Predicting Credit Card Approval

Shih-Yu Huang

Contents

Introduction

Credit card is one of the most popular payment methods nowadays, and there are 2.8 billion credit cards in use worldwide as of 2021 (Shift Processing, 2021). Traditionally, banks receive multiple credit card applications and manually review applicants' materials to evaluate whether to approve the application or not. During the process, banks should consider several factors that can analyse customers' credibility, such as income stability, credit score, debt, etc., and generate a credit report. However, it could be quite mundane, and time-consuming.

This project focuses on the credit card application process and aims to build an automatic credit card approval predictor, improving operation efficiency. In this project, we select four algorithms (Support Vector Machine, Decision Tree, Random Forest, and Logistic Regression) to explore the pattern of data and predict the approval. The outcome is expected to maximise the possibility of successful approval and mitigate the risk of credit card default.

Data

The data set used in this project is Credit Approval Data Set provided by UCI Machine Learning Repository and concerns credit card applications with multiple relevant attributes. Due to the confidentiality of the data, all attributes are anonymised. Hence, the data dictionary is provided to look up the description of each attribute. In this project, **V16** is chosen as the target variable to predict the approval class.

Attributes	Description
V1	Gender
V2	Age
V3	Debt
V4	Marital status
V5	Bank customer
V6	Education level
V7	Ethnicity
V8	Employment seniority
V9	Prior default
V10	Employment status
V11	Credit score
V12	Driver license
V13	Citizen
V14	Zip code
V15	Income
V16	Approval

Method

In this section, we will demonstrate how the business problems can be solved with the power of data and modelling techniques.

I. Data Understanding

In the **Data Understanding** phase, we aim to understand the strengths and limitations of data and outline the data requirements for the following actions.

To begin with, we import the data set with `read.csv()` and check the summary to see whether there are missing values or redundant attributes that we can remove.

```
# Read the dataset
cc_approvals <- read.csv("cc_approvals.csv", header = FALSE, stringsAsFactors = TRUE)

# Summary
summary(cc_approvals)
```

```
##  V1           V2            V3          V4       V5           V6           V7
##  ?: 12   ?      : 12   Min.   : 0.000   ?:  6   ? :  6   c      :137   v      :399
##  a:210   22.67  :  9   1st Qu.: 1.000   l:  2   g :519   q      : 78   h      :138
##  b:468   20.42  :  7   Median : 2.750   u:519   gg:  2   w      : 64   bb     : 59
##          18.83  :  6   Mean   : 4.759   y:163   p :163   i      : 59   ff     : 57
##          19.17  :  6   3rd Qu.: 7.207                    aa     : 54   ?      :  9
##          20.67  :  6   Max.   :28.000                    ff     : 53   j      :  8
##          (Other):644                                     (Other):245   (Other): 20
##        V8         V9      V10          V11          V12     V13          V14           V15
##  Min.   : 0.000   f:329   f:395   Min.   : 0.0   f:374   g:625   00000  :132   Min.   :     0.0
```

```
##  1st Qu.: 0.165   t:361   t:295   1st Qu.: 0.0   t:316   p:  8   00120  : 35   1st Qu.:     0.0
##  Median : 1.000                   Median : 0.0           s: 57   00200  : 35   Median :     5.0
##  Mean   : 2.223                   Mean   : 2.4                   00160  : 34   Mean   :  1017.4
##  3rd Qu.: 2.625                   3rd Qu.: 3.0                   00080  : 30   3rd Qu.:   395.5
##  Max.   :28.500                   Max.   :67.0                   00100  : 30   Max.   :100000.0
##                                                                  (Other):394
##  V16
##  -:383
##  +:307
##
##
##
##
##
```

Meanwhile, we check the structure of the data set to identify the issues of data types.

```
# Check the structure
str(cc_approvals)
```

```
## 'data.frame':    690 obs. of  16 variables:
##  $ V1 : Factor w/ 3 levels "?","a","b": 3 2 2 3 3 3 3 2 3 3 ...
##  $ V2 : Factor w/ 350 levels "?","13.75","15.17",..: 158 330 91 127 45 170 181 76 312 257 ...
##  $ V3 : num  0 4.46 0.5 1.54 5.62 ...
##  $ V4 : Factor w/ 4 levels "?","l","u","y": 3 3 3 3 3 3 3 3 4 4 ...
##  $ V5 : Factor w/ 4 levels "?","g","gg","p": 2 2 2 2 2 2 2 2 4 4 ...
##  $ V6 : Factor w/ 15 levels "?","aa","c","cc",..: 14 12 12 14 14 11 13 4 10 14 ...
##  $ V7 : Factor w/ 10 levels "?","bb","dd",..: 9 5 5 9 9 9 5 9 5 9 ...
##  $ V8 : num  1.25 3.04 1.5 3.75 1.71 ...
##  $ V9 : Factor w/ 2 levels "f","t": 2 2 2 2 2 2 2 2 2 2 ...
##  $ V10: Factor w/ 2 levels "f","t": 2 2 1 2 1 1 1 1 1 1 ...
##  $ V11: int  1 6 0 5 0 0 0 0 0 0 ...
##  $ V12: Factor w/ 2 levels "f","t": 1 1 1 2 1 2 2 1 1 2 ...
##  $ V13: Factor w/ 3 levels "g","p","s": 1 1 1 1 3 1 1 1 1 1 ...
##  $ V14: Factor w/ 171 levels "?","00000","00017",..: 70 13 98 33 39 117 56 25 64 17 ...
##  $ V15: int  0 560 824 3 0 0 31285 1349 314 1442 ...
##  $ V16: Factor w/ 2 levels "-","+": 2 2 2 2 2 2 2 2 2 2 ...
```

II. Data Preparation

In the **Data Preparation** phase, we plan to transform the data into a form that is ready for modelling and can yield a better result, and obtain the final data set from the initial data.

Data Cleaning

Firstly, some data values are required to be altered to certain values. We spot that the **V9**, **V10**, **V12** columns are binary results with `True` and `False` value. We also see the same situation in the target column **V16**, with `+` and `-`. Hence, we need to change these values into `1` or `0` to streamline data and maintain consistency. Furthermore, we can see missing values are represented by `?`, then we transform them to a standard string `N/A`.

```
# Change "-" and "+" to binary value
cc_approvals$V16 <- as.numeric(cc_approvals$V16 == '+')
cc_approvals$V9 <- as.numeric(cc_approvals$V9 == 't')
cc_approvals$V10 <- as.numeric(cc_approvals$V10 == 't')
cc_approvals$V12 <- as.numeric(cc_approvals$V12 == 't')
```

```
# Change question mark (?) into N/A
na_string <- cc_approvals == "?"
is.na(cc_approvals) <- na_string
```

After adjusting missing value to the appropriate value, we can clean the data set and prevent data noise. We use `na.omit()` to drop all missing values. From the data set summary, we can recognise the **V13** and **V14** columns are redundant since they fail to provide meaningful information or demonstrate difference.

```
# Remove missing value
cc_approvals <- na.omit(cc_approvals)

# Drop the citizen (V13) and zip code (V14) columns
cc_approvals$V13 <- NULL
cc_approvals$V14 <- NULL
```

Finally, we make sure that all data are classified into appropriate types. We change **V2** into numeric variable and all binary columns (**V9**, **V10**, **V12** and **V16**) into factor variables.

```
#Change data type
cc_approvals$V2 <- as.numeric(cc_approvals$V2)

columns <- c("V9", "V10", "V12", "V16")
cc_approvals[columns] <- lapply(cc_approvals[columns], as.factor)
```

Then, we check the structure again to see if the data is ready for next step.

```
# Check the structure
str(cc_approvals)
```

```
## 'data.frame':    653 obs. of  14 variables:
##  $ V1 : Factor w/ 3 levels "?","a","b": 3 2 2 3 3 3 3 2 3 3 ...
##  $ V2 : num  158 330 91 127 45 170 181 76 312 257 ...
##  $ V3 : num  0 4.46 0.5 1.54 5.62 ...
##  $ V4 : Factor w/ 4 levels "?","l","u","y": 3 3 3 3 3 3 3 3 4 4 ...
##  $ V5 : Factor w/ 4 levels "?","g","gg","p": 2 2 2 2 2 2 2 2 4 4 ...
##  $ V6 : Factor w/ 15 levels "?","aa","c","cc",..: 14 12 12 14 14 11 13 4 10 14 ...
##  $ V7 : Factor w/ 10 levels "?","bb","dd",..: 9 5 5 9 9 9 5 9 5 9 ...
##  $ V8 : num  1.25 3.04 1.5 3.75 1.71 ...
##  $ V9 : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
##  $ V10: Factor w/ 2 levels "0","1": 2 2 1 2 1 1 1 1 1 1 ...
##  $ V11: int  1 6 0 5 0 0 0 0 0 0 ...
##  $ V12: Factor w/ 2 levels "0","1": 1 1 1 2 1 2 2 1 1 2 ...
##  $ V15: int  0 560 824 3 0 0 31285 1349 314 1442 ...
##  $ V16: Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
##  - attr(*, "na.action")= 'omit' Named int [1:37] 72 84 87 93 98 203 207 244 249 255 ...
##   ..- attr(*, "names")= chr [1:37] "72" "84" "87" "93" ...
```

Data Split

Now, the data is ready for modelling. Before the next phase, we have to split data into training set and test set. We set the seed to ensure the same random value and then generate a split vector with `sample.split()` to partition the data into training and test sets with a training ratio of 0.7.

```
set.seed(123)

# Split data
```

```
split = sample.split(cc_approvals$V16, SplitRatio = 0.7)

# Generate the training and test sets by subsetting the data records from original dataset
training_set = subset(cc_approvals, split == TRUE)

test_set = subset(cc_approvals, split == FALSE)
```

Next, we check the ratio of the target variable **V16** and assess whether data balancing is required to ensure the modelling result will not be biased. It turns out that the the data is balanced and, ratio adjustment is not required.

```
# Check the distribution of feature approval
table(training_set$V16)
```

```
##
##   0   1
## 250 207
```

Feature Selection

To increase modelling accuracy and decreases the size of the data, information gain is calculated to determine which attributes are the most informative for the modelling. We obtain the information gain of all attributes with `information.gain()` and use function `cutoff.k()` to pick the top 6 most informative attributes and assign them to a new subset.

```
# Use function information.gain to compute information gain values of the attributes
attribute_weights <- information.gain(V16~., training_set)

# Save a copy of the weights
df <- attribute_weights

# Add row names as a column to keep them during ordering
df$attr <- rownames(df)

# Sort the weights in decreasing order of information gain values
df <- arrange(df, -attr_importance)
```

```
#Find the most informative attributes
filtered_attributes <- cutoff.k(attribute_weights, 6)

# Select a subset of the dataset by using filtered_attributes
modelling <- training_set[filtered_attributes]

modelling$V16 <- training_set$V16

# Preview the data set
head(modelling)
```

```
##    V9 V11 V10   V15 V6    V8 V16
## 1   1   1   1     0  w 1.250   1
## 3   1   0   0   824  q 1.500   1
## 6   1   0   0     0  m 2.500   1
## 7   1   0   0 31285  r 6.500   1
## 9   1   0   0   314  k 3.960   1
## 10  1   0   0  1442  w 3.165   1
```

III. Modelling

In the **Modelling** phase, we select four modelling techniques based on the project objective, including support vector machine (SVM), decision tree, random forest and logistic regression and then display test results.

Support Vector Machine

Load the package `e1071` and use the function `svm()` to build an SVM model with the data set `modelling`. We set the `kernel = "radial"`, `scale = TRUE`, and `probability = TRUE`, and then we are ready to predict the approval class. Plus, we save the result of the prediction and create a new column for the model.

```
# Build a SVM model
model_SVM  <- svm(V16 ~ ., data = modelling, kernel = "radial", scale = TRUE, probability = TRUE)

# Prediction
SVM_predict = predict(model_SVM, test_set)

# Save results
results <- test_set

# Create a column named PredictionSVM
results$PredictionSVM <- SVM_predict
```

Decision Tree

Load the package `rpart` and use the function `rpart()` to build a decision tree model with the data set `modelling`. We set the `type = "class"` in the function `predict()` and save the result.

```
# Build a decision tree
model_DT <- rpart(V16 ~ ., data = modelling)

# Prediction
DT_predict = predict(model_DT, test_set, type= "class")

# Create a column named PredictionTree
results$PredictionTree <- DT_predict
```

Random Forest

Load the package `randomForest` and use the function `randomForest()` to build a random forest model with the data set `modelling`.

```
# Build a RF model
model_RF <- randomForest(V16~., modelling)

# Prediction
RF_predict <- predict(model_RF, test_set)

# Create a column named PredictionRF
results$PredictionRF <- RF_predict
```

Logistics Regression

Use the function `glm()` to build a logistic regression model with the data set `modelling` and set `family = "binomial"`. To predict the class of test data, we use a threshold value of 0.5 to decide whether the probability of a record should be marked as `1` or `0` and then save them as factor variables.

```
# Build a Logistic Regression model
model_LogReg <- glm(V16~. , data = modelling, family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
# Prediction
LogReg_predict <- predict(model_LogReg, test_set, type="response")

# Predict the class
LOGREG_class <- ifelse(LogReg_predict > 0.5, 1, 0)

# Save the predictions as factor variables
LOGREG_class <- as.factor(LOGREG_class)

# Create a column named PredictionLogReg
results$PredictionLogReg <- LOGREG_class
```

Test Result Preview

```
##    V16 PredictionSVM PredictionTree PredictionRF PredictionLogReg
## 2    1             1              1            1                1
## 4    1             1              1            1                1
## 5    1             1              1            1                1
## 8    1             1              1            1                1
## 11   1             0              0            0                0
## 13   1             1              0            0                0
```

VI. Evaluation

In the **Evaluation** phase, we select two evaluation tools to help us find out which prediction model is more suitable and determine if it is good enough for deployment.

ROC Curve

Draw the receiver operating characteristic curve (ROC) to compare the performance of four prediction models on the test set and efficiently get a big picture. (See more details in the **Analysis** Section)

```
# Obtain class probabilities for SVM
SVM_pred <- predict(model_SVM, test_set, probability = TRUE)
SVM_prob <- attr(SVM_pred, "probabilities")

# Obtain class probabilities for Decision Tree
DT_prob <- predict(model_DT, test_set, type = "prob") %>%
  bind_cols(test_set)

# Obtain class probabilities for Random Forest
RF_prob <- predict(model_RF, test_set, type = "prob")
```

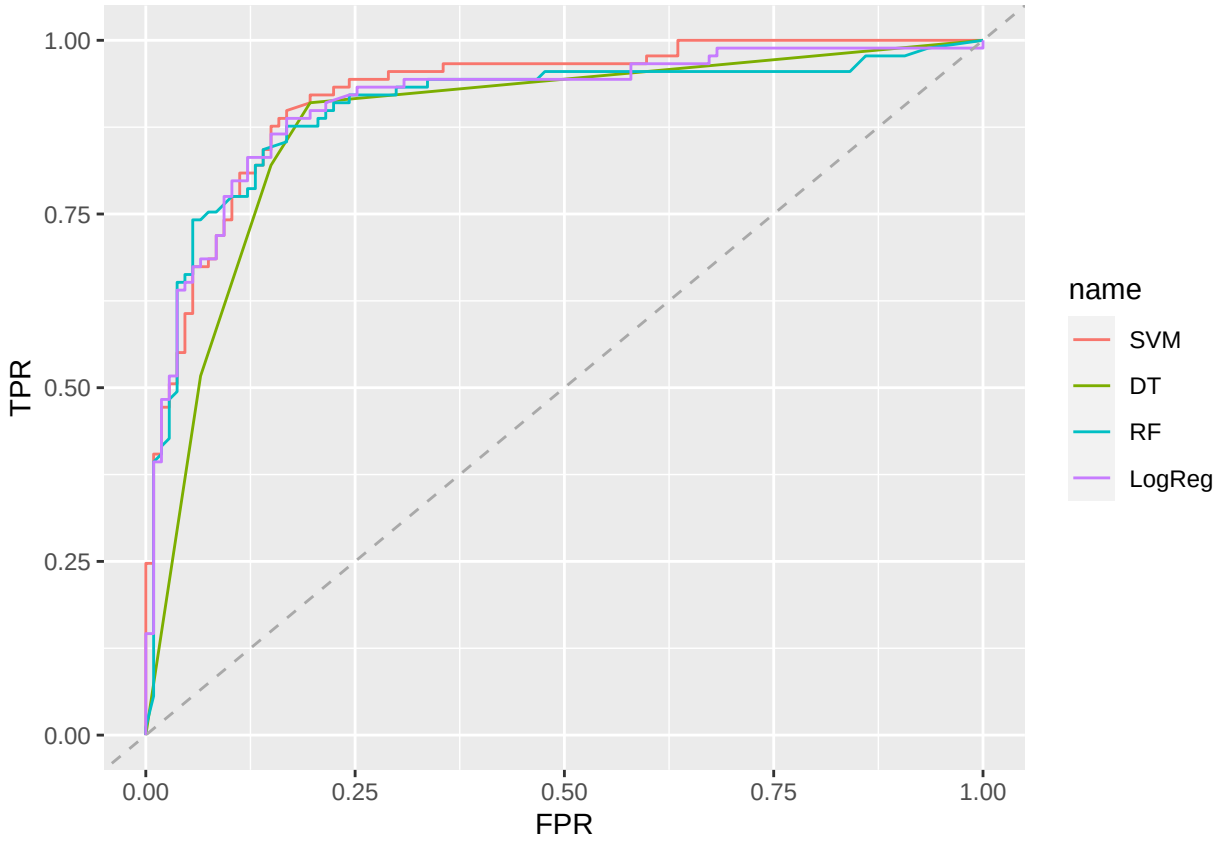
```
# ROC curve of SVM
ROC_SVM <- roc(test_set$V16, SVM_prob[, 2])

# ROC curve of Decision Tree
ROC_DT <- roc(test_set$V16, DT_prob[, 2])

# ROC curve of Random Forest
ROC_RF <- roc(test_set$V16, RF_prob[, 2])
```

```r
# ROC curve of Logistic Regression
ROC_LogReg <- roc(test_set$V16, LogReg_predict)
```

```r
# Plot the ROC curve for SVM, Decision Tree, Logistic Regression, and Random Forest.
ggroc(list(SVM = ROC_SVM, DT = ROC_DT, RF = ROC_RF, LogReg = ROC_LogReg), legacy.axes=TRUE) + xlab("FPR
   geom_abline(intercept = 0, slope = 1, color = "darkgrey", linetype = "dashed")
```



Confusion Matrix

Create confusion matrices to assess the performance of four prediction models by looking into more index to evaluate which model is more suitable. (See more details in the **Analysis** Section)

```r
# SVM
confusionMatrix(SVM_predict, test_set$V16, positive = "1", mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 86  7
##          1 21 82
##
##                Accuracy : 0.8571
##                  95% CI : (0.8002, 0.9029)
##     No Information Rate : 0.5459
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.7156
```

```
##
##  Mcnemar's Test P-Value : 0.01402
##
##               Precision : 0.7961
##                  Recall : 0.9213
##                      F1 : 0.8542
##              Prevalence : 0.4541
##          Detection Rate : 0.4184
##    Detection Prevalence : 0.5255
##       Balanced Accuracy : 0.8625
##
##        'Positive' Class : 1
##
```

```
#Decision Tree
confusionMatrix(DT_predict, test_set$V16, positive = "1", mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 91 16
##          1 16 73
##
##                Accuracy : 0.8367
##                  95% CI : (0.7774, 0.8856)
##     No Information Rate : 0.5459
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6707
##
##  Mcnemar's Test P-Value : 1
##
##               Precision : 0.8202
##                  Recall : 0.8202
##                      F1 : 0.8202
##              Prevalence : 0.4541
##          Detection Rate : 0.3724
##    Detection Prevalence : 0.4541
##       Balanced Accuracy : 0.8353
##
##        'Positive' Class : 1
##
```

```
#Random Forest
confusionMatrix(RF_predict, test_set$V16, positive = "1", mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 93 16
##          1 14 73
##
##                Accuracy : 0.8469
##                  95% CI : (0.7888, 0.8943)
```

```
##     No Information Rate : 0.5459
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6907
##
##  Mcnemar's Test P-Value : 0.8551
##
##               Precision : 0.8391
##                  Recall : 0.8202
##                      F1 : 0.8295
##              Prevalence : 0.4541
##          Detection Rate : 0.3724
##    Detection Prevalence : 0.4439
##       Balanced Accuracy : 0.8447
##
##        'Positive' Class : 1
##
```

```
#Logistics Regression
confusionMatrix(LOGREG_class, test_set$V16, positive = "1", mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 93 15
##          1 14 74
##
##                Accuracy : 0.852
##                  95% CI : (0.7945, 0.8986)
##     No Information Rate : 0.5459
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7013
##
##  Mcnemar's Test P-Value : 1
##
##               Precision : 0.8409
##                  Recall : 0.8315
##                      F1 : 0.8362
##              Prevalence : 0.4541
##          Detection Rate : 0.3776
##    Detection Prevalence : 0.4490
##       Balanced Accuracy : 0.8503
##
##        'Positive' Class : 1
##
```

Analysis

In this project, the ultimate goal is to automate the application review process and predict the approval to avoid issuing credit cards to potentially default customers with an automated prediction system. In this section, we will generate insights from the modelling process and evaluate modelling performance for the final decision.

Given the outcome of feature selection, we can rank the factors involved in the credit card application from the most to the least informative. In figure 1, we can see the attribute **V9** (Prior Default) is the most important factor contributing to credit card approvals. The **V11** (Credit Score) and **V10** (Employment Status) also exert effects on it. Overall, we can efficiently identify key factors for banks to approach the task and solve business problems.

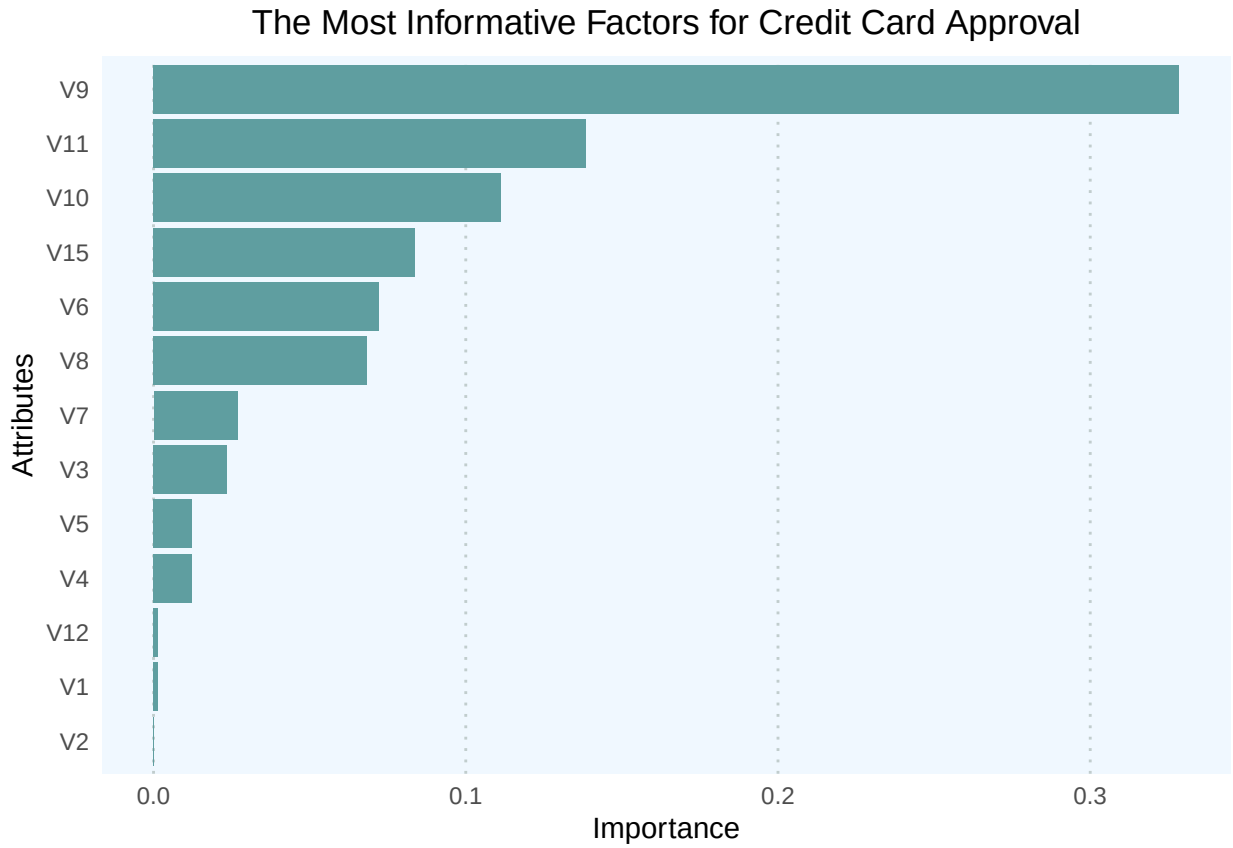


Figure 1: Figure 1. The Most Informative Factors for Credit Card Approval.

After that, we have to analyse the performance of each modelling technique. Firstly, we expect to identify the most suitable model by looking into the ROC curves and searching for the closest curve to the top-left corner. However, these four algorithms present similar profiles (Figure 2.) and, thereby, we cannot tell the clear difference in overall performance.

Next, we conduct a deeper analysis with Confusion Matrix to compare different indicators based on the project objective. Table 1 indicates that the SVM modelling technique delivers better performance than others. To ensure the outcome will align with the project goal, we need to examine the error rate further from a statistical viewpoint. Given that banks want to avoid the risk of default caused by misidentification, we can look into the percentage (Type I error rate) of incorrectly predicted approvals among non-approval cases, which will pose default risk at banks. From table 2, we can notice the SVM algorithm has the highest error rate instead, while the Random Forest and Logistic Regression algorithms give a better performance.

Table 2: Table 1. Confusion Matrix of Each Model

Indicators	SVM	Decision Tree	Random Forest	Logistic Regression
Accuracy	**85.71%**	83.67%	84.69%	85.2%
Recall	**92.13%**	82.02%	82.02%	83.15%
Precision	79.61%	82.02%	83.91%	**84.09%**

Indicators	SVM	Decision Tree	Random Forest	Logistic Regression
F1	**85.42%**	82.02%	82.95%	83.62%%

Table 3: Table 2. Error Rate of Non-Approval as Approval.

SVM	Decision Tree	Random Forest	Logistic Regression
19.63%	14.95%	**13.08%**	**13.08%**

Conclusion

Overall, I will recommend deploying the **Logistic Regression** model in the banking system. The project objective is to prevent default risk during the credit card approval process. Although the SVM model performs better in multiple indicators, we cannot confirm it is the best due to the high error rate. By contrast, the Logistic Regression model can deliver similar performance as well and has the lowest error rate, minimising the risk. Therefore, I believe that the **Logistic Regression** model is a more appropriate alternative tailored for this project.

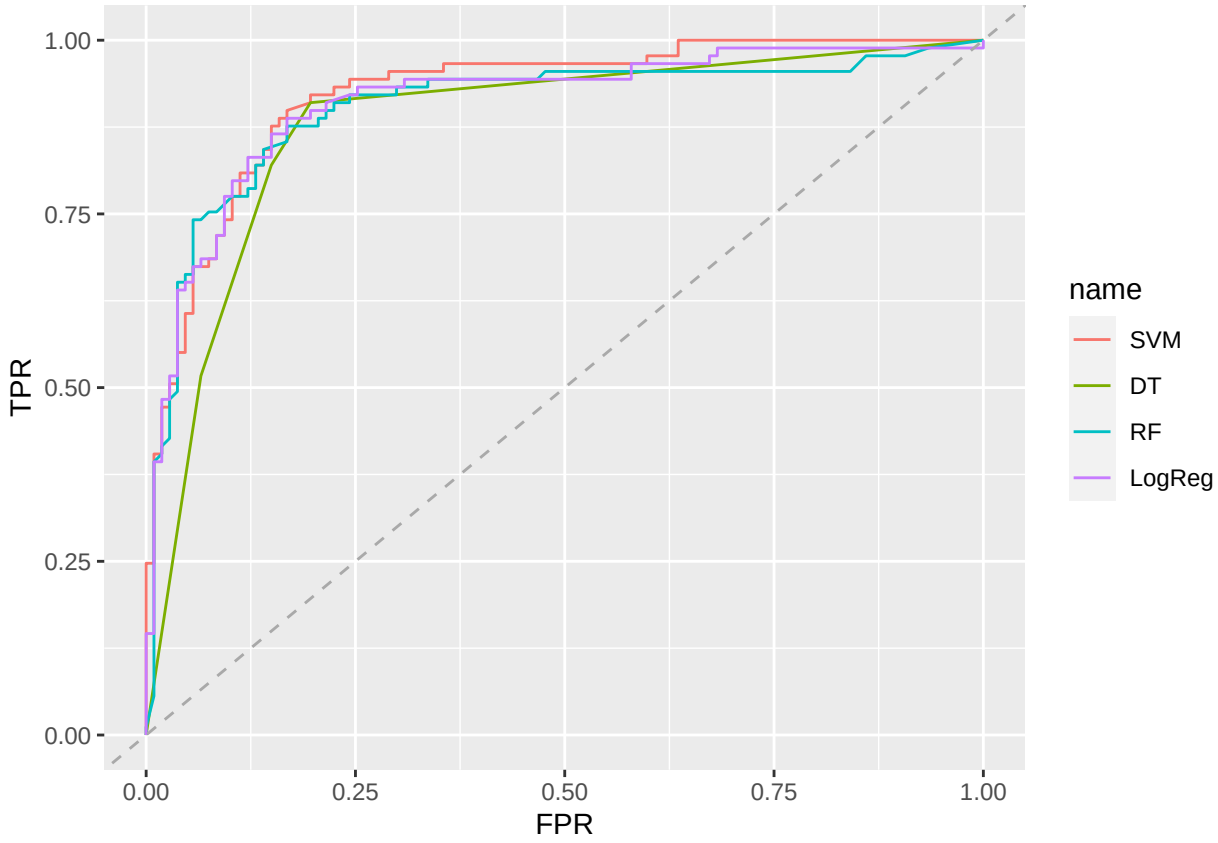


Figure 2: Figure 2. The ROC Curve of Each Model.