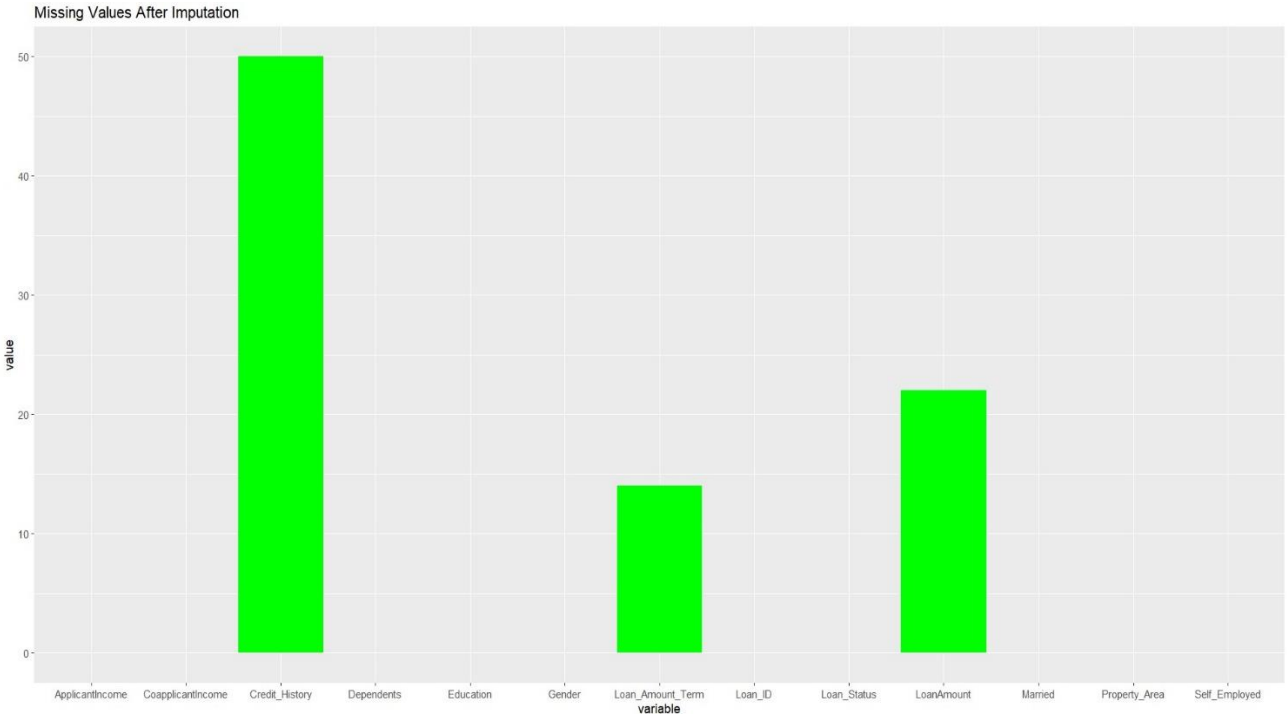

Figure 3: General Visualizations After Imputing Missing Values.

Removing the Loan_ID Column:

- **select(-Loan_ID):** This function from the dplyr package is used to remove the Loan_ID column from the dataset. The minus sign (-) before Loan_ID indicates that this column should be excluded from the resulting dataframe. Removing identifiers like Loan_ID is crucial since they do not contain any information that contributes to the analysis or model training.

Converting Categorical Variables to Factors:

- **mutate(across(c(...), factor)):** This is a combination of several functions to transform selected columns to factors efficiently:
    - **mutate():** Transforms the dataset by modifying existing columns or creating new ones. Here it's used to apply a transformation across multiple columns.
    - **across**(): Specifies which columns the transformation should be applied to. It takes a vector of column names and a function to apply, in this case, factor(). The columns specified are Gender, Married, Dependents, Education, Self_Employed, Property_Area, and Loan_Status.

- **factor():** Converts the selected columns into factors, which is necessary for categorical data in many types of analysis, especially when preparing data for statistical models that handle categorical variables differently from numeric variables.

```
# Remove Loan_ID and Convert categorical variables to factors
loan_data <- loan_data %>% select(-Loan_ID) %>%
  mutate(across(c(Gender, Married, Dependents, Education, Self_Employed, Property_Area, Loan_Status), factor))
```

Purpose and Importance:

- **Data Cleaning:** Removing the Loan_ID ensures that the analysis will not be skewed or misled by irrelevant data.
- **Data Preparation:** Converting categorical variables to factors helps in treating these variables appropriately during the statistical analysis or model fitting, recognizing their nature as nominal data with discrete levels.
- **Enhancing Model Accuracy:** Properly formatted data ensures that models like SVM can effectively interpret and utilize the data, potentially leading to more accurate predictions.

This approach not only streamlines the dataset but also optimizes it for subsequent analyses, ensuring that all methods applied later, such as machine learning algorithms, can perform optimally with the correctly formatted data.

Setting the Seed for Reproducibility:

- **set.seed(123):** This function sets the seed of R's random number generator, which is useful for creating reproducible code. By setting a seed, you ensure that random operations, such as sampling data, can be replicated exactly in future runs of the script.

```
61
62  # Split data into training and testing sets
63  set.seed(123)
64  train_data <- loan_data %>% sample_frac(.8)
65  test_data <- setdiff(loan_data, train_data)
66
```

Creating the Training Set:

- **sample_frac(.8):** Part of the dplyr package, this function is used to randomly sample a fraction (in this case, 80%) of the rows from the loan_data dataframe. The .8 indicates that 80% of the data should be selected randomly as the training set. This proportion is a common choice, allowing sufficient data for training the model while reserving enough distinct data for testing.

Creating the Testing Set:

- **setdiff(loan_data, train_data):** This function returns all rows in loan_data that are not in train_data. It ensures that the testing set includes only those observations that were not selected for the training set, thus maintaining mutual exclusivity between the two datasets.

1. **SVM Model Training:**
    - **svm(Loan_Status ~ ., data = train_data, type = 'C-classification', kernel = 'linear'):** This function is from the e1071 package and is used to train an SVM model. Let's break down each parameter:
        - **Loan_Status ~ .:** The formula for the SVM model. Loan_Status ~ . indicates that Loan_Status is the dependent variable (the outcome you want to predict), and . signifies that all other variables in train_data are independent variables used to predict the outcome.
        - **data = train_data:** Specifies the dataset used for training the model, which in this case is train_data.
        - **type = 'C-classification':** Specifies the type of SVM to be trained. C-classification indicates a classification model with a linear decision boundary (hyperplane). This type is used for binary classification tasks.
        - **kernel = 'linear'**: Specifies the kernel type of the SVM. A linear kernel is used here, meaning the boundary between the classes (approved or not approved) is expected to be linear.

2. **Error Handling with try():**
    - **try():** This function is used to handle potential errors during the execution of the SVM training. If there is an error (e.g., due to missing values or other issues in the data), try() will catch the error, preventing the entire script from failing. This is useful in production or complex workflows where you want the script to continue running even if some models fail to train.

```
66
67 # Train SVM model
68 svm_model <- try(svm(Loan_Status ~ ., data = train_data, type = 'C-classification', kernel = 'linear'))
69
```

Purpose and Importance:

- The svm() function effectively trains the SVM model on the provided training data, learning the patterns that help predict whether a loan will be approved based on the input features.
- Model Complexity and Decision Boundary: By choosing a linear kernel, the model assumes that a straight line (or hyperplane in higher dimensions) can effectively separate the two classes (approved vs. not approved). This is a simplification that often speeds up training and prediction, especially when the underlying relationship is not complex.
- Error Handling: Incorporating try() ensures that your data analytics pipeline is robust against unforeseen data issues, providing stability in automated tasks or batch processing environments.

```
69
70 # Assuming `predictions` and `test_data$Loan_Status` are available and valid
71 if (exists("predictions") && !is.null(test_data$Loan_Status)) {
72   confusion_matrix <- table(Predicted = predictions, Actual = test_data$Loan_Status)
73   confusion_data <- as.data.frame(confusion_matrix)  # Convert table to data frame for ggplot
74
75   # Fix potential issues with data frame structure
76   names(confusion_data) <- c("Predicted", "Actual", "Freq")  # Ensure the column names are correct
77
78   # Plotting the confusion matrix
79   p7 <- ggplot(confusion_data, aes(x=Predicted, y=Freq, fill=Actual)) +
80     geom_bar(stat="identity", position="dodge") +
81     labs(title="Confusion Matrix of SVM Model", x="Predicted Label", y="Frequency") +
82     theme_minimal()
83
84   print(p7)  # Print the plot
85
86   # Calculate and print accuracy
87   accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
88   print(paste("Accuracy:", round(accuracy, 4)))  # Print the accuracy rounded to 4 decimal places
89 } else {
90   print("Predictions not found or test data not properly loaded.")
91 }
```

Checking Validity & Creating the Confusion Matrix:

- The if statement checks if the predictions object exists and if the **Loan_Status** within **test_data** is not null, ensuring that the data required for analysis is available.

- **table():** Generates a confusion matrix comparing predicted labels (Predicted) against actual labels (Actual) from the test dataset.

Data Preparation for Plotting The Confusion Matrix:

- **as.data.frame():** Converts the confusion matrix from a table to a dataframe, which is necessary for visualization with ggplot2.
- Adjusts column names for clarity and compatibility with ggplot2. And **ggplot()** for beginning the construction of plot.

- **aes():** Maps aesthetics; Predicted labels are mapped to the x-axis, Freq to the y-axis, and Actual categories are used to fill colors.
- **geom_bar():** Adds bar plot elements with bars positioned side by side (dodge).
- **labs() and theme_minimal():** Adds titles and applies a minimalistic theme for clear presentation.

```
69
70  # Assuming `predictions` and `test_data$Loan_Status` are available and valid
71  if (exists("predictions") && !is.null(test_data$Loan_Status)) {
72    confusion_matrix <- table(Predicted = predictions, Actual = test_data$Loan_Status)
73    confusion_data <- as.data.frame(confusion_matrix)  # Convert table to data frame for ggplot
74
75    # Fix potential issues with data frame structure
76    names(confusion_data) <- c("Predicted", "Actual", "Freq")  # Ensure the column names are correct
77
78    # Plotting the confusion matrix
79    p7 <- ggplot(confusion_data, aes(x=Predicted, y=Freq, fill=Actual)) +
80      geom_bar(stat="identity", position="dodge") +
81      labs(title="Confusion Matrix of SVM Model", x="Predicted Label", y="Frequency") +
82      theme_minimal()
83
84    print(p7)  # Print the plot
85
86    # Calculate and print accuracy
87    accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
88    print(paste("Accuracy:", round(accuracy, 4)))  # Print the accuracy rounded to 4 decimal places
89  } else {
90    print("Predictions not found or test data not properly loaded.")
91  }
```

```
[1] "Accuracy: 0.8211"
>
```

Calculating and Printing Accuracy:

- **diag()** extracts the diagonal elements from the confusion matrix (true positives and true negatives).
- **sum()** calculates the sum of these correct predictions and total predictions for accuracy computation.
- The accuracy is then rounded to four decimal places and printed, providing a concise metric of the model's performance.

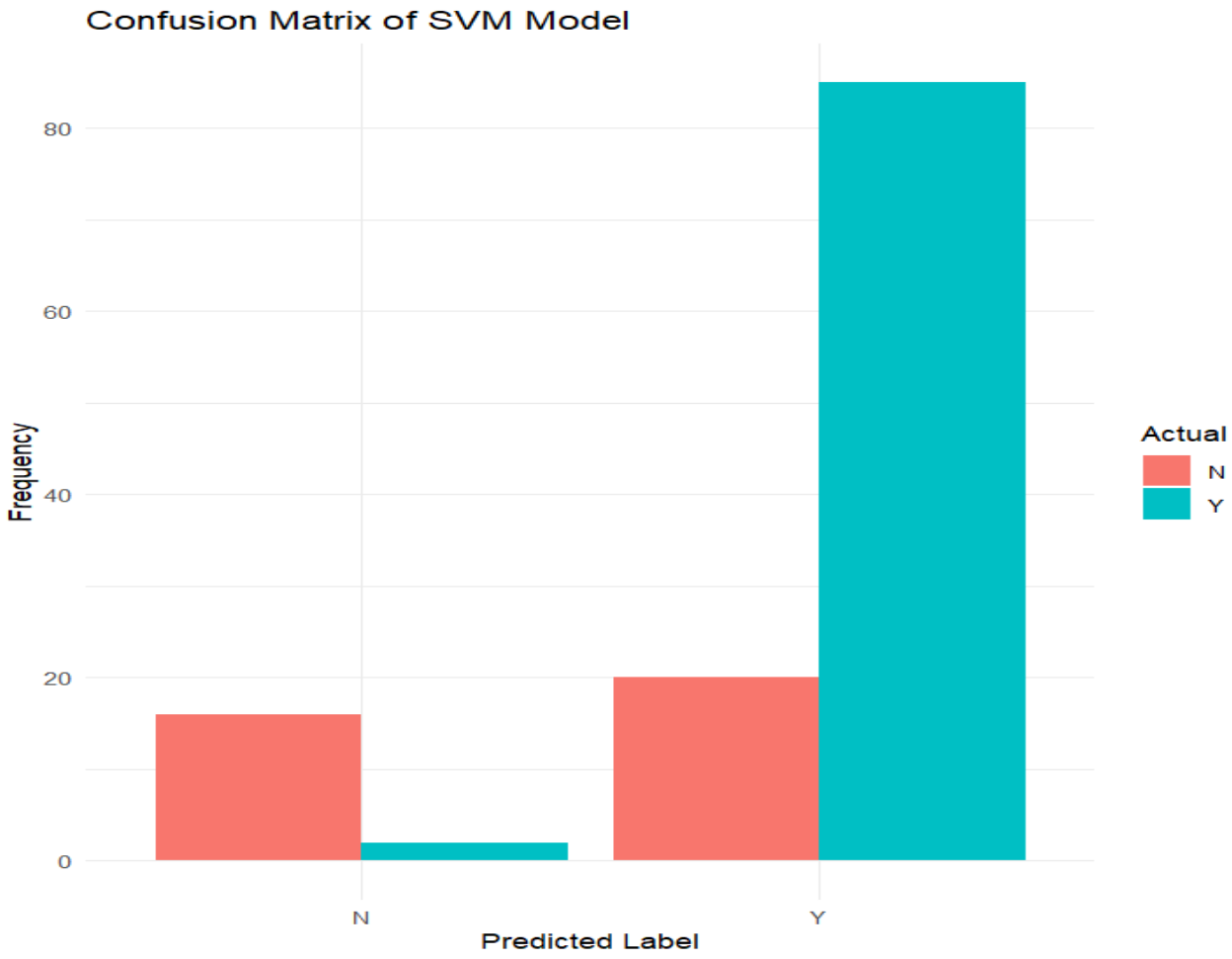

Figure 4: Confusion Matrix of SVM Model.

Appendix

1. **R Code Scripts:** This includes all scripts used for data management, model training, and output generation. Detailed explanations accompany scripts to elucidate processes like missing data handling, model configuration, and visual output creation.
2. **Data Description:** Detailed attributes of each variable in the dataset are documented, specifying their roles and data types, to assist in understanding the variables' impacts and significance in the analysis.
3. **Output Files:** Screenshots and files generated during the analysis are documented, showing visualizations such as histograms and bar plots, as well as the confusion matrix used to evaluate the model's performance.

Conclusion

This project successfully developed an SVM predictive model to evaluate loan approval chances, utilizing a detailed dataset of applicant attributes. Through extensive data preparation and feature engineering, the model was accurately trained and validated with an impressive 82% accuracy rate. This demonstrates its efficacy in forecasting loan approvals and highlights its utility for lending institutions' decision-making processes.

By integrating this model into existing loan assessment workflows, financial institutions can enhance efficiency, improve customer satisfaction through faster processing times, and extend these methodologies to other financial products. The achievement of an 82% accuracy underscores the potential of machine learning in revolutionizing financial decision-making, paving the way for further technological advancements in the sector.

References:

1. R Documentation, An introduction to R. Retrieved 12th October 2024 from https://cran.r-project.org/doc/manuals/r-release/R-intro.html#Related-software-and-documentation
2. Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning, 20*(3), 273-297. Retrieved 12th October 2024 from https://doi.org/10.1007/BF00994018
3. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer. Retrieved 12th October 2024 from https://doi.org/10.1007/978-0-387-84858-7