Final_Capstone_Project_MovieLens

Kavish Seechurn

Sys.Date()

```
{r setup, include=FALSE} knitr::opts_chunk$set(echo = TRUE)
```

Movie Recommendation System | Machine Learning Capstone Project

HarvardX: PH125.9x Data Science Capstone

1. Introduction

Background and Motivation

Ever wonder how websites like Amazon or Netflix suggest movies you might enjoy? These suggestions come from a special kind of system called a **recommender system.**

Imagine a recommender system as a friend who knows your tastes really well. Based on your past choices, the system predicts which new items (in this case, movies) you'd likely be interested in. It does this by analyzing data about you and other users with similar preferences.

Recommender systems are used everywhere from online shopping to music streaming services. They even help suggest research articles, news stories, or even potential romantic partners on dating sites!

Big companies like Netflix rely heavily on recommender systems. In fact, Netflix once offered a million-dollar prize to anyone who could improve their system's accuracy by just 10%! It's important to note that Netflix wasn't necessarily trying to predict your exact rating for a movie. Their goal was to suggest movies you'd likely enjoy, even if you didn't end up giving it a perfect score.

This project focuses on building a movie recommender system, similar to what Netflix uses. While our goal might not be to win a million dollars, we can still learn a lot about how these systems work and how to make movie recommendations more personalized.

Data Loading

```{r results = 'hide', message = FALSE, warning = FALSE} if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org") if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org") if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org") if(!
```

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

library(tidyverse) library(caret) library(data.table) library(recosystem) library(dplyr)
```

MovieLens 10M dataset:

https://grouplens.org/datasets/movielens/10m/

http://files.grouplens.org/datasets/movielens/ml-10m.zip

```
options(timeout = 120)

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

ratings <- fread(text = gsub("::", "", readLines(unzip(dl,"ml-10M100K/ratings.dat"))),
col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId), title =
as.character(title), genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")
```

Validation set will be 10% of MovieLens data

```
set.seed(1, sample.kind="Rounding") test_index <- createDataPartition(y =
movielens$rating, times = 1, p = 0.1, list = FALSE) edx <- movielens[-test_index,] temp <-
movielens[test_index,]
```

Make sure userId and movieId in validation set are also in edx set

```
validation <- temp %>% semi_join(edx, by = "movieId") %>% semi_join(edx, by = "userId")
```

Add rows removed from validation set back into edx set

```
removed <- anti_join(temp, validation) edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

```
### Training and Testing of the Algorithm
The 10M dataset is split into training (edx) and testing sets.
Training is further divided for building and testing the model. Final
evaluation uses the validation set.

# Create train and test sets
```{r, include=TRUE, echo=TRUE}
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1,
list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

# Matching userId and movieId in both train and test sets
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Adding back rows into train set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)
```

Navigating the Data

The **data** contains nine million movies ratings from users. Each rating has information like the user who rated it, the movie itself, the rating they gave, when they rated it, the movie's title, and its genres.

```
{r, include=TRUE, echo=TRUE} str(edx)
```

The summary function provides a statistical summary of the data.

```
edx %>% select(-genres) %>% summary()
```

There are 69,878 unique users and 10,677 movies.

```
edx %>% summarize(n_users = n_distinct(userId), n_movies =
n_distinct(movieId))
```

The most popular movies in this dataset.

```
{r fig.align='center'} edx %>%   group_by(title) %>%   summarize(count
= n()) %>%   arrange(-count) %>%   top_n(20, count) %>%
ggplot(aes(count, reorder(title, count))) +   geom_bar(color =
"black", fill = "green", stat = "identity") +   xlab("Count") +
ylab(NULL) +   theme_bw()
```
```

The distribution of the movie ratings shows a range of 0.5 to 5 with whole numbers used more often.

```
{r fig.align='center'} edx %>%   ggplot(aes(rating, y = ..prop..)) +
geom_bar(color = "black", fill = "green") +   labs(x = "Ratings", y =
"Relative Frequency") +   scale_x_continuous(breaks = c(0.5, 1, 1.5,
2, 2.5, 3, 3.5, 4, 4.5, 5)) +   theme_bw()
```

The number of ratings VS users shows a right skew in its distribution.

```
{r fig.align='center'} edx %>% group_by(userId) %>%   summarize(count
= n()) %>%   ggplot(aes(count)) +   geom_histogram(color = "black",
fill = "green", bins = 40) +   xlab("Ratings") +   ylab("Users") +
scale_x_log10() +   theme_bw()
```

Approach and Evaluation

Initialise an empty table for storing RMSE results

```
{r results = 'hide', message = FALSE, warning = FALSE} rmse_table <-
data.frame(Method = character(), Model_Name = character())
```

##Model 1 - Baysesian # Define Bayesian Average function

```{r echo=TRUE} bayesian_average <- function(x, mu, lambda) { prior_mean <- mean(x) prior_variance <- var(x) posterior_mean <- (lambda * prior_mean + mu * sum(x)) / (lambda + length(x)) posterior_variance <- (prior_variance / lambda + sum((x - mu)^2)) / (lambda + length(x) - 1) return(list(mean = posterior_mean, variance = posterior_variance)) }

## Calculate global average rating (can be adjusted if needed)

global_avg_rating <- mean(edx$rating)

## Tuning parameter for Bayesian Average, adjust as needed

lambda = 10 results_reco <- 0 # Apply Bayesian Average to predictions bayesian_predictions <- mapply(bayesian_average, results_reco, MoreArgs = list(mu = global_avg_rating, lambda = lambda), SIMPLIFY = FALSE) bayesian_predictions <- unlist(lapply(bayesian_predictions, function(x) x$mean))

## Calculate RMSE using Bayesian Average predictions

rmse_bayesian <- sqrt(mean((bayesian_predictions - test_set$rating)^2)) cat("RMSE using Bayesian Average:", rmse_bayesian, "") rmse_table <- rbind(rmse_table, data.frame(Method = "Bayesian Average", Model_Name = "Model 1", RMSE = rmse_bayesian)) cat("Method Model Name RMSE") print(rmse_table)
```

Model 2: Matrix Factorization with Recosystem
```{r echo=TRUE}
set.seed(1, sample.kind = "Rounding")

# Create Recosystem data objects with descriptive names
train_data_reco <- with(train_set, data_memory(user_index = userId,
item_index = movieId, rating = rating))
test_data_reco <- with(test_set, data_memory(user_index = userId,
item_index = movieId, rating = rating))

# Descriptive model name
recommender_model <- Reco()

# Tuning parameters
tuning_results <- recommender_model$tune(train_data_reco, opts =
list(dim = c(20, 30),

costp_l2 = c(0.01, 0.1),

costq_l2 = c(0.01, 0.1),

lrate = c(0.01, 0.1),

nthread = 4,

niter = 10))

# Train the model with descriptive object names
recommender_model$train(train_data_reco, opts = c(tuning_results$min,
nthread = 4, niter = 30))

# Generate predictions with descriptive names
matrix_factorization_predictions <-
recommender_model$predict(test_data_reco, out_memory())

# Calculate and print RMSE
matrix_factorization_rmse <- RMSE(matrix_factorization_predictions,
test_set$rating)
cat("RMSE using Matrix Factorization with Recosystem:",
matrix_factorization_rmse, "\n")
rmse_table <- rbind(rmse_table, data.frame(Method = "Matrix
Factorization", Model_Name = "Model 2", RMSE =
matrix_factorization_rmse))
cat("\n")  # Add an empty line for visual separation
print(rmse_table)
```

###Model 3 - Collaborative Filtering techniques Unfortunately I had too many issues though I wanted to try this method ```{r echo=TRUE} rmse_table <- rbind(rmse_table,

data.frame(Method = “Collaborative Filtering techniques”, Model_Name = “Model 3”, RMSE = NA))

Print the table after Model 3

cat(“”) # Add another empty line print(rmse_table)

```
###Model 4 -  Means + Bias
```{r echo=TRUE}
mean_training <- mean(train_set$rating)
naive_rmse <- RMSE(test_set$rating, mean_training)
cat("RMSE using Means:", naive_rmse, "\n")

bias <- train_set %>%
        group_by(movieId) %>%
        summarize(bias_i = mean(rating - mean_training))

bias %>% ggplot(aes(bias_i)) +
  geom_histogram(color = "black", fill = "green", bins = 10) +
  xlab("Movie Bias") +
  ylab("Count") +
  theme_bw()


predicted_ratings <- mean_training + test_set %>%
  left_join(bias, by = "movieId") %>%
  pull(bias_i)
mean_bias_rmse <- RMSE(predicted_ratings, test_set$rating)

cat("RMSE using Mean + Bias", mean_bias_rmse, "\n")
rmse_table <- rbind(rmse_table, data.frame(Method = "RMSE using Mean +
Bias", Model_Name = "Model 4", RMSE = mean_bias_rmse))
cat("\n")  # Add an empty line for visual separation
print(rmse_table)
```

###Final Model with testing against test data set ```{r echo=TRUE} set.seed(1, sample.kind = “Rounding”)

Create Recosystem data objects with descriptive names for edx and validation sets

edx_data_reco <- with(edx, data_memory(user_index = userId, item_index = movieId, rating = rating)) validation_data_reco <- with(validation, data_memory(user_index = userId, item_index = movieId, rating = rating))
```

Descriptive model name

final_recommender_model <- Reco()

Tuning parameters

final_tuning_results <- final_recommender_model$tune(edx_data_reco, opts = list(dim = c(20, 30), costp_l2 = c(0.01, 0.1), costq_l2 = c(0.01, 0.1), lrate = c(0.01, 0.1), nthread = 4, niter = 10))

Train the final model with descriptive object names

final_recommender_model$train¿min, nthread = 4, niter = 30))

Generate final predictions with descriptive names

final_validation_predictions <- final_recommender_model$predict(validation_data_reco, out_memory())

Calculate and print final RMSE

final_validation_rmse <- RMSE(final_validation_predictions, validation$rating) cat("RMSE using Matrix Factorization with Recosystem on Validation Set:", final_validation_rmse, "")

rmse_table <- rbind(rmse_table, data.frame(Method = "Matrix Factorization (Validation)", Model_Name = "Final Validation", RMSE = final_validation_rmse))

Print the final table with all results

cat("") # Add another empty line print(rmse_table)

```
###Selection
Based on the table, the Matrix Factorization with Recosystem is better
and will be used with the final out data set as per below



####### Predicting with final_holdout_test
```{r echo=TRUE}
source("Capstone_Project.R")
final_holdout_test_reco <- with(final_holdout_test,
data_memory(user_index = userId, item_index = movieId, rating =
```
```

```
rating))

final_holdout_predictions <-
final_recommender_model$predict(final_holdout_test_reco, out_memory())
final_holdout_rmse <- RMSE(final_holdout_predictions,
final_holdout_test$rating)
cat("RMSE on Final Holdout Set:", final_holdout_rmse, "\n")

rmse_table <- rbind(rmse_table, data.frame(Method = "Matrix
Factorization (Holdout Set)", Model_Name = "Final Validation", RMSE =
final_holdout_rmse))
print(rmse_table)
```

3. Conclusion: Evaluating Model Performance and Selecting the Best Approach

The final RMSE is **0.78** thanks to the Matrix Factorization.

Our exploration of various recommender systems has yielded valuable insights into user preferences within the MovieLens dataset. We employed a rigorous approach, building and testing several models to identify the most effective recommendation strategy.

Based on the implemented models (replace with the specific models you tested, e.g., Bayesian Average, Collaborative Filtering, etc.), our analysis revealed that Matrix Factorization achieved the best performance on the validation set, as measured by Root Mean Squared Error (RMSE).

Here's a breakdown of the key factors contributing to Matrix Factorization's success in this scenario:

Reduced dimensionality: Matrix Factorization decomposes the user-item rating matrix into two lower-dimensional matrices, capturing latent factors influencing user preferences and item features. This approach can be particularly efficient for sparse datasets like MovieLens. **Scalability:** Matrix Factorization scales well with large datasets due to its efficient implementation within the recosystem package. This makes it a suitable choice for real-world recommendation systems with potentially millions of users and items. **Interpretability:** While not the primary focus here, Matrix Factorization allows some level of interpretation of the latent factors, providing insights into user and item characteristics associated with high ratings. This selection process emphasizes the importance of model testing and evaluation. Evaluating various models on a validation set before deploying them on unseen data ensures we can select the most accurate and effective approach for the specific problem.

###Future Considerations:

Hyperparameter Tuning: Further optimization of Matrix Factorization hyperparameters, such as dimensionality and regularization coefficients, could potentially improve performance. **Alternative Metrics:** Exploring additional evaluation metrics beyond RMSE,

such as precision or recall, could provide a more nuanced understanding of model effectiveness depending on the specific application. **Hybrid Approaches:** Combining Matrix Factorization with other techniques, like content-based filtering or incorporating additional user and item information, might offer further improvements in recommendation accuracy. By continually evaluating and refining our recommender system, we can ensure it delivers the most relevant and engaging movie recommendations for our users.