Credit Card Fraud Detection

Usama

2024-10-21

Introduction

In this project, I aim to build machine learning models that detect fraudulent credit card transactions using a dataset of transactions made by European cardholders. The dataset is highly imbalanced, with a very small percentage of transactions being fraudulent. I will apply **Logistic Regression** and **Decision Tree** models to classify the transactions and evaluate the models' performance using various metrics such as accuracy, precision, recall, and ROC-AUC.

Methods

Data Loading and Exploration

I first load and explore the dataset. I examine the class distribution to highlight the class imbalance problem and check for missing values.

```
# Load required libraries
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## Warning: package 'tidyverse' was built under R version 4.3.2
```

```
## Warning: package 'ggplot2' was built under R version 4.3.2
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
if(!require(ROSE)) install.packages("ROSE", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: ROSE
```

```
## Warning: package 'ROSE' was built under R version 4.3.3
```

```
## Loaded ROSE 0.0-4
```

```
if(!require(pROC)) install.packages("pROC", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: pROC
```

```
## Warning: package 'pROC' was built under R version 4.3.3
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
if(!require(rpart)) install.packages("rpart", repos = "http://cran.us.r-project.org") # For decision tr
```

```
## Loading required package: rpart
```

```
library(tidyverse)
library(caret)
library(ROSE)
library(pROC)
library(rpart)
# Load the dataset and inspect its structure
credit_data <- read.csv("C:/Users/usama/Downloads/Credit Card Fraud Detection/creditcard.csv")
str(credit_data)
```

```
## 'data.frame':    284807 obs. of  31 variables:
##  $ Time  : num  0 0 1 1 2 2 4 7 7 9 ...
##  $ V1    : num  -1.36 1.192 -1.358 -0.966 -1.158 ...
##  $ V2    : num  -0.0728 0.2662 -1.3402 -0.1852 0.8777 ...
##  $ V3    : num  2.536 0.166 1.773 1.793 1.549 ...
```

```
##  $ V4    : num  1.378 0.448 0.38 -0.863 0.403 ...
##  $ V5    : num  -0.3383 0.06 -0.5032 -0.0103 -0.4072 ...
##  $ V6    : num  0.4624 -0.0824 1.8005 1.2472 0.0959 ...
##  $ V7    : num  0.2396 -0.0788 0.7915 0.2376 0.5929 ...
##  $ V8    : num  0.0987 0.0851 0.2477 0.3774 -0.2705 ...
##  $ V9    : num  0.364 -0.255 -1.515 -1.387 0.818 ...
##  $ V10   : num  0.0908 -0.167 0.2076 -0.055 0.7531 ...
##  $ V11   : num  -0.552 1.613 0.625 -0.226 -0.823 ...
##  $ V12   : num  -0.6178 1.0652 0.0661 0.1782 0.5382 ...
##  $ V13   : num  -0.991 0.489 0.717 0.508 1.346 ...
##  $ V14   : num  -0.311 -0.144 -0.166 -0.288 -1.12 ...
##  $ V15   : num  1.468 0.636 2.346 -0.631 0.175 ...
##  $ V16   : num  -0.47 0.464 -2.89 -1.06 -0.451 ...
##  $ V17   : num  0.208 -0.115 1.11 -0.684 -0.237 ...
##  $ V18   : num  0.0258 -0.1834 -0.1214 1.9658 -0.0382 ...
##  $ V19   : num  0.404 -0.146 -2.262 -1.233 0.803 ...
##  $ V20   : num  0.2514 -0.0691 0.525 -0.208 0.4085 ...
##  $ V21   : num  -0.01831 -0.22578 0.248 -0.1083 -0.00943 ...
##  $ V22   : num  0.27784 -0.63867 0.77168 0.00527 0.79828 ...
##  $ V23   : num  -0.11 0.101 0.909 -0.19 -0.137 ...
##  $ V24   : num  0.0669 -0.3398 -0.6893 -1.1756 0.1413 ...
##  $ V25   : num  0.129 0.167 -0.328 0.647 -0.206 ...
##  $ V26   : num  -0.189 0.126 -0.139 -0.222 0.502 ...
##  $ V27   : num  0.13356 -0.00898 -0.05535 0.06272 0.21942 ...
##  $ V28   : num  -0.0211 0.0147 -0.0598 0.0615 0.2152 ...
##  $ Amount: num  149.62 2.69 378.66 123.5 69.99 ...
##  $ Class : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
summary(credit_data)
```

```
##       Time              V1                    V2                   V3
##  Min.   :     0   Min.   :-56.40751   Min.   :-72.71573   Min.   :-48.3256
##  1st Qu.: 54202   1st Qu.: -0.92037   1st Qu.: -0.59855   1st Qu.: -0.8904
##  Median : 84692   Median :  0.01811   Median :  0.06549   Median :  0.1799
##  Mean   : 94814   Mean   :  0.00000   Mean   :  0.00000   Mean   :  0.0000
##  3rd Qu.:139321   3rd Qu.:  1.31564   3rd Qu.:  0.80372   3rd Qu.:  1.0272
##  Max.   :172792   Max.   :  2.45493   Max.   : 22.05773   Max.   :  9.3826
##        V4                 V5                    V6                 V7
##  Min.   :-5.68317   Min.   :-113.74331   Min.   :-26.1605   Min.   :-43.5572
##  1st Qu.:-0.84864   1st Qu.:  -0.69160   1st Qu.: -0.7683   1st Qu.: -0.5541
##  Median :-0.01985   Median :  -0.05434   Median : -0.2742   Median :  0.0401
##  Mean   : 0.00000   Mean   :   0.00000   Mean   :  0.0000   Mean   :  0.0000
##  3rd Qu.: 0.74334   3rd Qu.:   0.61193   3rd Qu.:  0.3986   3rd Qu.:  0.5704
##  Max.   :16.87534   Max.   :  34.80167   Max.   : 73.3016   Max.   :120.5895
##        V8                 V9                   V10                 V11
##  Min.   :-73.21672   Min.   :-13.43407   Min.   :-24.58826   Min.   :-4.79747
##  1st Qu.: -0.20863   1st Qu.: -0.64310   1st Qu.: -0.53543   1st Qu.:-0.76249
##  Median :  0.02236   Median : -0.05143   Median : -0.09292   Median :-0.03276
##  Mean   :  0.00000   Mean   :  0.00000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.:  0.32735   3rd Qu.:  0.59714   3rd Qu.:  0.45392   3rd Qu.: 0.73959
##  Max.   : 20.00721   Max.   : 15.59500   Max.   : 23.74514   Max.   :12.01891
##       V12                V13                  V14                 V15
##  Min.   :-18.6837   Min.   :-5.79188   Min.   :-19.2143   Min.   :-4.49894
##  1st Qu.: -0.4056   1st Qu.:-0.64854   1st Qu.: -0.4256   1st Qu.:-0.58288
```

```
##  Median :  0.1400   Median :-0.01357   Median :  0.0506   Median : 0.04807
##  Mean   :  0.0000   Mean   : 0.00000   Mean   :  0.0000   Mean   : 0.00000
##  3rd Qu.:  0.6182   3rd Qu.: 0.66251   3rd Qu.:  0.4931   3rd Qu.: 0.64882
##  Max.   :  7.8484   Max.   : 7.12688   Max.   : 10.5268   Max.   : 8.87774
##       V16                V17                V18
##  Min.   :-14.12985   Min.   :-25.16280   Min.   :-9.498746
##  1st Qu.: -0.46804   1st Qu.: -0.48375   1st Qu.:-0.498850
##  Median :  0.06641   Median : -0.06568   Median :-0.003636
##  Mean   :  0.00000   Mean   :  0.00000   Mean   : 0.000000
##  3rd Qu.:  0.52330   3rd Qu.:  0.39968   3rd Qu.: 0.500807
##  Max.   : 17.31511   Max.   :  9.25353   Max.   : 5.041069
##       V19                V20                V21
##  Min.   :-7.213527   Min.   :-54.49772   Min.   :-34.83038
##  1st Qu.:-0.456299   1st Qu.: -0.21172   1st Qu.: -0.22839
##  Median : 0.003735   Median : -0.06248   Median : -0.02945
##  Mean   : 0.000000   Mean   :  0.00000   Mean   :  0.00000
##  3rd Qu.: 0.458949   3rd Qu.:  0.13304   3rd Qu.:  0.18638
##  Max.   : 5.591971   Max.   : 39.42090   Max.   : 27.20284
##       V22                V23                V24
##  Min.   :-10.933144   Min.   :-44.80774   Min.   :-2.83663
##  1st Qu.: -0.542350   1st Qu.: -0.16185   1st Qu.:-0.35459
##  Median :  0.006782   Median : -0.01119   Median : 0.04098
##  Mean   :  0.000000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.:  0.528554   3rd Qu.:  0.14764   3rd Qu.: 0.43953
##  Max.   : 10.503090   Max.   : 22.52841   Max.   : 4.58455
##       V25                V26                V27
##  Min.   :-10.29540   Min.   :-2.60455   Min.   :-22.565679
##  1st Qu.: -0.31715   1st Qu.:-0.32698   1st Qu.: -0.070840
##  Median :  0.01659   Median :-0.05214   Median :  0.001342
##  Mean   :  0.00000   Mean   : 0.00000   Mean   :  0.000000
##  3rd Qu.:  0.35072   3rd Qu.: 0.24095   3rd Qu.:  0.091045
##  Max.   :  7.51959   Max.   : 3.51735   Max.   : 31.612198
##       V28                Amount             Class
##  Min.   :-15.43008   Min.   :    0.00   Min.   :0.000000
##  1st Qu.: -0.05296   1st Qu.:    5.60   1st Qu.:0.000000
##  Median :  0.01124   Median :   22.00   Median :0.000000
##  Mean   :  0.00000   Mean   :   88.35   Mean   :0.001728
##  3rd Qu.:  0.07828   3rd Qu.:   77.17   3rd Qu.:0.000000
##  Max.   : 33.84781   Max.   :25691.16   Max.   :1.000000
```

```r
# Check for missing values
sum(is.na(credit_data))
```

```
## [1] 0
```

Handling Class Imbalance

Since the dataset is highly imbalanced, with only 0.17% of the transactions being fraudulent, I apply the ROSE technique to balance the data.

```r
# Apply ROSE to balance the dataset
balanced_data <- ROSE(Class ~ ., data = credit_data, seed = 1)$data
```

```r
# Check the new class distribution
balanced_data %>%
  group_by(Class) %>%
  summarise(count = n()) %>%
  ggplot(aes(x = as.factor(Class), y = count, fill = as.factor(Class))) +
  geom_bar(stat = "identity") +
  labs(title = "Balanced Class Distribution", x = "Class", y = "Count")
```

Splitting the Data into Training and Test Sets

I split the data into **training** and **testing** sets (80% training, 20% testing) to ensure that our models can be evaluated effectively.

```r
# Split the data into training and testing sets (80% training, 20% testing)
set.seed(123)
train_index <- createDataPartition(balanced_data$Class, p = 0.8, list = FALSE)
train_data <- balanced_data[train_index, ]
test_data <- balanced_data[-train_index, ]

# Verify the split
table(train_data$Class)
```

```
##
##      0      1
## 113587 114259
```

```r
table(test_data$Class)
```

```
##
##     0     1
## 28562 28399
```

Modeling

I apply **Logistic Regression** and **Decision Tree** models to the dataset.

```r
# Logistic Regression Model
log_model <- glm(Class ~ ., data = train_data, family = binomial)
```

Logistic Regression

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

```r
# Predictions
log_preds <- predict(log_model, test_data, type = "response")
log_preds_class <- ifelse(log_preds > 0.5, 1, 0)

# Confusion Matrix for Logistic Regression
log_cm <- confusionMatrix(as.factor(log_preds_class), as.factor(test_data$Class))
log_cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 28072  3181
##          1   490 25218
##
##                Accuracy : 0.9356
##                  95% CI : (0.9335, 0.9376)
##     No Information Rate : 0.5014
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8711
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9828
##             Specificity : 0.8880
##          Pos Pred Value : 0.8982
##          Neg Pred Value : 0.9809
##              Prevalence : 0.5014
##          Detection Rate : 0.4928
##    Detection Prevalence : 0.5487
```

```
##       Balanced Accuracy : 0.9354
##
##        'Positive' Class : 0
##
```

```r
# Decision Tree Model
tree_model <- rpart(Class ~ ., data = train_data, method = "class")

# Predictions
tree_preds <- predict(tree_model, test_data, type = "class")

# Confusion Matrix for Decision Tree
tree_cm <- confusionMatrix(tree_preds, as.factor(test_data$Class))
tree_cm
```

Decision Tree

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 27518   971
##          1  1044 27428
##
##                Accuracy : 0.9646
##                  95% CI : (0.9631, 0.9661)
##     No Information Rate : 0.5014
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9292
##
##  Mcnemar's Test P-Value : 0.1087
##
##             Sensitivity : 0.9634
##             Specificity : 0.9658
##          Pos Pred Value : 0.9659
##          Neg Pred Value : 0.9633
##              Prevalence : 0.5014
##          Detection Rate : 0.4831
##    Detection Prevalence : 0.5001
##       Balanced Accuracy : 0.9646
##
##        'Positive' Class : 0
##
```

Evaluation: ROC Curves

I evaluate both models using ROC-AUC curves to compare their performance.

```r
# ROC Curve for Logistic Regression
roc_curve_log <- roc(test_data$Class, log_preds)
```

```
## Setting levels: control = 0, case = 1
```

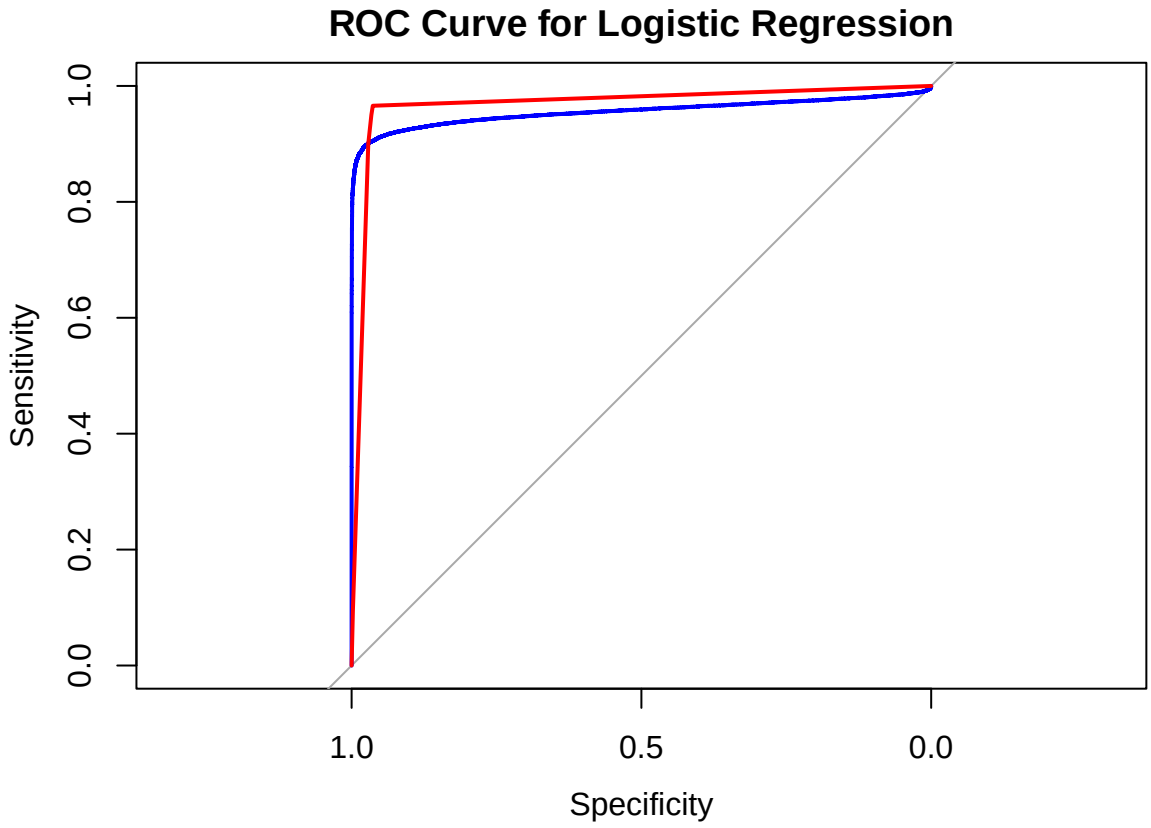
```
## Setting direction: controls < cases
```

```r
plot(roc_curve_log, col = "blue", main = "ROC Curve for Logistic Regression")

# ROC Curve for Decision Tree
tree_probs <- predict(tree_model, test_data, type = "prob")[, 2]
roc_curve_tree <- roc(test_data$Class, tree_probs)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```r
plot(roc_curve_tree, col = "red", add = TRUE)
```




```r
# Display AUC values
log_auc <- auc(roc_curve_log)
tree_auc <- auc(roc_curve_tree)
cat("Logistic Regression AUC:", log_auc, "\n")
```

```
## Logistic Regression AUC: 0.9553965
```

```
cat("Decision Tree AUC:", tree_auc, "\n")
```

```
## Decision Tree AUC: 0.9675895
```

Results

The table below summarizes the confusion matrices and AUC values for both **Logistic Regression** and **Decision Tree** models:

Model	Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	0.9355524	0.8982178	0.9828443	0.9386274	0.9553965
Decision Tree	0.9646249	0.9659167	0.9634479	0.9646807	0.9675895

Conclusion

In this project, I applied both **Logistic Regression** and **Decision Tree** models to detect fraudulent credit card transactions. Both models performed well, and further improvements can be made by experimenting with additional algorithms or improving feature engineering.

References

- UCI Machine Learning Repository: Credit Card Fraud Detection