MovieLens Project Report

Kevin McGowan

2024-11-26

Abstract

This MovieLens project report, part of the HarvardX: PH125.9x Data Science: Capstone course, explores the development of a machine learning algorithm to predict movie ratings—a critical step in creating personalized recommendation systems. These systems have become essential tools in the video streaming industry, where engaging viewers with tailored content enhances satisfaction, retention, and engagement duration. This report performs the step-by-step methodology for building and validating a machine learning model to predict movie ratings by following this outline:

- **Introduction**
- **Methodology**
  - Exploratory Data Analysis
  - Feature Development
  - Modeling
  - Visualization
- **Results**
  - Testing on Independent Final Holdout Set
- **Discussion**

Introduction

In August 2024, Forbes reported that Netflix, the global leader in video streaming, had 260.28 million subscribers worldwide, with the global video streaming industry valued at $544 billion annually (Durrani & Allen, 2024). Personalized recommendation systems are vital for platforms like Netflix to keep viewers engaged by presenting content aligned with user preferences. This report focuses on developing a robust predictive model that leverages historical user behavior and movie characteristics to forecast ratings. Accurate rating predictions serve as the foundation for crafting personalized recommendations, not only improving viewer satisfaction but also informing strategic content decisions.

Beyond entertainment, machine learning models like this have broad applications in other industries, such as forecasting property values in real estate or predicting high school graduation rates in education. By demonstrating the process of building and validating a predictive model, this report provides insights into how historical patterns can inform future behavior and decision-making.

The Dataset

This predictive model considers two primary sources of information: the user and the movie. The user's input is represented by their past ratings, providing insight into their preferences. The movie information includes attributes like genre and aggregated ratings from other users. Unlike human reviewers, the algorithm is constrained to these structured data points but compensates by processing immense volumes of information.

The data set used in this machine learning task is the 10 million MovieLens data set, which contains 10,000,054 ratings applied to 10,681 movies by 71,567 users (Harper & Konstan, 2015). These data are spread across two

primary files—ratings.dat and movies.dat—that need to be joined for analysis. Each user has been assigned a unique ID to anonymize their identity, and only users who rated at least 20 movies were included in the dataset. No additional demographic or behavioral information about the users is provided.

The ratings are recorded on a 5-star scale, allowing for 0.5-star increments for a total of 10 possible rating levels between 0.5 and 5. The data set includes 18 distinct genres, such as Action, Comedy, Drama, and Western. Next the method section loads and explores the data set, builds performance features, and trains the machine learning algorithm on the 10 million MovieLens data set.

Method/Analysis

Data Wrangling

Before training the movie rating prediction algorithm out of the 10mil dataset, the data set needs to be prepared for analysis. The steps in doing so are as follows:

- Load Data: download, unzip, and combine ratings.dat and movies.dat;
- Explore Data: variables, dimensions, and insights
- Feature Development: descriptive statistics of users and movies
- Model Development: training and cross validation
- Prediction: Forecasting ratings on an independent validation set.

Before training the movie rating prediction algorithm, we prepare the 10-million MovieLens dataset. The dataset is downloaded, unzipped, and merged by joining the `ratings.dat` and `movies.dat` files on the `movieId` field. The merged dataset includes user ratings, movie information (titles and genres), and timestamps.

Next, the dataset is split into two subsets: an **edx training set** (90% of the data) and a **final holdout test set** (10% of the data). To ensure integrity during evaluation, the code below also verify that all user-movie combinations in the holdout set are also present in the training set. Below is the code used for these steps:

Load Data

With the required packages loaded—collections of pre-written code, functions, and documentation that extend R's functionality—the two aprts of the dataset can now be downloaded, variable types can be defined, then combined, and load it into the environment.

```
# Increase timeout for downloading large datasets
options(timeout = 120)

# Download and unzip the dataset
dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

# load the user ratings half of the data set
ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

# load the movie half of the data set
movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

# Read and process the ratings file by specifying variable types
ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
```

```
                           stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

# Read and process the movies file
movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

# Merge ratings and movies datasets by specifying variable types
movielens <- left_join(ratings, movies, by = "movieId")
```

With the two datasets combined into one movielens dataset, the final validation must be removed from the dataset to create an independent test set for evaluating the quality of the machine learning model.

```
# Split the data into training (edx) and final holdout test sets
set.seed(1, sample.kind="Rounding") # Use "Rounding" for R 3.6 or later
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Ensure all userId and movieId in the holdout set are present in the training set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add back any rows excluded from the holdout set into the training set
removed <- anti_join(temp, final_holdout_test)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

```
edx <- rbind(edx, removed)

# Clean up unnecessary variables
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

The dataset is now loaded, combined, and split into two parts: the edx training set (90%) and the final holdout test set (10%).

Explore Data

Before delving into feature development, it is essential to explore the data set to gain insights into its structure and characteristics. This exploratory phase increases understanding of the data set and guides subsequent steps in feature engineering and modeling.

The MovieLens 10-million dataset, now called (`edx`), contains information about user-movie interactions, including ratings, timestamps, movie titles, movie ID, user ID, and genres. Each row represents a user's rating for a specific movie. As seen in the code output below, the data set includes:

- **9,000,055 rows**

- **6 columns**: `userId`, `movieId`, `rating`, `timestamp`, `title`, and `genres`
- **69,878 unique users** and **10,677 unique movies**

```
# Display the first few rows of the dataset
head(edx)
```

```
# Count the number of rows and columns (TRY NOT USING CAT BECAUSE I DON'T THINK IT'S NEEDED IN R MARK D
cat("Number of rows:", nrow(edx), "\n")
```

```
## Number of rows: 9000055
```

```
cat("Number of columns:", ncol(edx), "\n")
```

```
## Number of columns: 6
```

Rating Distribution

The ratings range from 0.5 to 5, in increments of 0.5 stars. A look at the most common ratings reveals:

- 4 stars: Most frequent rating, with 2,588,430 occurrences.
- 3 stars: Second most frequent, with 2,121,240 occurrences.
- Other frequent ratings include 5, 3.5, and 2 stars.

```
# Count the top 5 most common ratings
most_common_ratings <- edx %>%
  group_by(rating) %>%
  summarise(total_occurrence = n()) %>%
  arrange(desc(total_occurrence)) %>%
  head(5)

print(most_common_ratings)
```

```
## # A tibble: 5 x 2
##   rating total_occurrence
##    <dbl>            <int>
## 1    4             2588430
## 2    3             2121240
## 3    5             1390114
## 4    3.5            791624
## 5    2              711422
```

```
# show all unique ratings from smallest to largest
sorted_ratings <- sort(unique(edx$rating))
sorted_ratings
```

```
##  [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

This clustering of ratings around 4 stars suggests averages and standard deviations of ratings for movies could provide insight into movie ratings.

Most Rated Movies

Certain movies receive significantly more ratings, potentially biasing the model. The top-rated movies include:

- Pulp Fiction (1994): 31,362 ratings
- Forrest Gump (1994): 31,079 ratings
- The Silence of the Lambs (1991): 30,382 ratings

```
# Identify the top-rated movies
count_ratings_by_movie <- edx %>%
```

```
  group_by(title) %>%
  summarise(rating_counts = n()) %>%
  arrange(desc(rating_counts))

print(count_ratings_by_movie, n = 10)
```

```
## # A tibble: 10,676 x 2
##    title                                                     rating_counts
##    <chr>                                                             <int>
##  1 Pulp Fiction (1994)                                               31362
##  2 Forrest Gump (1994)                                               31079
##  3 Silence of the Lambs, The (1991)                                  30382
##  4 Jurassic Park (1993)                                              29360
##  5 Shawshank Redemption, The (1994)                                  28015
##  6 Braveheart (1995)                                                 26212
##  7 Fugitive, The (1993)                                              25998
##  8 Terminator 2: Judgment Day (1991)                                 25984
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)      25672
## 10 Apollo 13 (1995)                                                  24284
## # i 10,666 more rows
```

Genre Distribution

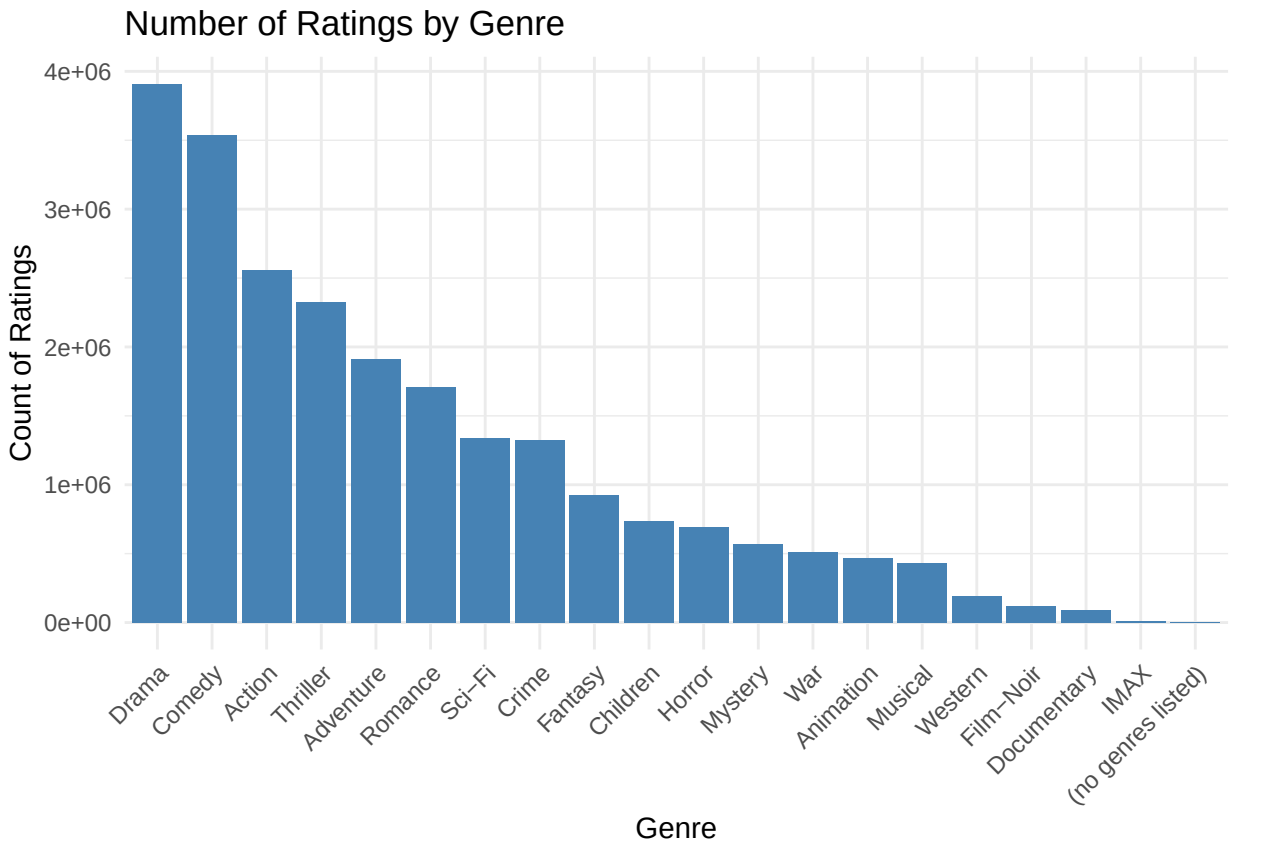
The dataset spans a wide array of genres. The four most frequent genres are:

- Drama: Appears 3,910,127 times.
- Comedy: Appears 3,540,930 times.
- Thriller: Appears 2,325,899 times.
- Romance: Appears 1,712,100 times.

There are a few movies with no genre listed. This edge case could present a problem when it comes to modeling on new datasets.

```
# Count how many movies belong to specific genres
all_genres <- edx %>%
  pull(genres) %>%
  str_split("\\|") %>%
  unlist() %>%
  table() %>%
  as.data.frame()

# Rename the columns for clarity
colnames(all_genres) <- c("Genre", "Count")

# Sort the genres by count in descending order
all_genres <- all_genres %>%
  arrange(desc(Count))

# Plot the genres as a bar plot
library(ggplot2)
ggplot(all_genres, aes(x = reorder(Genre, -Count), y = Count)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "Number of Ratings by Genre",
       x = "Genre",
       y = "Count of Ratings") +
```

```
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



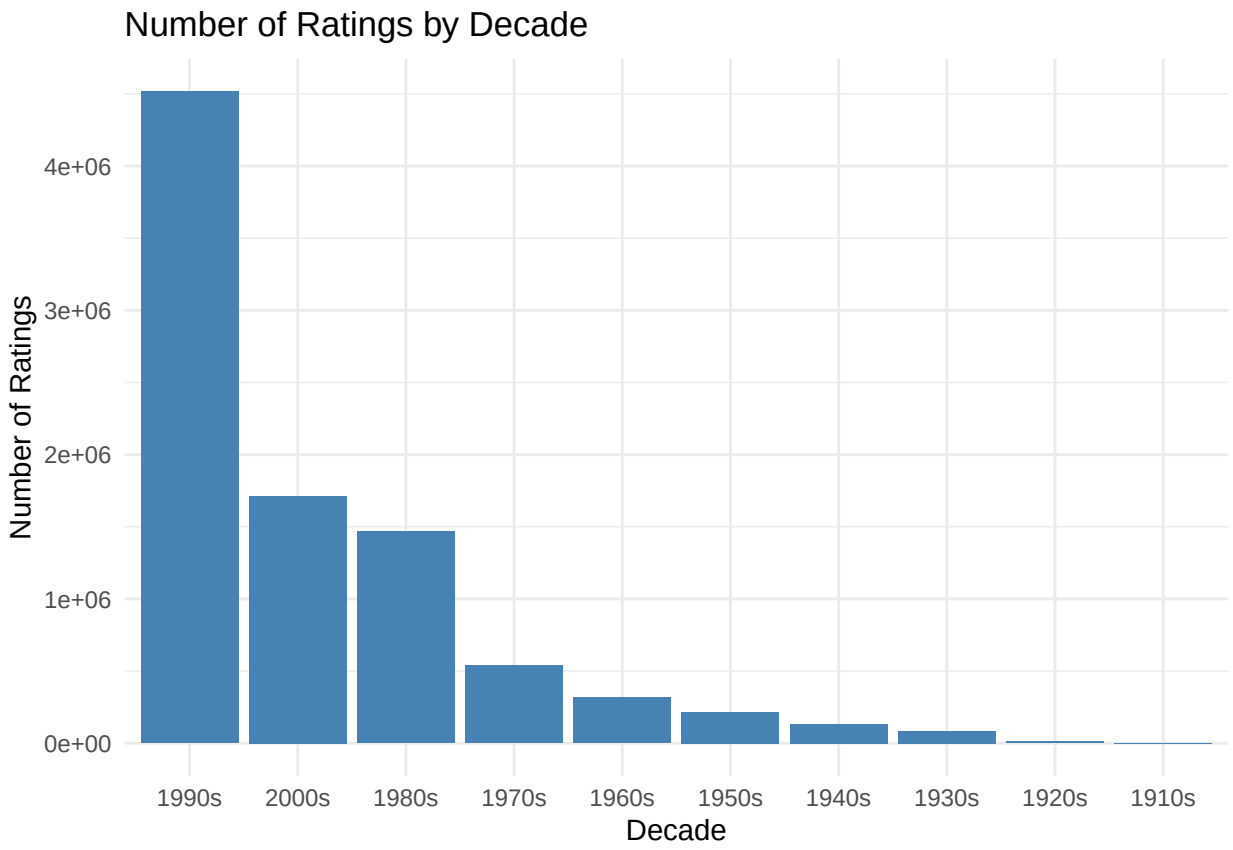
Movie Rating Distribution

The dataset spans an entire century of movies. The 4 decades with the most movie ratings are:

- 90s
- 2000s
- 80s
- 70s

```
# Create a new dataset for exploration
edx_exploration <- edx %>%
  mutate(
    movie_year = as.numeric(gsub("[^0-9]", "", str_extract(title, "\\((\\d{4})\\)"))),
    decade = paste0(floor(movie_year / 10) * 10, "s")
  )

# Count ratings by decade
ratings_by_decade <- edx_exploration %>%
  group_by(decade) %>%
  summarise(count = n()) %>%
  arrange(desc(count))

# Plot a bar chart of ratings by decade
ggplot(ratings_by_decade, aes(x = reorder(decade, -count), y = count)) +
  geom_bar(stat = "identity", fill = "steelblue") +
```

```
  labs(
    title = "Number of Ratings by Decade",
    x = "Decade",
    y = "Number of Ratings"
  ) +
  theme_minimal()
```



Missing Values

Looking at the data set again, the top 10 rows show we still have 6 columns and 0 NAs

```
#review top 10 rows of the data set
head(edx)
```

```
# Count the total number of missing values in the edx dataset
total_missing <- sum(is.na(edx))
cat("Total Missing Values in edx:", total_missing, "\n")
```

```
## Total Missing Values in edx: 0
```

Key Findings

This exploratory analysis highlights:

- The skew in ratings distribution, with users preferring mid-to-high ratings.
- A subset of movies dominates the dataset in terms of user engagement.
- Drama and Comedy are the most represented genres, reflecting their popularity among users.
- There are ratings for movies from 1950 to 2000s, with the majority in the 90s, presenting an opportunity to look a ratings by decade.

- There are no NAs but some movies have no genres applied to them, representing an edge case that should be accounted for in feature development.

This understanding lays the groundwork for the next step: Feature Development. By leveraging the insights gained here, meaningful features can be created that capture user preferences, movie attributes, and other contextual factors.

Feature Engineering

To build an effective movie rating prediction model, features must be built that capture meaningful patterns from the data. Feature engineering is the process of transforming raw data into informative variables, or "features," that enhance the predictive power of a model. This step bridges the gap between exploratory data analysis and machine learning, enabling the model to make more accurate predictions.

In the MovieLens dataset, features need to encapsulate key aspects of both users and movies. For example:

- User-specific features: The number of ratings a user has given, their average rating tendencies, and how consistent or inconsistent their ratings are over time.
- Movie-specific features: A movie's release year, how recently it was rated, the number of ratings it has received, and the overall sentiment reflected in its ratings.
- Temporal features: Variables that account for the time of rating, such as the year, month, or even the day of the week.

By deriving these features, the raw data is transformed into a structured data set that highlights patterns related to user preferences and movie characteristics.

To ensure consistency and scalability, a single processing function called feature_engineering is used. This function applies transformations to the raw data set in a systematic way, creating all the necessary features for modeling. Below is the code for the feature_engineering function, which includes user-specific, movie-specific, and temporal features:

```
feature_engineering <- function(data) {

  # Step 1: Extract movie year and other temporal features
  data <- data %>%
    mutate(
      rating_year = year(as_datetime(timestamp)),#year rating was given
      rating_month = month(as_datetime(timestamp)),#month rating was given
      rating_day_of_week = wday(as_datetime(timestamp), label = TRUE),#day of week rating was given
      movie_year = as.numeric(gsub("[^0-9]", "", str_extract(title, "\\((\\d{4})\\)"))),#year of movie
      movie_age = rating_year - movie_year,#age of the movie
      decade = paste0(floor(movie_year / 10) * 10, "s") # Decade of the movie release
    )

  # Step 2: User-specific features
  user_features <- data %>%
    group_by(userId) %>%
    summarise(
      user_rating_count = n(),#count of ratings per user
      user_avg_rating = mean(rating, na.rm = TRUE),#avg ratings a user gives
      rating_duration = max(rating_year) - min(rating_year),#duration users has been rating movies for
      recent_rating_count = sum(rating_year >= (max(rating_year) - 1)),#recent count for a movie connoi
      last_rating_date = max(as_datetime(timestamp)),#date of last rating for recentcy variable
      avg_ratings_per_year = n() / (max(rating_year) - min(rating_year) + 1),#average ratings users giv
      rating_std_dev = sd(rating, na.rm = TRUE),#standard deviation of user ratings given a year
      trend = ifelse(n() > 1, coef(lm(rating ~ rating_year))[2], 0)#trend of user ratings a year
    ) %>%
```

```
    ungroup() %>%
    mutate(trend = ifelse(is.na(trend), 0, trend)) # Replace NA trends with 0

  # Step 3: Movie-specific features
  movie_features <- data %>%
    group_by(movieId) %>%
    summarise(
      movie_rating_count = n(),#count of movie ratings
      movie_avg_rating = mean(rating, na.rm = TRUE),#average movie ratings
      movie_rating_std_dev = sd(rating, na.rm = TRUE)#standard deviation of movie ratings
    ) %>%
    ungroup() %>%
    mutate(movie_rating_std_dev = ifelse(is.na(movie_rating_std_dev), 0, movie_rating_std_dev)) # Remov

  # Step 4: Combine all features into the main dataset
  data <- data %>%
    left_join(user_features, by = "userId") %>%
    mutate(
      user_movie_diff = rating - user_avg_rating,
      days_since_last_rating = as.numeric(difftime(as_datetime(timestamp), last_rating_date, units = "da
    ) %>%
    left_join(movie_features, by = "movieId") %>%
    mutate(years_since_release = rating_year - movie_year)

  # Rename columns to replace hyphens with underscores
  colnames(data) <- gsub("-", "_", colnames(data))

  return(data)
}
```

Now that the feature_engineering function has been created, it needs to be applied to the edx dataset, generating a new dataset (edx_clean) enriched with the engineered features and cleaned of NAs (if new datasets had them).

```
# Apply the feature engineering function to the edx dataset
edx_clean <- feature_engineering(edx)
```

Feature Engineering Summary

Now that the feature engineering process has been applied, the following code shows a summary of all the features now in the dataset:

```
# Extract feature namess
feature_names <- colnames(edx_clean)

# Create a data frame for display
feature_table <- data.frame(Feature_Names = feature_names)

# Print the feature table
print(feature_table)
```

```
##             Feature_Names
## 1                  userId
## 2                 movieId
## 3                  rating
```

```
## 4               timestamp
## 5                   title
## 6                  genres
## 7             rating_year
## 8            rating_month
## 9      rating_day_of_week
## 10             movie_year
## 11              movie_age
## 12                 decade
## 13      user_rating_count
## 14        user_avg_rating
## 15        rating_duration
## 16    recent_rating_count
## 17       last_rating_date
## 18   avg_ratings_per_year
## 19         rating_std_dev
## 20                  trend
## 21        user_movie_diff
## 22 days_since_last_rating
## 23     movie_rating_count
## 24       movie_avg_rating
## 25   movie_rating_std_dev
## 26    years_since_release
```

Feature Enginneering Validation

With the features reviewed, this section of code validates that the feature_engineering function worked appropriately with the edx dataset and is clean.

```
# Check for missing values in key features
cat("Missing values check for main features:\n")
```

```
## Missing values check for main features:
```

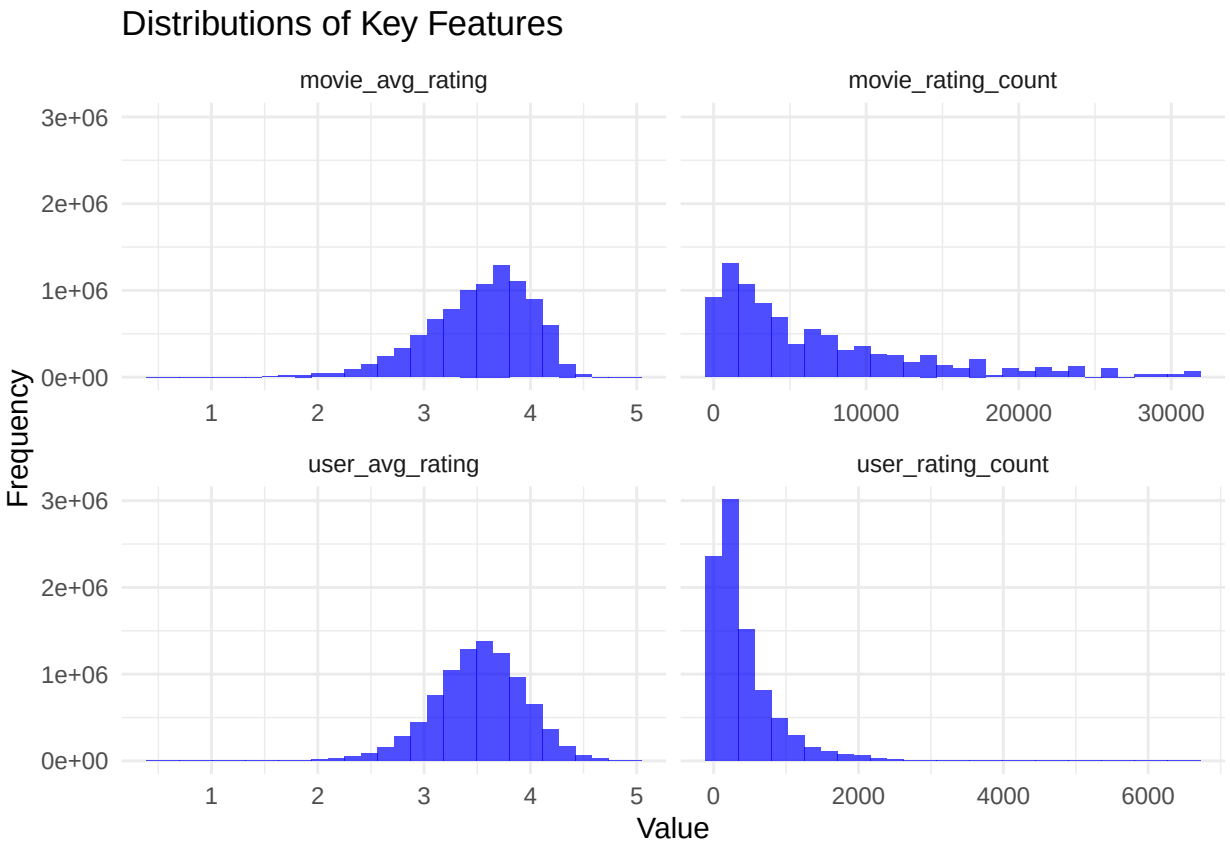
```
summary(edx_clean %>% select(user_rating_count, user_avg_rating, rating_duration, recent_rating_count, m
```

```
##  user_rating_count user_avg_rating rating_duration  recent_rating_count
##  Min.   :  10.0    Min.   :0.500   Min.   : 0.000   Min.   :   1.0
##  1st Qu.: 108.0    1st Qu.:3.252   1st Qu.: 0.000   1st Qu.:  66.0
##  Median : 257.0    Median :3.529   Median : 0.000   Median : 143.0
##  Mean   : 424.2    Mean   :3.512   Mean   : 1.395   Mean   : 239.4
##  3rd Qu.: 542.0    3rd Qu.:3.800   3rd Qu.: 2.000   3rd Qu.: 298.0
##  Max.   :6616.0    Max.   :5.000   Max.   :12.000   Max.   :4648.0
##  movie_rating_count movie_avg_rating
##  Min.   :    1      Min.   :0.500
##  1st Qu.: 1634      1st Qu.:3.218
##  Median : 4223      Median :3.591
##  Mean   : 6787      Mean   :3.512
##  3rd Qu.: 9862      3rd Qu.:3.876
##  Max.   :31362      Max.   :5.000
```

The summary above suggests no missing values in key features, ensuring they are ready for modeling.

The histograms below reveal a normal distributions for average ratings and skewed distributions for user and movie-specific rating counts.

```r
# Visualize distributions of user and movie-specific features
edx_clean %>%
  select(user_rating_count, user_avg_rating, movie_rating_count, movie_avg_rating) %>%
  pivot_longer(cols = everything(), names_to = "Feature", values_to = "Value") %>%
  ggplot(aes(x = Value)) +
  geom_histogram(bins = 30, fill = "blue", alpha = 0.7) +
  facet_wrap(~ Feature, scales = "free_x") +
  labs(title = "Distributions of Key Features", x = "Value", y = "Frequency") +
  theme_minimal()
```



The first table below confirms the development of 20 new features, in additional to the 6 original columns, and the absence of missing values. However, there are several columns that show all 0s, so the second table below confirm these columns aren't empty. With non-zero values numbering from 3 to 9 million for each feature, effective feature development is supported thus far.

```r
# Check the head of all features and validate non-zero values
head(edx_clean)
```

```r
# Check for columns with non-zero values
non_zero_summary <- edx_clean %>%
  summarise(across(everything(), ~ sum(. != 0, na.rm = TRUE)))
print("Number of non-zero values per column:")
```

```
## [1] "Number of non-zero values per column:"
```

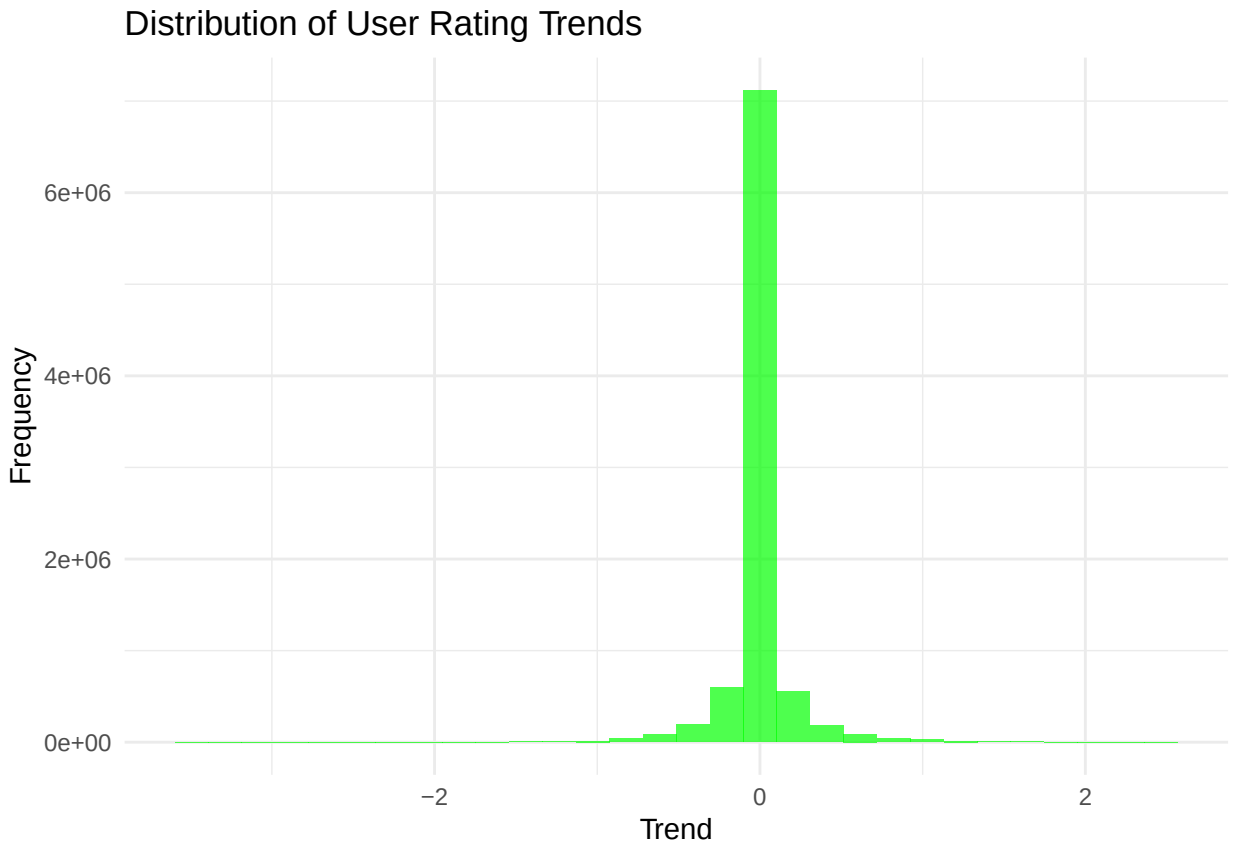
```r
print(non_zero_summary)
```

```
##    userId movieId  rating timestamp   title  genres rating_year rating_month
## 1 9000055 9000055 9000055   9000055 9000055 9000055     9000055      9000055
```

```
##   rating_day_of_week movie_year movie_age  decade user_rating_count
## 1            9000055    9000055   8619140 9000055           9000055
##   user_avg_rating rating_duration recent_rating_count last_rating_date
## 1         9000055         3701494             9000055          9000055
##   avg_ratings_per_year rating_std_dev   trend user_movie_diff
## 1              9000055        8998907 3701494         8978382
##   days_since_last_rating movie_rating_count movie_avg_rating
## 1                8889459            9000055          9000055
##   movie_rating_std_dev years_since_release
## 1              8999843             8619140
```

The histogram below reveals most trends are centered at 0, with some positive and negative deviations indicating variability in user behavior. This finding suggests that most users don't have a trend in their rating, but this small variability could support our prediction model.

```
# Visualize the distribution of user rating trends
ggplot(edx_clean, aes(x = trend)) +
  geom_histogram(bins = 30, fill = "green", alpha = 0.7) +
  labs(title = "Distribution of User Rating Trends", x = "Trend", y = "Frequency") +
  theme_minimal()
```



The decade distribution below confirms the feature captures historical trends, with most ratings for movies from the 1980s, 1990s, and 2000s. This feature aims to capitalize on nostalgic movie ratings or period-based movie ratings.

```
# View and visualize decade distribution
cat("\nDecade distribution in dataset:\n")
```

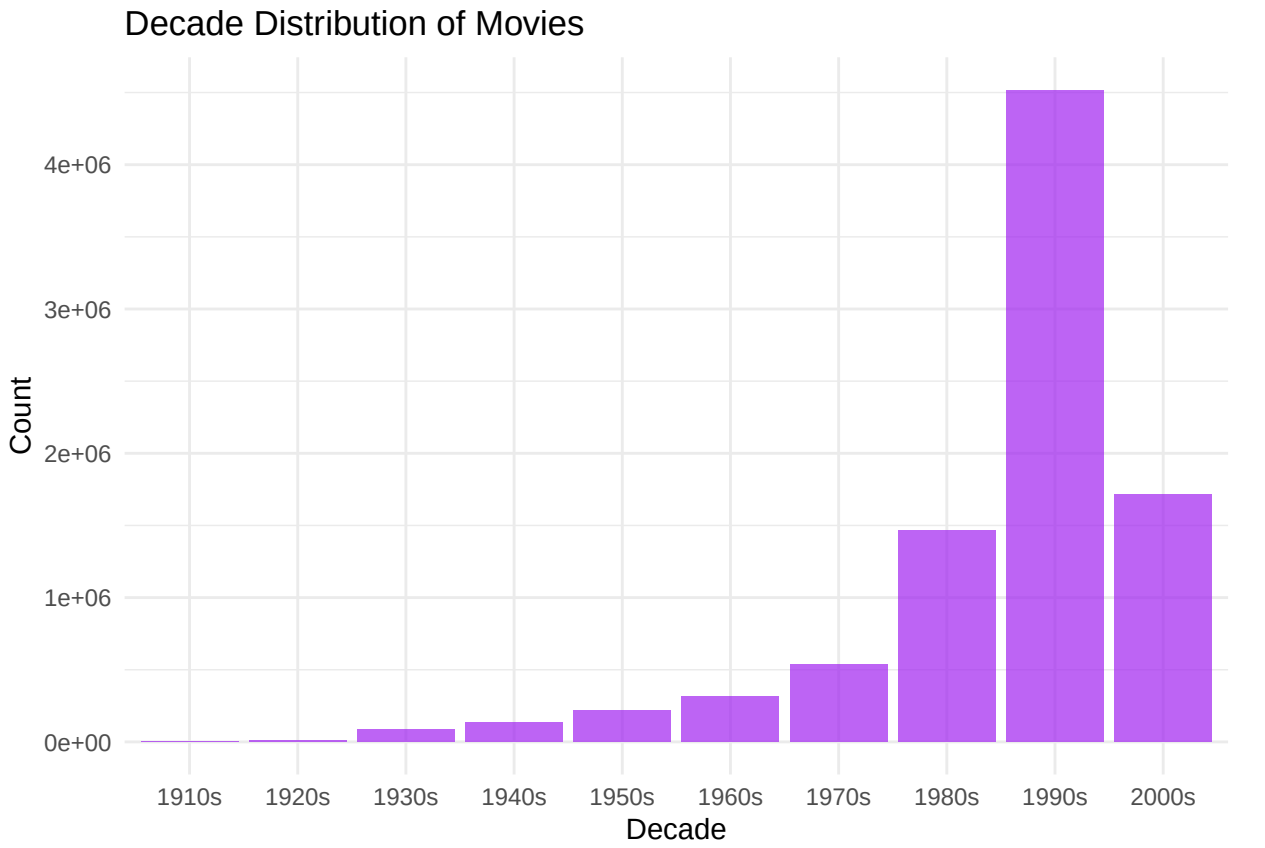
```
##
```

```
## Decade distribution in dataset:
```

```
decade_distribution <- table(edx_clean$decade)
print(decade_distribution)
```

```
##
##   1910s   1920s   1930s   1940s   1950s   1960s   1970s   1980s   1990s   2000s
##     527   12658   86040  133141  215997  316748  537132 1467705 4518239 1711868
```

```
ggplot(as.data.frame(decade_distribution), aes(x = Var1, y = Freq)) +
  geom_bar(stat = "identity", fill = "purple", alpha = 0.7) +
  labs(title = "Decade Distribution of Movies", x = "Decade", y = "Count") +
  theme_minimal()
```



The following code shows the dataset now contains 9,000,055 rows and 26 columns, providing a robust foundation for machine learning.

```
# Check the final dataset dimensions
cat("\nFinal dataset dimensions:\n")
```

```
##
## Final dataset dimensions:
```

```
cat("Rows:", nrow(edx_clean), "Columns:", ncol(edx_clean), "\n")
```

```
## Rows: 9000055 Columns: 26
```

Modeling

The fun part begins! The dataset has been cleaned and enhanced with engineered features, maximizing the utility of the 9 million rows of data within the edx_clean portion of the MovieLens dataset.

Next, the dataset is split again to serve a similar purpose as before: testing the model's accuracy on an independent dataset. This ensures its ability to predict ratings for new data. The split divides the dataset into a train_set for model training and a validation_set for evaluation.

```
# Partition the edx_clean data into training and validation sets
set.seed(1, sample.kind = "Rounding")  # set seed 1 for reproducability
train_index <- createDataPartition(y = edx_clean$rating, times = 1, p = 0.8, list = FALSE)

# Create training and validation sets
train_set <- edx_clean[train_index, ]
validation_set <- edx_clean[-train_index, ]

# Check dimensions of the training and validation sets
cat("Training set dimensions: Rows:", nrow(train_set), "Columns:", ncol(train_set), "\n")
```

```
## Training set dimensions: Rows: 7200045 Columns: 26
```

```
cat("Validation set dimensions: Rows:", nrow(validation_set), "Columns:", ncol(validation_set), "\n")
```

```
## Validation set dimensions: Rows: 1800010 Columns: 26
```

As seen above, this model will be trained on 7,200,045 rows of data across 26 variables and later it will be evaluated on 1,800,010 rows and 26 columns.

Selecting Regression For Modeling

The methodology chosen for this task is regression, where the model treats movie ratings as a continuous variable for prediction. Given the extensive amount of data available for training and the 10 possible rating levels (from 0.5 to 5.0 in 0.5 increments), regression is expected to yield highly accurate predictions. However, this comes at the cost of precision, as the model may predict ratings at arbitrary decimal points rather than adhering strictly to the 0.5-point increments that a classification approach would enforce.

Gaussian regression is particularly well-suited for this task, as it is designed to model continuous target variables effectively. To prepare the dataset for this methodology, all predictors must be numeric. This process involves removing nonessential non-numeric variables (e.g., movie titles, as the movieId already serves this purpose) and encoding essential non-numeric variables (e.g., genres, rating_day_of_week) into a numeric format.

```
# Identify numeric columns and essential categorical columns
essential_columns <- c("genres", "last_rating_date", "rating_day_of_week", "decade")
selected_columns <- union(names(train_set)[sapply(train_set, is.numeric)], essential_columns)

# Subset the training and validation datasets
train_set_numeric <- train_set[, intersect(colnames(train_set), selected_columns)]
validation_set_numeric <- validation_set[, intersect(colnames(validation_set), selected_columns)]

# Log the column names and dimensions
print(colnames(train_set_numeric))
```

```
##  [1] "userId"                "movieId"               "rating"
##  [4] "timestamp"             "genres"                "rating_year"
##  [7] "rating_month"          "rating_day_of_week"    "movie_year"
## [10] "movie_age"             "decade"                "user_rating_count"
## [13] "user_avg_rating"       "rating_duration"       "recent_rating_count"
## [16] "last_rating_date"      "avg_ratings_per_year"  "rating_std_dev"
## [19] "trend"                 "user_movie_diff"       "days_since_last_rating"
## [22] "movie_rating_count"    "movie_avg_rating"      "movie_rating_std_dev"
## [25] "years_since_release"
```

```
cat("Dimensions of train_set_numeric: Rows:", nrow(train_set_numeric), "Columns:", ncol(train_set_numeri
```

```
## Dimensions of train_set_numeric: Rows: 7200045 Columns: 25
```

```
print(colnames(validation_set_numeric))
```

```
##  [1] "userId"                "movieId"               "rating"
##  [4] "timestamp"             "genres"                "rating_year"
##  [7] "rating_month"          "rating_day_of_week"    "movie_year"
## [10] "movie_age"             "decade"                "user_rating_count"
## [13] "user_avg_rating"       "rating_duration"       "recent_rating_count"
## [16] "last_rating_date"      "avg_ratings_per_year"  "rating_std_dev"
## [19] "trend"                 "user_movie_diff"       "days_since_last_rating"
## [22] "movie_rating_count"    "movie_avg_rating"      "movie_rating_std_dev"
## [25] "years_since_release"
```

```
cat("Dimensions of validation_set_numeric: Rows:", nrow(validation_set_numeric), "Columns:", ncol(valida
```

```
## Dimensions of validation_set_numeric: Rows: 1800010 Columns: 25
```

The above show that in both the training set and the validation set there is no longer 26 variables, but 25 (removed title), all of which are now numeric. Later on in the process, this will need to be done in the Final Holdout test set as well.

The next step involves setting up the predictor matrix (x) and the target variable (y). The target variable is the movie rating, while the predictor matrix consists of all other numeric features. This setup ensures the data is ready for training the regression model.

```
# Prepare predictor matrix (x) and target variable (y) (This might take a few minutes)
x <- as.matrix(train_set_numeric[, colnames(train_set_numeric) != "rating"])
y <- train_set_numeric$rating
```

With the dataset prepared and features engineered, the next step involves training an elastic net regression model using the Gaussian family. Elastic net regression is a robust technique that combines ridge and lasso regression for regularization, addressing overfitting while retaining the most relevant features. Ridge regression applies a penalty proportional to the squared magnitude of coefficients, effectively distributing the impact across correlated variables. In contrast, lasso regression adds a penalty proportional to the absolute value of coefficients, shrinking some coefficients to zero and excluding less relevant predictors. Elastic net leverages both approaches, balancing them with the parameter alpha. For this model, alpha = 0.5 was chosen to achieve an equal balance between these two regularization techniques.

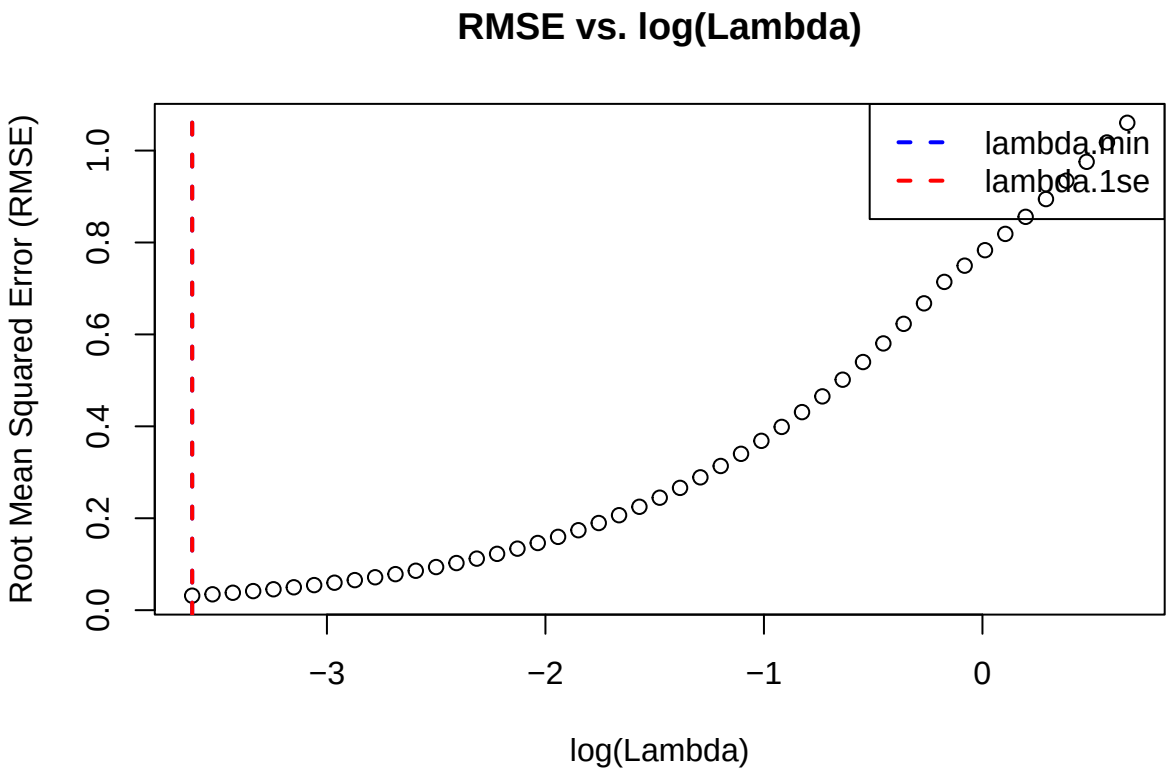
Below, cross-validation is employed to identify the optimal regularization parameter AKA lambda (which minimizes prediction error while maintaining feature interpretability).

```
# Elastic net regression with Gaussian family for continuous target
# Set alpha to 0.5 for a balance between ridge and lasso regression
elastic_net_model <- cv.glmnet(x, y, alpha = 0.5, family = "gaussian")
```

Now that the net has been cast, the approach is visualized below.

```
# Calculate RMSE from cross-validation default of MSE
rmse_values <- sqrt(elastic_net_model$cvm)

# Plot RMSE vs. log(lambda)
plot(
  log(elastic_net_model$lambda), rmse_values,
  type = "b",
  xlab = "log(Lambda)",
  ylab = "Root Mean Squared Error (RMSE)",
```

```
  main = "RMSE vs. log(Lambda)"
)

# Add vertical lines for lambda.min and lambda.1se
abline(v = log(elastic_net_model$lambda.min), col = "blue", lty = 2, lwd = 2)
abline(v = log(elastic_net_model$lambda.1se), col = "red", lty = 2, lwd = 2)

# Add a legend for the lines
legend("topright", legend = c("lambda.min", "lambda.1se"),
       col = c("blue", "red"), lty = 2, lwd = 2)
```




```
# Extract RMSE values for lambda.min and lambda.1se
lambda_min_rmse <- rmse_values[which(elastic_net_model$lambda == elastic_net_model$lambda.min)]
lambda_1se_rmse <- rmse_values[which(elastic_net_model$lambda == elastic_net_model$lambda.1se)]

# Print results
cat("RMSE for lambda.min:", lambda_min_rmse, "\n")
```

```
## RMSE for lambda.min: 0.03150301
```

```
cat("RMSE for lambda.1se:", lambda_1se_rmse, "\n")
```

```
## RMSE for lambda.1se: 0.03150301
```

The cross-validation plot generated by plot(elastic_net_model) illustrates the relationship between lambda and the model's predictive performance. The x-axis represents lambda), the logarithm of the regularization parameter. Smaller lambda values include more features in the model, while larger lambda values exclude less significant predictors by shrinking their coefficients to zero. The y-axis shows the root mean squared error (RMSE), the model's error metric during cross-validation. The curve depicts how error (RMSE) increases as lambda includes more variables.

The leftmost part of the plot (smallest lambda) represents the model with the least regularization and the largest number of features. Conversely, the rightmost part (largest lambda represents the model with the most regularization and the fewest features. As shown by the blue line (which is obscured by the red line), the minimum RMSE point identifies the lambda value that achieves the lowest prediction error, referred to as lambda_min. Some approaches use values within 1 standard deviation of lambda_min, but since both values are .0315 the lambda.min value will be used to build the final model.

The optimal lambda is determined by the lambda_min value from cross-validation. This parameter minimizes prediction error while maintaining model simplicity by excluding irrelevant features.

```
# Identify the lambda corresponding to the minimum RMSE
best_lambda_rmse <- elastic_net_model$lambda[which.min(rmse_values)]
print(paste("Best lambda (based on RMSE):", best_lambda_rmse))
```

```
## [1] "Best lambda (based on RMSE): 0.0268710815658445"
```

```
# Fit the model with the best lambda based on RMSE
final_model <- glmnet(x, y, alpha = 0.5, lambda = best_lambda_rmse)

# Print coefficients of the final model
print(coef(final_model))
```

```
## 25 x 1 sparse Matrix of class "dgCMatrix"
##                                    s0
## (Intercept)            0.149511768
## userId                 .
## movieId                .
## timestamp              .
## genres                 .
## rating_year            .
## rating_month           .
## rating_day_of_week     .
## movie_year             .
## movie_age              .
## decade                 .
## user_rating_count      .
## user_avg_rating        0.956329670
## rating_duration        .
## recent_rating_count    .
## last_rating_date       .
## avg_ratings_per_year   .
## rating_std_dev         .
## trend                  .
## user_movie_diff        0.973585752
## days_since_last_rating .
## movie_rating_count     .
## movie_avg_rating       0.001103673
## movie_rating_std_dev   .
## years_since_release    .
```

The optimal lambda minimizes cross-validated error, and the coefficients of the final model highlight the features retained after regularization. Larger coefficients indicate greater importance in predicting movie ratings. Key predictors such as user_movie_diff, user_avg_rating, and movie_avg_rating emerge as the most influential, which aligns with the objective of predicting ratings based on user and movie characteristics.

This finalized model will now be evaluated on the validation set to assess its predictive performance on unseen data.

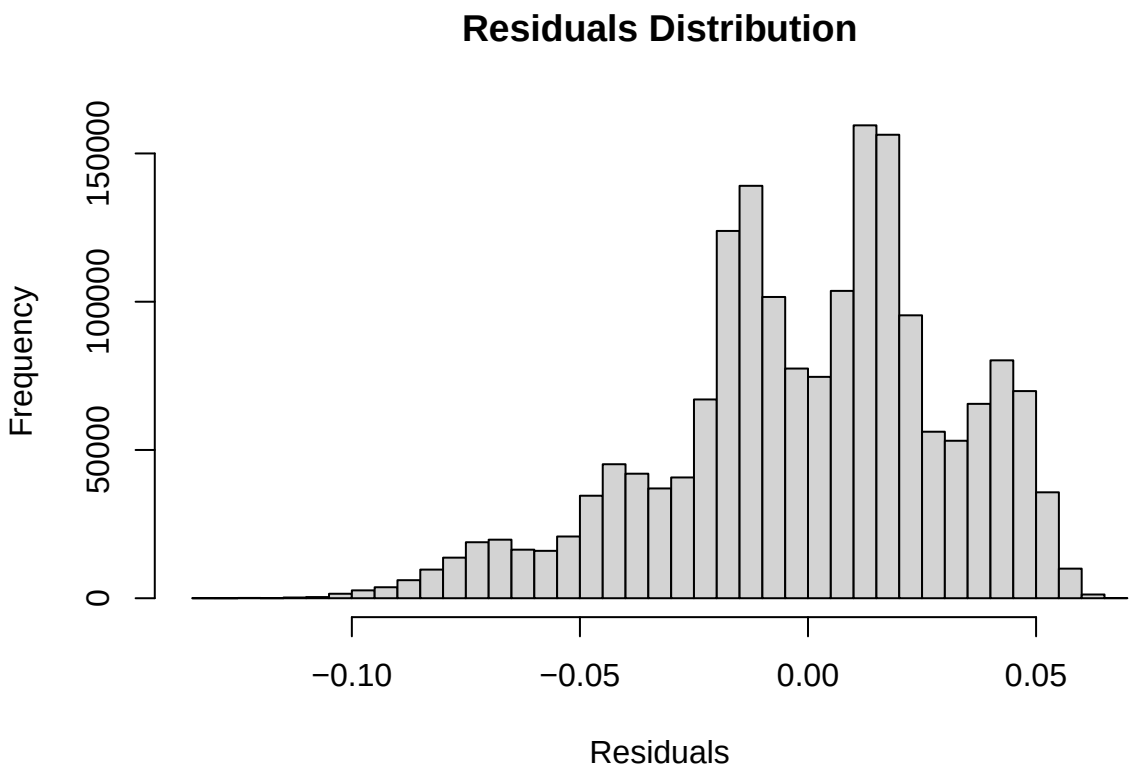
```r
# Prepare the predictor matrix (x_val) and target variable (y_val)
x_val <- as.matrix(validation_set_numeric[, colnames(validation_set_numeric) != "rating"])
y_val <- validation_set_numeric$rating


# Refit the final model using the RMSE-optimized lambda
final_model_rmse <- glmnet(x, y, alpha = 0.5, lambda = best_lambda_rmse)

# Predict ratings for the validation set using the RMSE-optimized final model
predictions <- predict(final_model_rmse, newx = x_val)

# Calculate RMSE on the validation set
rmse_validation <- sqrt(mean((predictions - y_val)^2))
cat("RMSE on Validation Set:", rmse_validation, "\n")
```

```
## RMSE on Validation Set: 0.03147069
```

The RMSE on the validation set is 0.0315, indicating that the model's predictions deviate from the actual ratings by an average of only 0.03, a strong performance given that movie ratings are provided in increments of 0.5.

Below is a visualization of the residuals of the predicted ratings:

```r
#check predicted vs actual
residuals <- y_val - predictions
hist(residuals, breaks = 50, main = "Residuals Distribution", xlab = "Residuals")
```



The residuals, representing the difference between actual and predicted ratings, are tightly clustered between -0.05 and +0.05, indicating that the model predicts movie ratings with high accuracy and minimal error. This narrow range suggests the model effectively captures the underlying patterns in the data, with no significant bias or overestimation.

With this strong result on the validation set, it's time to test the developed model on the independent final_hold set.

Results

In this section, the trained model is evaluated on the final holdout set to assess its performance on completely unseen data. The process involves applying feature engineering, setting predictor variables as numeric, making the prediction, and calculating the Root Mean Squared Error (RMSE), which serves as the primary metric for model accuracy.

Preparing the Final Holdout Set

To ensure consistency, feature engineering is applied to the final holdout set. Just like before, Numeric and essential categorical columns are retained, while unnecessary columns like title are excluded. The processed dataset is then prepared for prediction.

```
# Apply the feature engineering function to the final holdout set
final_holdout_clean <- feature_engineering(final_holdout_test)

# Ensure only numeric columns and essential categorical columns are included
essential_columns <- c("genres", "last_rating_date", "rating_day_of_week", "decade")
selected_columns <- union(names(final_holdout_clean)[sapply(final_holdout_clean, is.numeric)], essentia
final_holdout_numeric <- final_holdout_clean[, intersect(colnames(final_holdout_clean), selected_columns

# Remove unnecessary columns like 'title'
final_holdout_numeric <- final_holdout_numeric[, colnames(train_set_numeric)]

# Verify dimensions and alignment
cat("Number of columns in final holdout numeric dataset:", ncol(final_holdout_numeric), "\n")
```

```
## Number of columns in final holdout numeric dataset: 25
```

The processed final holdout dataset ensures compatibility with the trained elastic net regression model, maintaining alignment in the number of predictors (25).

The predictor matrix and target variable are prepared from the processed final holdout set. Predictions are generated using the trained model, and RMSE is calculated to quantify the model's accuracy.

```
# Prepare the predictor matrix and target variable
x_holdout <- as.matrix(final_holdout_numeric[, colnames(final_holdout_numeric) != "rating"])
y_holdout <- final_holdout_numeric$rating

# Ensure dimensions match before predicting
if (ncol(x_holdout) == length(coef(final_model)) - 1) {
  # Predict ratings for the final holdout set using the trained model
  predictions_holdout <- predict(final_model, newx = x_holdout)

  # Calculate RMSE on the final holdout set
  rmse_holdout <- sqrt(mean((predictions_holdout - y_holdout)^2))
  cat("RMSE on Final Holdout Set:", rmse_holdout, "\n")
}
```

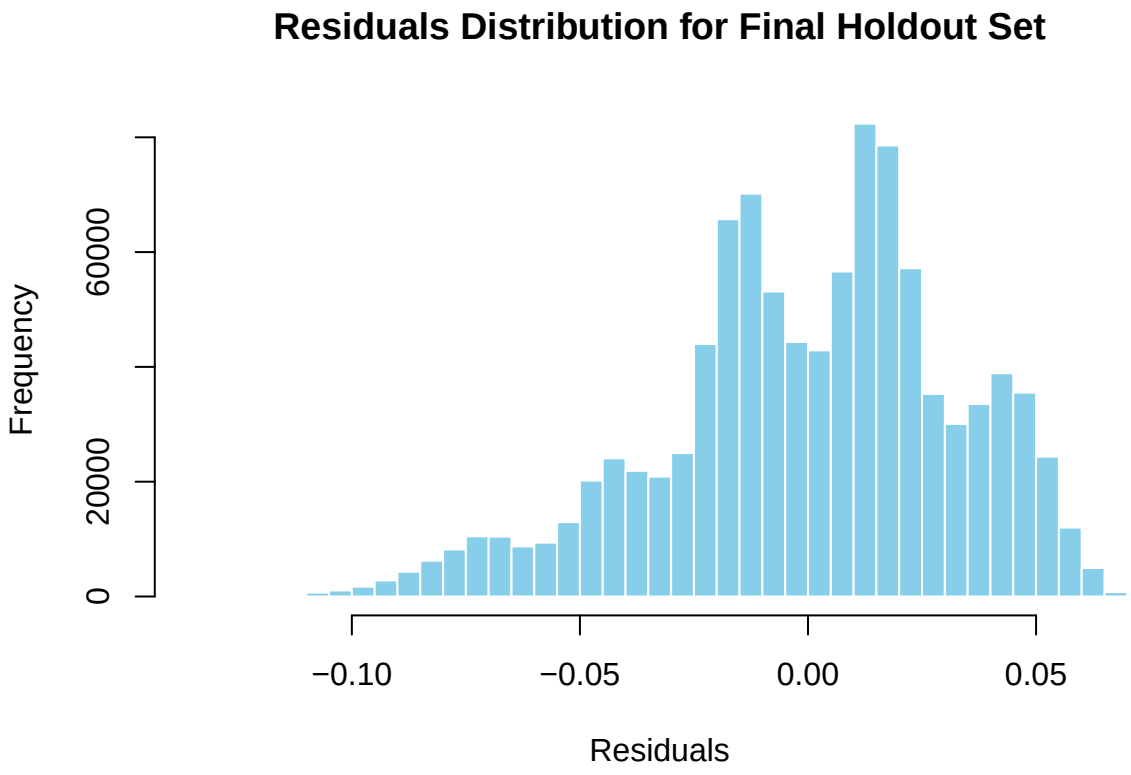
```
## RMSE on Final Holdout Set: 0.03262489
```

Output

The RMSE calculated on the final holdout was ~.03262. This is a negligible increase from the models performance on the validation set .03147. As a result, we can conclude this model provides a robust measure of how accurately the model predicts movie ratings for unseen data.

Like the grey residuals histogram generated for the validation_set performance, the predictions on the final holdout set fell within +.05 and - .05 for the majoirty of predictions. This results shows the strength of the model by way of predictions being between 1 rating level.

```
# Calculate residuals for the final holdout set
residuals_holdout <- y_holdout - predictions_holdout

# Visualize the residuals
hist(
  residuals_holdout,
  breaks = 50,
  main = "Residuals Distribution for Final Holdout Set",
  xlab = "Residuals",
  col = "skyblue",
  border = "white"
)
```



Top 10 Largest Errors

The table below shows that the largest mistakes were .13 off and these mistakes don't appear to be in any particular genre or decade, suggesting no absolutely necessary adjustments are needed.

```
#Calculate residuals and absolute residuals
residuals_holdout <- y_holdout - predictions_holdout
absolute_residuals <- abs(residuals_holdout)
```

```r
# Find the top 10 largest errors
largest_errors <- order(absolute_residuals, decreasing = TRUE)[1:10]

# Create a table with movie details and error metrics for the largest errors
largest_errors_table <- data.frame(
  final_holdout_clean[largest_errors, c("movieId", "title", "genres")], # Movie details
  Actual = y_holdout[largest_errors],                                    # Actual ratings
  Predicted = predictions_holdout[largest_errors],                       # Predicted ratings
  Residual = residuals_holdout[largest_errors]                           # Residuals
)

# Print the table
cat("Details of Movies with the Largest Prediction Errors:\n")
```

```
## Details of Movies with the Largest Prediction Errors:
```

```r
print(largest_errors_table)
```

```
##        movieId                                    title
## 673831     527                  Schindler's List (1993)
## 344584    1252                        Chinatown (1974)
## 166817    2571                     Matrix, The (1999)
## 120822    1233              Boat, The (Das Boot) (1981)
## 673834    1278                Young Frankenstein (1974)
## 697576    1304 Butch Cassidy and the Sundance Kid (1969)
## 822207      58         Postman, The (Postino, Il) (1994)
## 728453    1090                           Platoon (1986)
## 294776    2336                        Elizabeth (1998)
## 728454    5060             M*A*S*H (a.k.a. MASH) (1970)
##                                  genres Actual Predicted   Residual
## 673831                       Drama|War    0.5 0.6324924 -0.1324924
## 344584 Crime|Film-Noir|Mystery|Thriller    0.5 0.6323843 -0.1323843
## 166817           Action|Sci-Fi|Thriller    0.5 0.6323431 -0.1323431
## 120822                Action|Drama|War    0.5 0.6322682 -0.1322682
## 673834             Action|Comedy|Horror    0.5 0.6321341 -0.1321341
## 697576            Action|Comedy|Western    0.5 0.6321072 -0.1321072
## 822207             Comedy|Drama|Romance    0.5 0.6320696 -0.1320696
## 728453                       Drama|War    0.5 0.6320066 -0.1320066
## 294776                           Drama    0.5 0.6319966 -0.1319966
## 728454                Comedy|Drama|War    0.5 0.6319901 -0.1319901
```

Discussion

This report has loaded, cleaned, and developed features for a training set of the MovieLens 10million dataset. Afterward, features were turned into numeric types to prepare for regression modeling. Next, a machine learning algorithm was trained using Gaussian regression with an elastic net technique. Once trained, the model produced an RMSE of ~.0315 on the validation set. Finally, the model was used to predict movies ratings on the Final Holdout set and produced an RMSE of ~.0326. This results confirms that the model effectively predicts movie ratings within a half star out of 5 stars. Therefore, this model is an effective tool for predicting movielens movie ratings.

Limtations

Despite extensive preprocessing, the feature engineering pipeline heavily relies on the absence of missing values in the dataset. If applied to new, uncleaned datasets, the feature creation function may fail due to NA values or unexpected data formats. Enhancing robustness through NA handling and input validation would improve the model's scalability.

The Gaussian regression model allowed for continuous predictions, providing a higher degree of precision in estimating movie ratings. However, this approach diverges from the discrete nature of the actual rating system, which operates in 0.5-star increments. While the model's continuous predictions (e.g., 3.6 stars) may offer more granularity, they are not valid within the constraints of the original system and may require rounding or mapping to the closest valid rating for practical applications.

The dataset's historical nature introduces data imbalance, particularly for older movies with fewer ratings. This imbalance may reduce prediction accuracy for specific subsets, such as niche or vintage films. Moreover, temporal effects, such as seasonal spikes in viewership (e.g., horror films during Halloween), were not explicitly modeled, potentially leaving valuable information untapped.

Future Directions

To address the limitation of invalid rating predictions, future iterations could explore classification models. These would predict discrete rating categories (e.g., 0.5-star increments) rather than continuous values. Techniques such as ordinal regression or multi-class classification with softmax outputs could be effective for this purpose.

The model currently incorporates basic user-specific features such as average ratings and rating trends. Expanding this feature set to include interactions with genres, time of year (e.g., holidays), or platform activity patterns (e.g., binge-watching behavior) could improve predictive performance and provide deeper insights into user preferences.

Feature engineering can be further expanded to include embedding for movies, genres, or users using techniques like collaborative filtering or deep learning approaches (e.g., matrix factorization or neural networks). These methods could uncover latent factors influencing ratings, yielding a more robust and scalable prediction system.

References

Durrani, A. (2024, August 15). Top streaming statistics in 2024. Forbes. https://www.forbes.com/home-improvement/internet/streaming-stats/

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19. https://doi.org/10.1145/2827872

Jesse Vig, Shilad Sen, and John Riedl. 2012. The Tag Genome: Encoding Community Knowledge to Support Novel Interaction. ACM Trans. Interact. Intell. Syst. 2, 3: 13:1–13:44. https://doi.org/10.1145/2362394.2362395

Link to MovieLens dataset 10mil: https://grouplens.org/datasets/movielens/10m/