Predicting Movie Box Office Revenues Using Sentiment Analysis and SVR

Promothesh Chatterjee*

Predicting Movie Box Office Revenues Using Sentiment Analysis and SVR

This R script predicts movie box office revenues using sentiment analysis of movie reviews and Support Vector Regression (SVR). The process involves collecting and preparing data, performing sentiment analysis, engineering features, training the model, and evaluating its performance.

```
# Install and load necessary libraries
#install.packages(c("e1071", "quanteda", "textdata", "tidyverse", "lubridate", "caret"))

library(e1071)
library(quanteda)
library(textdata)
library(tidyverse)
library(lubridate)
library(caret)
```

Load IMDB Review Data

Load a dataset of IMDB reviews and take a random sample of 5000 reviews to keep the dataset manageable.

```
imdb_reviews <- textdata::dataset_imdb()

# Sample data to keep it manageable (optional, adjust as needed)
set.seed(123)
imdb_reviews <- imdb_reviews %>% sample_n(5000)
```

Create Synthetic Box Office Revenue Data

Define a list of movie names and generate synthetic box office revenue data using a sine wave pattern to simulate weekly trends with added random noise for variability.

```
# Define some movie names
movie_names <- c("Movie A", "Movie B", "Movie C", "Movie D", "Movie E")
movie_count <- length(movie_names)

# Create synthetic box office revenue data for corresponding review dates
dates <- seq(from = as.Date("2022-01-01"), to = as.Date("2022-12-31"), by = "day")

# Generate revenues for each movie
set.seed(123)
revenue_trend <- sin(seq(0, 2 * pi, length.out = length(dates))) * 1000000 + 2000000
random_noise <- rnorm(length(dates), mean = 0, sd = 50000)
box_office_data <- tibble(
```

```
  date = rep(dates, movie_count),
  movie = rep(movie_names, each = length(dates)),
  revenue = rep(revenue_trend, movie_count) + random_noise
)

# Assign dates and movie names to reviews (assume reviews are evenly distributed over dates and movies)
imdb_reviews <- imdb_reviews %>%
  mutate(date = rep(dates, length.out = nrow(imdb_reviews)),
         movie = rep(movie_names, length.out = nrow(imdb_reviews)))
```

Perform Sentiment Analysis

Use the quanteda package for text processing and sentiment analysis.

```
# Using quanteda for text processing and sentiment analysis
data_corpus <- corpus(imdb_reviews$text)
tokens <- tokens(data_corpus, remove_punct = TRUE)
tokens <- tokens_tolower(tokens)

# Load a sentiment lexicon
afinn <- textdata::lexicon_afinn()
afinn_dict <- dictionary(list(positive = afinn$word[afinn$value > 0], negative = afinn$word[afinn$value

# Convert tokens to a document-feature matrix (dfm)
dfm_tokens <- dfm(tokens)

# Calculate sentiment scores using dfm_lookup
sentiment_dfm <- dfm_lookup(dfm_tokens, dictionary = afinn_dict)
sentiment_scores <- convert(sentiment_dfm, to = "data.frame") %>%
  mutate(sentiment = positive - negative) %>%
  pull(sentiment)

# Add sentiment scores to the data
imdb_reviews <- imdb_reviews %>%
  mutate(sentiment = sentiment_scores)
```

Aggregate Sentiment Scores and Prepare Features

Aggregate sentiment scores by date and movie, and prepare features including lagged revenue and sentiment. Creating lagged features in a time series context is crucial for several reasons:

1. **Capturing Temporal Dependencies**: Time series data often exhibit temporal dependencies where past values influence future values. Lagged features allow the model to incorporate this historical

information. For example, the revenue of a movie on a particular day might be influenced by the revenue of the previous day.

2. **Predictive Power**: Including lagged variables can enhance the predictive power of the model by providing additional relevant information that might not be captured by the current values alone.
3. **Handling Autocorrelation**: Time series data often suffer from autocorrelation, where the residuals are not independent. Lagged features help in modeling this autocorrelation, leading to better model performance.
4. **Trend and Seasonality**: Lagged features can help capture trends and seasonal patterns in the data. For instance, if there are weekly patterns in the movie revenue data, lagged features will help the model to learn and predict these patterns.

In our code:

- `lagged_revenue`: Captures the revenue from the previous day, which can provide insights into short-term trends and fluctuations in movie revenue.
- `lagged_sentiment`: Captures the sentiment from the previous day, which can be an indicator of public opinion or hype that might affect revenue.

By including these lagged features, we are equipping your Support Vector Regression (SVR) model with historical context, potentially leading to better performance in forecasting future revenue.

```r
# Aggregate sentiment scores by date and movie
daily_sentiment <- imdb_reviews %>%
  group_by(date, movie) %>%
  summarize(daily_sentiment = mean(sentiment, na.rm = TRUE))

# Merge with box office revenue
model_data <- left_join(box_office_data, daily_sentiment, by = c("date", "movie")) %>%
  na.omit()

# Create lagged features
model_data <- model_data %>%
  group_by(movie) %>%
  mutate(lagged_revenue = lag(revenue, 1),
         lagged_sentiment = lag(daily_sentiment, 1)) %>%
  ungroup() %>%
  na.omit()
```

Train SVR Model

Split the data into training and testing sets, then train an SVR model.

```r
# Split data into training and testing sets
set.seed(123)
train_index <- sample(seq_len(nrow(model_data)), size = 0.8 * nrow(model_data))
train_data <- model_data[train_index, ]
test_data <- model_data[-train_index, ]

# Train SVR model
svr_model <- svm(revenue ~ lagged_revenue + lagged_sentiment + movie, data = train_data, kernel = "radi

# Make predictions
predictions <- predict(svr_model, test_data)
```

Support Vector Regression (SVR) is a type of Support Vector Machine (SVM) that is used for regression tasks. Unlike traditional regression models that aim to minimize the error between predicted and actual values, SVR tries to fit the best line within a predefined margin of tolerance, called epsilon.

SVR is robust to outliers and can handle high-dimensional data well, which makes it suitable for our task where we are dealing with text data.

SVR Model: We use the e1071 package to train an SVR model.Once the SVR model is trained, we use it to predict values for the input features. These predictions are then compared to the target values to calculate residuals.

Evaluate Model Performance

Evaluate the model's performance using the `caret` package to calculate RMSE and R-squared.

```r
# Use caret to calculate RMSE and R-squared
metrics <- postResample(predictions, test_data$revenue)
metrics
```

```
##         RMSE     Rsquared          MAE
## 1.162120e+05 9.751441e-01 9.231207e+04
```

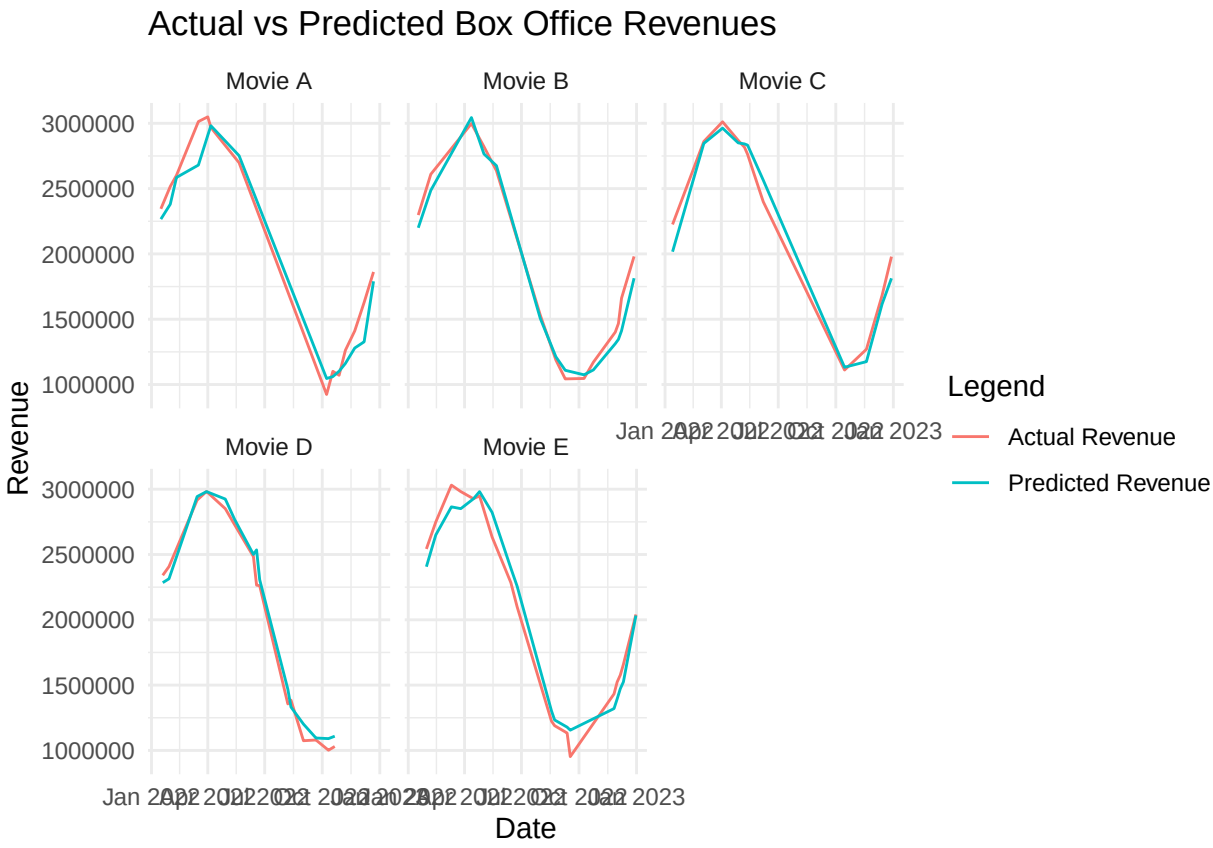
Visualize Results

Visualize the actual vs predicted box office revenues using `ggplot2`.

```r
# Plot actual vs predicted box office revenues
test_data$predicted_revenue <- predictions

ggplot(test_data, aes(x = date)) +
  geom_line(aes(y = revenue, color = "Actual Revenue")) +
  geom_line(aes(y = predicted_revenue, color = "Predicted Revenue")) +
```

```
  facet_wrap(~ movie) +
  labs(title = "Actual vs Predicted Box Office Revenues",
       x = "Date",
       y = "Revenue",
       color = "Legend") +
  theme_minimal()
```



We have done all necessary steps from data collection and preparation, sentiment analysis, feature engineering, model training, and evaluation, to visualization of the results. Adjust the parameters and data as needed to fit your specific requirements.