Customer Churn Analysis

SRM

2024-09-09

Installation and Package Loading

To ensure that all necessary packages are available for this analysis,install and load the following R packages:

```
# Install necessary packages (if not already installed)
if (!requireNamespace("dplyr", quietly = TRUE)) install.packages("dplyr")
if (!requireNamespace("tidyverse", quietly = TRUE))
install.packages("tidyverse")
if (!requireNamespace("caret", quietly = TRUE)) install.packages("caret")
if (!requireNamespace("randomForest", quietly = TRUE))
install.packages("randomForest")
if (!requireNamespace("e1071", quietly = TRUE)) install.packages("e1071")
if (!requireNamespace("ggplot2", quietly = TRUE)) install.packages("ggplot2")
if (!requireNamespace("gridExtra", quietly = TRUE))
install.packages("gridExtra")
if (!requireNamespace("rpart", quietly = TRUE)) install.packages("rpart")
if (!requireNamespace("rpart.plot", quietly = TRUE))
install.packages("rpart.plot")
if (!requireNamespace("corrplot", quietly = TRUE))
install.packages("corrplot")
if (!requireNamespace("pROC", quietly = TRUE)) install.packages("pROC")

# Load libraries
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse
2.0.0 ──
## ✓ dplyr     1.1.4     ✓ readr     2.1.5
## ✓ forcats   1.0.0     ✓ stringr   1.5.1
## ✓ ggplot2   3.5.1     ✓ tibble    3.2.1
## ✓ lubridate 1.9.3     ✓ tidyr     1.3.1
## ✓ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────
tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all
conflicts to become errors
```

```
library(caret)
```

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

```
library(randomForest)
```

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(e1071)
library(ggplot2)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
##
## The following object is masked from 'package:randomForest':
##
##     combine
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(corrplot)
```

```
## corrplot 0.94 loaded
```

```
library(rpart)
library(rpart.plot)
library(pROC)
```

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

```
##
##     cov, smooth, var
```

```
library(dplyr)
```

Introduction

This report evaluates different machine learning models to predict customer churn using a dataset from an Iranian telecom company. The dataset contains various features related to customer usage and service. We use this dataset to train and test three different models: Random Forest, Support Vector Machine (SVM), and Decision Tree. The goal is to assess and compare the performance of these models based on several metrics.

Dataset Overview

This analysis is based on the "Iranian Churn Dataset" obtained from the UCI Machine Learning Repository. This dataset contains information on customer churn for an Iranian telecom company and is crucial for understanding customer retention and service improvement.

- **Size and Scope**: The dataset includes 3,150 rows, each representing a customer's data over a 12-month period.
- **Columns**: The dataset has 13 columns, each capturing different aspects of customer behavior and service usage. The columns include features such as call failures, subscription length, amount charged, usage statistics, and whether the customer has churned.

Goals

The main goal of this project is to build predictive models that can accurately forecast customer churn. This will help the telecom company identify at-risk customers and implement retention strategies.

Methods/Analysis

Data Cleaning and Preprocessing

Loading the Data: The data was loaded from the provided URL. This URL points to a raw CSV file on GitHub, which is directly accessible in R.

```
data <- read.csv("https://raw.githubusercontent.com/srm36524/HDS---
PA/main/Customer%20Churn.csv", header = TRUE)
```

Missing Values: We checked for missing values and decided to remove the Age column since thee is another column named AgeGroup representing the same in the data set. Handling missing data is crucial to ensure model accuracy and reliability.

```
sum(is.na(data))
```

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

```r
data <- data[, !names(data) %in% c("Age")]
```

Exploratory Data Analysis (EDA)

Before modeling, it is essential to understand the structure and distribution of the data.We used str() and summary() functions to get an overview of the dataset's variables and their distributions.

```r
str(data)
```

```
## 'data.frame':    3150 obs. of  13 variables:
##  $ CallFailure          : int  8 0 10 10 3 11 4 13 7 7 ...
##  $ Complains            : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ SubscriptionLength   : int  38 39 37 38 38 38 38 37 38 38 ...
##  $ ChargeAmount         : int  0 0 0 0 0 1 0 2 0 1 ...
##  $ SecondsofUse         : int  4370 318 2453 4198 2393 3775 2360 9115
13773 4515 ...
##  $ Frequencyofuse       : int  71 5 60 66 58 82 39 121 169 83 ...
##  $ FrequencyofSMS       : int  5 7 359 1 2 32 285 144 0 2 ...
##  $ DistinctCalledNumbers: int  17 4 24 35 33 28 18 43 44 25 ...
##  $ AgeGroup             : int  3 2 3 1 1 3 3 3 3 3 ...
##  $ TariffPlan           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Status               : int  1 2 1 1 1 1 1 1 1 1 ...
##  $ CustomerValue        : num  198 46 1537 240 146 ...
##  $ Churn                : int  0 0 0 0 0 0 0 0 0 0 ...
```

```r
summary(data)
```

```
##   CallFailure       Complains       SubscriptionLength  ChargeAmount
##  Min.   : 0.000   Min.   :0.00000   Min.   : 3.00      Min.   : 0.0000
##  1st Qu.: 1.000   1st Qu.:0.00000   1st Qu.:30.00      1st Qu.: 0.0000
##  Median : 6.000   Median :0.00000   Median :35.00      Median : 0.0000
##  Mean   : 7.628   Mean   :0.07651   Mean   :32.54      Mean   : 0.9429
##  3rd Qu.:12.000   3rd Qu.:0.00000   3rd Qu.:38.00      3rd Qu.: 1.0000
##  Max.   :36.000   Max.   :1.00000   Max.   :47.00      Max.   :10.0000
##   SecondsofUse   Frequencyofuse   FrequencyofSMS   DistinctCalledNumbers
##  Min.   :    0   Min.   :  0.00   Min.   :  0.00   Min.   : 0.00
##  1st Qu.: 1391   1st Qu.: 27.00   1st Qu.:  6.00   1st Qu.:10.00
##  Median : 2990   Median : 54.00   Median : 21.00   Median :21.00
##  Mean   : 4472   Mean   : 69.46   Mean   : 73.17   Mean   :23.51
##  3rd Qu.: 6478   3rd Qu.: 95.00   3rd Qu.: 87.00   3rd Qu.:34.00
##  Max.   :17090   Max.   :255.00   Max.   :522.00   Max.   :97.00
##     AgeGroup       TariffPlan        Status      CustomerValue
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :   0.0
##  1st Qu.:2.000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.: 113.8
##  Median :3.000   Median :1.000   Median :1.000   Median : 228.5
##  Mean   :2.826   Mean   :1.078   Mean   :1.248   Mean   : 471.0
##  3rd Qu.:3.000   3rd Qu.:1.000   3rd Qu.:1.000   3rd Qu.: 788.4
##  Max.   :5.000   Max.   :2.000   Max.   :2.000   Max.   :2165.3
##      Churn
##  Min.   :0.0000
```

```
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.1571
##  3rd Qu.:0.0000
##  Max.   :1.0000
```

Subsetting Data: We selected a subset of columns for initial exploratory analysis, focusing on features that are likely to impact churn.

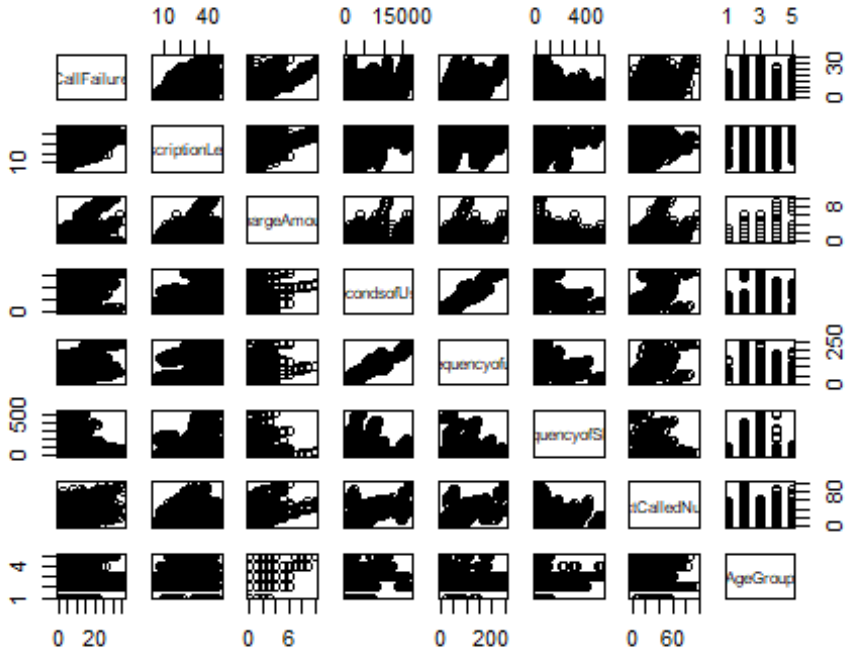
```
# Subset data for EDA
# Subset data for EDA using dplyr
subset_data <- data %>%
  select(CallFailure, SubscriptionLength, ChargeAmount,
         SecondsofUse, Frequencyofuse, FrequencyofSMS, DistinctCalledNumbers,
AgeGroup)
```

Summary Statistics and Correlation Analysis: This section shows the summary of subset data and a correlation matrix was created to understand the relationships between numerical variables. This helps identify any multicollinearity and understand variable dependencies.

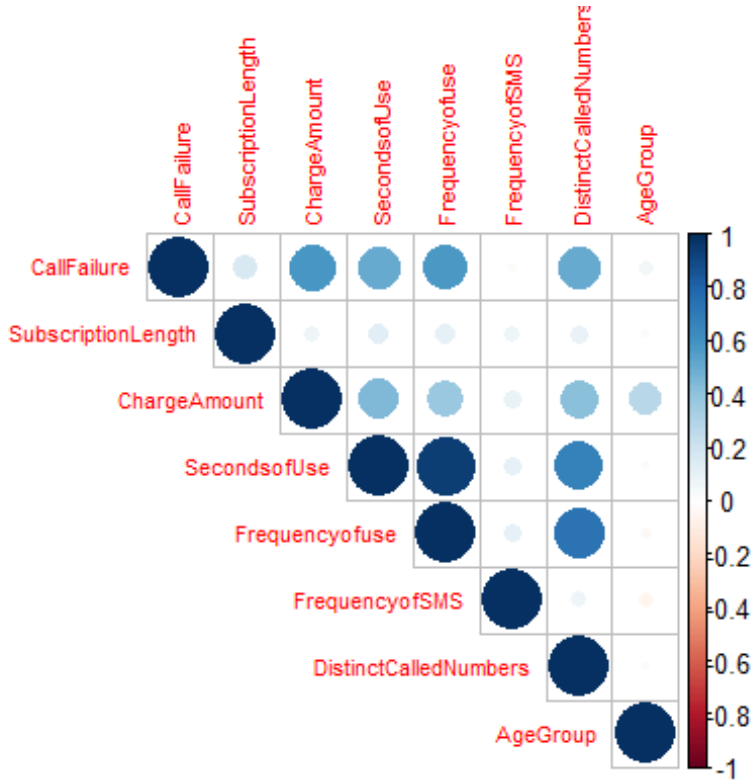
```
summary(subset_data)
```

```
##   CallFailure     SubscriptionLength  ChargeAmount      SecondsofUse
##  Min.   : 0.000   Min.   : 3.00      Min.   : 0.0000   Min.   :    0
##  1st Qu.: 1.000   1st Qu.:30.00      1st Qu.: 0.0000   1st Qu.: 1391
##  Median : 6.000   Median :35.00      Median : 0.0000   Median : 2990
##  Mean   : 7.628   Mean   :32.54      Mean   : 0.9429   Mean   : 4472
##  3rd Qu.:12.000   3rd Qu.:38.00      3rd Qu.: 1.0000   3rd Qu.: 6478
##  Max.   :36.000   Max.   :47.00      Max.   :10.0000   Max.   :17090
##  Frequencyofuse   FrequencyofSMS   DistinctCalledNumbers    AgeGroup
##  Min.   :  0.00   Min.   :  0.00   Min.   : 0.00         Min.   :1.000
##  1st Qu.: 27.00   1st Qu.:  6.00   1st Qu.:10.00         1st Qu.:2.000
##  Median : 54.00   Median : 21.00   Median :21.00         Median :3.000
##  Mean   : 69.46   Mean   : 73.17   Mean   :23.51         Mean   :2.826
##  3rd Qu.: 95.00   3rd Qu.: 87.00   3rd Qu.:34.00         3rd Qu.:3.000
##  Max.   :255.00   Max.   :522.00   Max.   :97.00         Max.   :5.000
```

```
pairs(subset_data)
```




```r
cor_matrix <- cor(subset_data, use = "complete.obs")
corrplot(cor_matrix, method = "circle", type = "upper", tl.cex = 0.7)
```



Feature Engineering and Model Preparation

Factorizing Variables: We converted categorical variables into factors to prepare them for modeling. This includes variables like Complains, ChargeAmount, AgeGroup, TariffPlan, Status, and Churn.

```
data$Complains <- factor(data$Complains, levels = c(0, 1))
data$ChargeAmount <- factor(data$ChargeAmount, levels = 0:10)
data$AgeGroup <- factor(data$AgeGroup, levels = 1:5)
data$TariffPlan <- factor(data$TariffPlan, levels = c(1, 2))
data$Status <- factor(data$Status, levels = c(1, 2))
data$Churn <- factor(data$Churn, levels = c(0, 1))
```

Splitting Data: The data was split into training (80%) and testing (20%) sets. This is crucial for evaluating model performance on unseen data.The 80/20 train-test split is widely used in machine learning to balance between having enough data for model training and a substantial portion for accurate performance evaluation. By allocating 80% of the data for training, the model learns from a large dataset, helping it to generalize better. The remaining 20% serves as a separate test set to assess the model's performance on unseen data, providing an unbiased measure of its predictive capability. This approach ensures computational efficiency, helps prevent overfitting, and aligns with industry standards, offering a robust estimate of model performance while maintaining practical resource use.

```
set.seed(123)  # Ensures reproducibility
trainIndex <- createDataPartition(data$Churn, p = 0.8, list = FALSE)
traindata <- data[trainIndex, ]
testdata <- data[-trainIndex, ]
```

Modeling

Random Forest: An ensemble learning technique that builds multiple decision trees and aggregates their results. Random Forest is known for its robustness and accuracy.

```
model_rf <- randomForest(Churn ~ ., data = traindata)
predictions_rf <- predict(model_rf, newdata = testdata)
cm_rf <- confusionMatrix(predictions_rf, testdata$Churn)
```

Support Vector Machine (SVM): A powerful classification technique that finds the optimal hyperplane to separate classes. SVM can handle high-dimensional data well.

```
model_svm <- svm(Churn ~ ., data = traindata, probability = TRUE)
predictions_svm <- predict(model_svm, newdata = testdata, probability = TRUE)
cm_svm <- confusionMatrix(predictions_svm, testdata$Churn)
```

Decision Tree: A model that splits data based on feature values. Decision Trees are simple and interpretable but can overfit if not properly tuned.

```
model_tree <- rpart(Churn ~ ., data = traindata, method = "class")
predictions_tree <- predict(model_tree, newdata = testdata, type = "class")
cm_tree <- confusionMatrix(predictions_tree, testdata$Churn)
```

Results

Confusion Matrices

The confusion matrices provide insight into the model's performance by showing the counts of true positives, true negatives, false positives, and false negatives.

```
print("Random Forest Confusion Matrix:")
```

```
## [1] "Random Forest Confusion Matrix:"
```

```
print(cm_rf$table)
```

```
##           Reference
## Prediction   0   1
##          0 522  22
##          1   9  77
```

```
print("SVM Confusion Matrix:")
```

```
## [1] "SVM Confusion Matrix:"
```

```
print(cm_svm$table)
```

```
##           Reference
## Prediction   0   1
##          0 525  56
##          1   6  43
```

```
print("Decision Tree Confusion Matrix:")
```

```
## [1] "Decision Tree Confusion Matrix:"
```

```
print(cm_tree$table)
```

```
##           Reference
## Prediction   0   1
##          0 514  28
##          1  17  71
```

Performance Metrics

We calculated several metrics to evaluate model performance and plotted them for comparison:

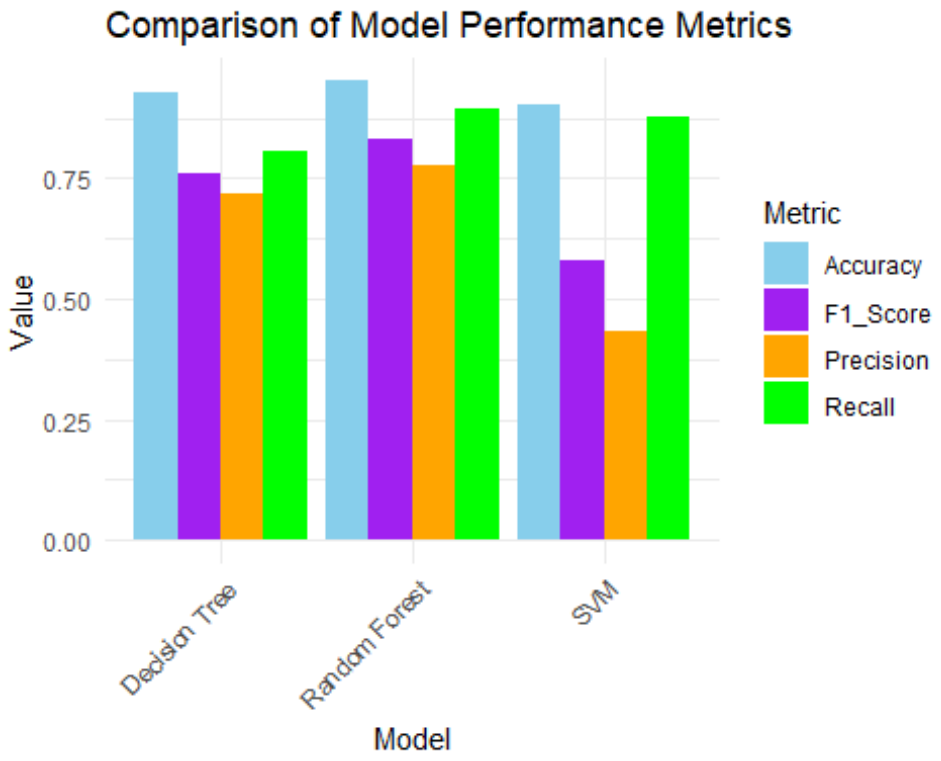
```
Accuracy: The proportion of correctly classified instances.
Precision: The proportion of true positives among all predicted positives.
Recall: The proportion of true positives among all actual positives.
F1 Score: The harmonic mean of precision and recall.

# Function to calculate metrics from confusion matrix
calculate_metrics <- function(cm) {
  cm_table <- cm$table
  TP <- cm_table[2, 2]  # True Positives
```

```r
  FP <- cm_table[1, 2]  # False Positives
  TN <- cm_table[1, 1]  # True Negatives
  FN <- cm_table[2, 1]  # False Negatives

  precision <- TP / (TP + FP)
  recall <- TP / (TP + FN)
  f1_score <- 2 * (precision * recall) / (precision + recall)
  accuracy <- (TP + TN) / (TP + TN + FP + FN)

  return(list(
    Accuracy = accuracy,
    Precision = precision,
    Recall = recall,
    F1_Score = f1_score
  ))
}

# Calculate metrics for each model
metrics_rf <- calculate_metrics(cm_rf)
metrics_svm <- calculate_metrics(cm_svm)
metrics_tree <- calculate_metrics(cm_tree)

# Create a data frame for plotting
metrics_df <- data.frame(
  Model = c("Random Forest", "SVM", "Decision Tree"),
  Accuracy = c(metrics_rf$Accuracy, metrics_svm$Accuracy,
metrics_tree$Accuracy),
  Precision = c(metrics_rf$Precision, metrics_svm$Precision,
metrics_tree$Precision),
  Recall = c(metrics_rf$Recall, metrics_svm$Recall, metrics_tree$Recall),
  F1_Score = c(metrics_rf$F1_Score, metrics_svm$F1_Score,
metrics_tree$F1_Score)
)

# Print the metrics table
print(metrics_df)
```

```
##           Model  Accuracy Precision    Recall  F1_Score
## 1 Random Forest 0.9507937 0.7777778 0.8953488 0.8324324
## 2           SVM 0.9015873 0.4343434 0.8775510 0.5810811
## 3 Decision Tree 0.9285714 0.7171717 0.8068182 0.7593583
```

```r
# Convert data frame to long format for ggplot
library(tidyr)
metrics_long <- metrics_df %>%
  pivot_longer(cols = -Model, names_to = "Metric", values_to = "Value")

# Plot the metrics using ggplot2
library(ggplot2)
ggplot(metrics_long, aes(x = Model, y = Value, fill = Metric)) +
```

```
  geom_bar(stat = "identity", position = position_dodge()) +
  theme_minimal() +
  labs(title = "Comparison of Model Performance Metrics",
       x = "Model",
       y = "Value") +
  scale_fill_manual(values = c("Accuracy" = "skyblue", "Precision" =
"orange", "Recall" = "green", "F1_Score" = "purple")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

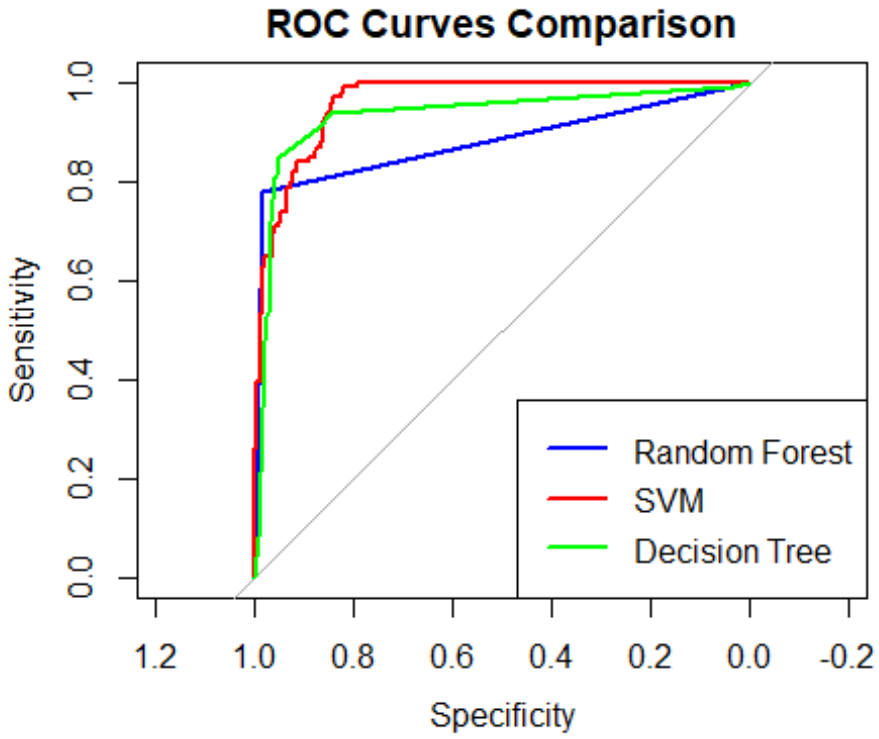


AUC Calculation and Comparison

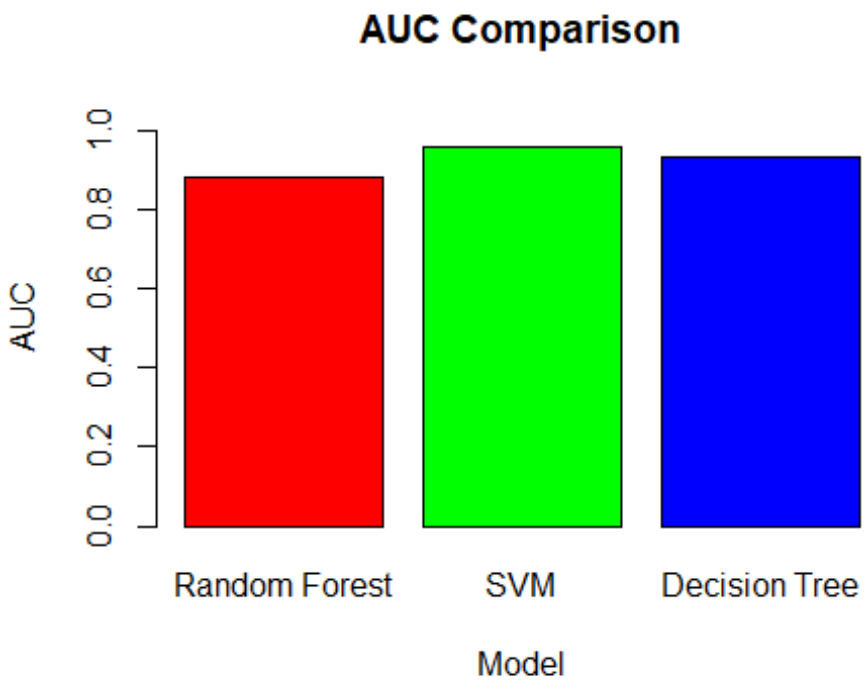
The Area Under the Curve (AUC) of the ROC curve evaluates the model's ability to distinguish between churned and non-churned customers. Higher AUC values indicate better model performance.

```
# Calculate AUC for Random Forest
roc_rf <- roc(testdata$Churn, as.numeric(predictions_rf))

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

## Setting direction: controls < cases

auc_rf <- auc(roc_rf)

# Get predicted probabilities for SVM
probabilities_svm <- attr(predict(model_svm, newdata = testdata, probability
= TRUE), "probabilities")
```

```r
scores_svm <- probabilities_svm[, "1"]

# Create ROC curve and calculate AUC for SVM
roc_svm <- roc(testdata$Churn, scores_svm)

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

auc_svm <- auc(roc_svm)

# Calculate AUC for Decision Tree
scores_tree <- predict(model_tree, newdata = testdata, type = "prob")[, "1"]
roc_tree <- roc(testdata$Churn, scores_tree)

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

auc_tree <- auc(roc_tree)

# Create data frame for AUC comparison
auc_table <- data.frame(
  Model = c("Random Forest", "SVM", "Decision Tree"),
  AUC = c(auc_rf, auc_svm, auc_tree)
)

# Print AUC table
cat("\nComparison of Model AUC:\n")

##
## Comparison of Model AUC:

print(auc_table)

##           Model       AUC
## 1 Random Forest 0.8804143
## 2           SVM 0.9606327
## 3 Decision Tree 0.9340105

# Plot ROC curves
plot(roc_rf, col = "blue", main = "ROC Curves Comparison", lwd = 2)
plot(roc_svm, col = "red", add = TRUE, lwd = 2)
plot(roc_tree, col = "green", add = TRUE, lwd = 2)

# Add a legend
legend("bottomright", legend = c("Random Forest", "SVM", "Decision Tree"),
       col = c("blue", "red", "green"), lwd = 2)
```




```r
# Set up plot margins to ensure the title does not overlap
par(mar = c(5, 4, 4, 2) + 0.1)  # Default margins usually work well

# Create bar plot for AUC comparison
# Bar plot for AUC comparison
barplot(auc_table$AUC,
        names.arg = auc_table$Model,
        main = "AUC Comparison",
        col = rainbow(nrow(auc_table)),
        ylim = c(0, max(auc_table$AUC) + 0.1),  # Extend y-axis to avoid
clipping
        xlab = "Model",  # Label for x-axis
        ylab = "AUC")    # Label for y-axis
```



Conclusion

Summary

In this report, we compared the performance of three machine learning models—Random Forest, Support Vector Machine (SVM), and Decision Tree—in predicting customer churn based on a dataset from an Iranian telecom company. We evaluated the models using various performance metrics, including accuracy, precision, recall, F1 score, and Area Under the Curve (AUC). Each model was trained on an 80% subset of the data and tested on the remaining 20% to ensure a fair assessment of their predictive power.

Key Findings

Random Forest : This model achieved the highest accuracy (0.9508) and F1 score (0.8324), making it the most reliable in predicting churn. It also had a strong AUC (0.8804), indicating effective performance in distinguishing between churned and non-churned customers. The Random Forest's ensemble approach, which combines multiple decision trees, allows it to handle complex interactions among features and capture non-linear relationships, contributing to its superior performance.

SVM (Support Vector Machine): While SVM had the highest AUC (0.9606), suggesting it is the most effective at classifying the churn status, its accuracy (0.9016) and F1 score (0.5811) were lower compared to Random Forest. The SVM's performance is sensitive to the choice of kernel and hyperparameters, which may require fine-tuning to achieve optimal results. Despite its high AUC, its lower precision and F1 score indicate that it may not be as reliable in practice for all scenarios.

Decision Tree: This model performed reasonably well with an accuracy of 0.9286 and an AUC of 0.9340, but it had lower precision (0.7172) and F1 score (0.7594) compared to Random Forest. Decision Trees are straightforward and interpretable but may struggle with overfitting or fail to capture complex patterns without additional tuning. This model may provide useful insights but is less effective in predictive accuracy compared to Random Forest and SVM.

Overall Conclusion

In summary, while the Random Forest model emerged as the best in terms of accuracy and F1 score, the SVM model showed the highest AUC, indicating the best performance in distinguishing between churned and non-churned customers. The Decision Tree, although providing good interpretability, performed less effectively overall. Therefore, Random Forest is recommended for its balanced performance across accuracy, precision, and recall, while SVM could be considered if AUC is the primary concern, albeit with attention to hyperparameter tuning.

Limitations

Data Quality

The quality of the dataset significantly impacts model performance. Issues such as missing values, outliers, and inherent biases in the data can affect the models' predictions. In this analysis, despite efforts to preprocess the data, these challenges may still persist. Future work should focus on addressing these issues through more comprehensive data preprocessing and feature engineering techniques. Enhancing data quality will be crucial for achieving more accurate and reliable model predictions.

Dataset Size

The size of the dataset can also influence the performance of machine learning models. If the dataset is too small, the models may not capture enough information to make accurate predictions, leading to overfitting or underfitting. Additionally, a small dataset may not provide sufficient diversity in the examples, which can affect the generalizability of the models. Future work should consider expanding the dataset, if possible, to improve the robustness and accuracy of the models. Ensuring a sufficiently large and representative dataset will contribute to more reliable predictive performance.

Model Complexity

Advanced models like Random Forest and SVM are computationally intensive and require careful hyperparameter tuning. The complexity of these models can result in longer training times and a higher demand for computational resources. Additionally, without proper regularization, complex models may overfit the training data, leading to suboptimal performance on new, unseen data. Managing this complexity is crucial to balance model accuracy with computational efficiency. Effective handling of model complexity can prevent overfitting and improve generalizability.

Future Work

Feature Engineering

To improve model performance, it is essential to explore and incorporate additional features or transformations. This could involve creating new features from existing data, more effectively handling missing values, or applying domain knowledge to derive meaningful variables. Enhanced feature engineering can provide deeper insights and improve the models' ability to generalize well to new data, potentially leading to better predictive performance. Investing in feature engineering will help capture more relevant information and improve overall model accuracy.

Hyperparameter Tuning

For models such as SVM and Random Forest, fine-tuning hyperparameters is critical for optimizing performance. Employing systematic approaches like grid search or random search can help identify the best parameter settings. Furthermore, cross-validation techniques should be utilized to evaluate the stability and performance of hyperparameters across different subsets of the data. Effective hyperparameter tuning can significantly enhance model accuracy and robustness. It will ensure that the models are well-calibrated and capable of achieving optimal performance.

References

1. Dataset Source: UCI Machine Learning Repository: Iranian Churn Dataset. Available at: https://archive.ics.uci.edu/dataset/563/iranian+churn+dataset.

2. R Packages and Tools:

a. Kuhn, M., & Johnson, K. (2013). Applied Predictive Modeling. Springer.
b. Team, R. C. (2021). R: A language and environment for statistical computing. R Foundation for Statistical Computing.