Waze

yong yang

2024-08-07

Project Report

Introduction/Overview/Executive Summary

Dataset Introduction This project utilizes an uncommon dataset, the Waze navigation app user behavior dataset. This dataset includes users' navigation behavior and app usage. Variables include total kilometers navigated by users, driving days, session count, device type, and more.

Project Objectives The goal of this project is to predict user churn. Advanced machine learning techniques are used to construct classification models, specifically applying Random Forest and XGBoost algorithms for modeling and evaluating model performance.

Overview of Execution Steps

1. Data loading and cleaning
2. Feature engineering
3. Dataset division
4. Model training and evaluation
5. Results analysis and summary

Methods/Analysis

Data Cleaning and Feature Engineering

1. **Data loading:** Load the dataset from the CSV file.
2. **Feature engineering:**
   - Created multiple new features, such as `km_per_driving_day`, `percent_sessions_in_last_month`, `professional_driver`, etc., to enhance the model's predictive power.
   - Removed missing values to ensure data integrity.
   - Created a new label feature `label2`, marking churned users as 1 and retained users as 0.

Dataset Division The dataset is divided into training, validation, and testing sets: - 80% of the data is used as the training and validation set, with 75% for training and 25% for validation. - The remaining 20% of the data is used as the testing set.

Modeling Methods

1. **Random Forest:**
   - Set up a parameter grid for hyperparameter tuning, selecting the best `mtry`, `splitrule`, and `min.node.size`.
   - Train the model using cross-validation.
   - Evaluate the model's performance on the validation set, calculating RMSE and accuracy.
2. **XGBoost:**
   - Set up a parameter grid for hyperparameter tuning, selecting the best `nrounds`, `max_depth`, `eta`, and other parameters.
   - Train the model using cross-validation.
   - Evaluate the model's performance on the validation set, calculating RMSE and accuracy.

Results

Model Performance

- **Random Forest Model:**
  - Validation set RMSE: 0.2199707
  - Validation set accuracy: 85.44%
- **XGBoost Model:**
  - Validation set RMSE: 0.3815879
  - Validation set accuracy: 85.44%

Conclusion

Summary of Report Content This project analyzes the Waze navigation app user dataset and constructs two machine learning models, Random Forest and XGBoost, to predict user churn. By performing feature engineering and data cleaning, the predictive performance of the model is improved.

Impact This project demonstrates how to apply advanced machine learning techniques for user churn prediction and has practical significance. By selecting an uncommon dataset, the project showcases the ability to handle new datasets and its innovativeness.

Limitations Although the model's performance is relatively good, it may require more features and more complex models for optimization in practical applications. Additionally, the parameter tuning process may require more computational resources and time.

Future Work In the future, we can consider: 1. Introducing more features, such as users' socioeconomic background, usage of other apps, etc. 2. Trying other advanced machine learning models, such as neural networks, support vector machines, etc. 3. Conducting a more detailed analysis of the data to discover potential influencing factors and improve the model's prediction accuracy.

By making the above improvements, the model's practicality and accuracy can be further enhanced, providing more reliable user churn predictions.

```r
# Install necessary packages if they are not already installed
#install.packages("dplyr")
#install.packages("tidyr")
#install.packages("caret")
#install.packages("xgboost")
#install.packages("randomForest")
#install.packages("Metrics")
```

```r
# Load required libraries
library(dplyr)
```

```
##
##    'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(tidyr)
library(caret)
```

```
##      ggplot2

##      lattice
```

```r
library(xgboost)
```

```
##
##    'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice
```

```r
library(randomForest)
```

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
##    'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

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

```r
library(Metrics)
```

```
##
##    'Metrics'
```

```
## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```r
# Load the dataset
df0 <- read.csv('waze_dataset.csv')
```

```r
# Display the first five rows of the dataset
head(df0)
```

```
##   ID    label sessions drives total_sessions n_days_after_onboarding
## 1  0 retained      283    226      296.74827                    2276
## 2  1 retained      133    107      326.89660                    1225
## 3  2 retained      114     95      135.52293                    2651
## 4  3 retained       49     40       67.58922                      15
## 5  4 retained       84     68      168.24702                    1562
## 6  5 retained      113    103      279.54444                    2637
##   total_navigations_fav1 total_navigations_fav2 driven_km_drives
## 1                    208                      0        2628.8451
## 2                     19                     64       13715.9206
## 3                      0                      0        3059.1488
## 4                    322                      7         913.5911
## 5                    166                      5        3950.2020
## 6                      0                      0         901.2387
##   duration_minutes_drives activity_days driving_days  device
## 1               1985.7751            28           19 Android
## 2               3160.4729            13           11  iPhone
## 3               1610.7359            14            8 Android
## 4                587.1965             7            3  iPhone
## 5               1219.5559            27           18 Android
## 6                439.1014            15           11  iPhone
```

```r
# Copy the dataset
df <- df0
```

```r
# Create 'km_per_driving_day' feature
df <- df %>%
  mutate(km_per_driving_day = driven_km_drives / driving_days) %>%
  mutate(km_per_driving_day = ifelse(is.infinite(km_per_driving_day), 0, km_per_driving_day))

# Create 'percent_sessions_in_last_month' feature
df <- df %>%
  mutate(percent_sessions_in_last_month = sessions / total_sessions)
```

```r
# Create 'professional_driver' feature
df <- df %>%
  mutate(professional_driver = ifelse(drives >= 60 & driving_days >= 15, 1, 0))

# Create 'total_sessions_per_day' feature
df <- df %>%
  mutate(total_sessions_per_day = total_sessions / n_days_after_onboarding)
```

```r
# Create 'km_per_hour' feature
df <- df %>%
  mutate(km_per_hour = driven_km_drives / (duration_minutes_drives / 60))

# Create 'km_per_drive' feature
df <- df %>%
  mutate(km_per_drive = driven_km_drives / drives) %>%
  mutate(km_per_drive = ifelse(is.infinite(km_per_drive), 0, km_per_drive))
```

```r
# Create 'percent_of_drives_to_favorite' feature
df <- df %>%
  mutate(percent_of_drives_to_favorite = (total_navigations_fav1 + total_navigations_fav2) / total_sess

# Drop rows with missing values in the 'label' column
df <- df %>%
  drop_na(label)
```

```r
# Create 'device2' and 'label2' features
df <- df %>%
  mutate(device2 = ifelse(device == 'Android', 0, 1),
         label2 = ifelse(label == 'churned', 1, 0))

# Drop the 'ID' column
df <- df %>%
  select(-ID)

# Display the first five rows of the modified dataset
head(df)
```

```
##      label sessions drives total_sessions n_days_after_onboarding
## 1 retained      283    226      296.74827                    2276
## 2 retained      133    107      326.89660                    1225
## 3 retained      114     95      135.52293                    2651
## 4 retained       49     40       67.58922                      15
## 5 retained       84     68      168.24702                    1562
## 6 retained      113    103      279.54444                    2637
##   total_navigations_fav1 total_navigations_fav2 driven_km_drives
## 1                    208                      0        2628.8451
## 2                     19                     64       13715.9206
## 3                      0                      0        3059.1488
## 4                    322                      7         913.5911
## 5                    166                      5        3950.2020
## 6                      0                      0         901.2387
##   duration_minutes_drives activity_days driving_days  device km_per_driving_day
```

```
## 1              1985.7751             28            19 Android          138.36027
## 2              3160.4729             13            11  iPhone         1246.90187
## 3              1610.7359             14             8 Android          382.39360
## 4               587.1965              7             3  iPhone          304.53037
## 5              1219.5559             27            18 Android          219.45567
## 6               439.1014             15            11  iPhone           81.93079
##   percent_sessions_in_last_month professional_driver total_sessions_per_day
## 1                      0.9536703                   1             0.13038149
## 2                      0.4068565                   0             0.26685436
## 3                      0.8411861                   0             0.05112144
## 4                      0.7249677                   0             4.50594808
## 5                      0.4992659                   1             0.10771256
## 6                      0.4042291                   0             0.10600851
##   km_per_hour km_per_drive percent_of_drives_to_favorite device2 label2
## 1    79.43030     11.63206                     0.7009308       0      0
## 2   260.38990    128.18617                     0.2539029       1      0
## 3   113.95346     32.20157                     0.0000000       0      0
## 4    93.35114     22.83978                     4.8676400       1      0
## 5   194.34297     58.09121                     1.0163627       0      0
## 6   123.14769      8.74989                     0.0000000       1      0
```

```r
# Split the dataset into training and testing sets
set.seed(1)
trainIndex <- createDataPartition(df$label2, p = .8,
                                  list = FALSE,
                                  times = 1)
dfTrain <- df[trainIndex, ]
dfTest  <- df[-trainIndex, ]
```

```r
# Split the training set into training and validation sets
trainIndex <- createDataPartition(dfTrain$label2, p = .75,
                                  list = FALSE,
                                  times = 1)
dfTrain <- dfTrain[trainIndex, ]
dfVal  <- dfTrain[-trainIndex, ]
```

```r
# Separate features and labels for training
X_train <- dfTrain %>% select(-label, -label2, -device)
y_train <- dfTrain$label2
```

```r
# Separate features and labels for validation
X_val <- dfVal %>% select(-label, -label2, -device)
y_val <- dfVal$label2
```

```r
# Separate features and labels for testing
X_test <- dfTest %>% select(-label, -label2, -device)
y_test <- dfTest$label2
```

```r
# Set up random forest parameters
rf_grid <- expand.grid(
  mtry = floor(sqrt(ncol(X_train))),
  splitrule = "gini",
```

```r
  min.node.size = 1
)
```

```r
# Train random forest model
# Ensure the response variable is a factor (classification task)
y_train <- as.factor(y_train)  # Convert y_train to factor
```

```r
# Define parameter grid
rf_grid <- expand.grid(mtry = c(2, 4, 6), splitrule = "gini", min.node.size = c(1, 5, 10))
```

```r
# Train random forest model
rf_model <- train(
  X_train, y_train,
  method = "ranger",
  trControl = trainControl(method = "cv", number = 4),
  tuneGrid = rf_grid,
  importance = 'impurity'
)
```

```r
# Optimal parameters and performance
rf_model$bestTune
```

```
##   mtry splitrule min.node.size
## 3    2      gini            10
```

```r
rf_model$results
```

```
##   mtry splitrule min.node.size  Accuracy      Kappa  AccuracySD    KappaSD
## 1    2      gini             1 0.8288878 0.08494672 0.004149850 0.02075044
## 2    2      gini             5 0.8287762 0.09034344 0.006261899 0.03194815
## 3    2      gini            10 0.8304430 0.10034798 0.005543605 0.02528076
## 4    4      gini             1 0.8282208 0.11521451 0.005250934 0.02544238
## 5    4      gini             5 0.8288878 0.11929849 0.004425948 0.02056662
## 6    4      gini            10 0.8289992 0.11883572 0.003034801 0.01532656
## 7    6      gini             1 0.8278880 0.12818912 0.003448159 0.01762048
## 8    6      gini             5 0.8282216 0.12816110 0.002367806 0.01750771
## 9    6      gini            10 0.8287765 0.13212216 0.004863544 0.02211582
```

```r
# Predict on validation set
rf_val_preds <- predict(rf_model, X_val)
```

```r
# Convert factor type to numeric type
y_val_num <- as.numeric(as.character(y_val))
rf_val_preds_num <- as.numeric(as.character(rf_val_preds))
```

```r
# Calculate RMSE
rf_val_rmse <- rmse(y_val_num, rf_val_preds_num)
rf_val_rmse
```

```
## [1] 0.2199707
```

```r
# Set up XGBoost parameters
xgb_grid <- expand.grid(
  nrounds = 300,
  max_depth = c(6, 12),
  eta = c(0.01, 0.1),
  gamma = 0,
  colsample_bytree = 1,
  min_child_weight = c(3, 5),
  subsample = 1
)
```

```r
# Train XGBoost model
xgb_model <- train(X_train, y_train,
                   method = "xgbTree",
                   trControl = trainControl(method = "cv", number = 4),
                   tuneGrid = xgb_grid)

# Optimal parameters and performance
xgb_model$bestTune
```

```
##   nrounds max_depth  eta gamma colsample_bytree min_child_weight subsample
## 2     300         6 0.01     0                1                5         1
```

```r
xgb_model$results
```

```
##    eta max_depth gamma colsample_bytree min_child_weight subsample nrounds
## 1 0.01         6     0                1                3         1     300
## 2 0.01         6     0                1                5         1     300
## 5 0.10         6     0                1                3         1     300
## 6 0.10         6     0                1                5         1     300
## 3 0.01        12     0                1                3         1     300
## 4 0.01        12     0                1                5         1     300
## 7 0.10        12     0                1                3         1     300
## 8 0.10        12     0                1                5         1     300
##    Accuracy     Kappa  AccuracySD    KappaSD
## 1 0.8277766 0.1208635 0.004874158 0.03480036
## 2 0.8301105 0.1339021 0.003450635 0.02394835
## 5 0.8224433 0.1621082 0.005496372 0.02249278
## 6 0.8223321 0.1586481 0.005534353 0.03101168
## 3 0.8232215 0.1409753 0.004340122 0.02274578
## 4 0.8249992 0.1508908 0.003908186 0.02227023
## 7 0.8186654 0.1508642 0.006738097 0.01015165
## 8 0.8156649 0.1462505 0.008769174 0.01706145
```

```r
# Predict on validation set
xgb_val_preds <- predict(xgb_model, X_val)
```

```r
# Convert factor type to numeric type
y_val_num <- as.numeric(as.character(y_val))
xgb_val_preds <- as.numeric(as.character(xgb_val_preds))
```

```r
# Calculate RMSE
xgb_val_rmse <- rmse(y_val, xgb_val_preds)
xgb_val_rmse
```

```
## [1] 0.3815879
```

```r
# Calculate accuracy
accuracy <- mean(y_val == xgb_val_preds)
accuracy
```

```
## [1] 0.8543907
```