Data612_Final_Project

Anna Moy & Natalie Kalukeerthie

2025-07-05

```r
#Load library
library(sparklyr)
```

```
##
## Attaching package: 'sparklyr'

## The following object is masked from 'package:stats':
##
##     filter
```

```r
library(dplyr)
```

```
##
## Attaching package: '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(stringr)
library(tidytext)
library(knitr)
library(purrr)
```

```
##
## Attaching package: 'purrr'

## The following object is masked from 'package:sparklyr':
##
##     invoke
```

```r
library(tm)
```

```
## Loading required package: NLP
```

Introduction

In this project we will be building a hybrid movie recommendation systems using the Movielens 100k dataset from [Kaggle - MovieLens 100k] (https://www.kaggle.com/datasets/abhikjha/movielens-100k). Our dataset contains 100k ratings, 3,600 tags, 9k movies and 600 users.

- Ratings - userid, movieid, rating, timestamp
- Movies - movieid, title, genres
- Tags - userid, movieid, tag, timestamp

Our approach combines collaborative filtering with the Alternative Least Squares (ALS) algorithm to personalize recommendations based on user ratings. We also incorporate content-based filtering, which recommends movies using descriptive features like genres and user-generated tags.

Additionally, we evaluate the quality of recommendations using diversity and novelty metrics. Diversity measures how varied the recommended movies are, ensuring users get a broad range of different types of films. Novelty assesses how fresh or surprising the recommendations are, helping balance popular movies with lesser-known options.

We combined movie genres with user-generated tags to build richer content features for content-based filtering. While genres provide broad categories (e.g., "Action", "Comedy"), tags offer more nuanced descriptors like "quirky", "mind-bending", or "strong female lead." This hybrid of structured and unstructured features improves similarity comparisons between movies.

To prepare the text for cosine similarity, we:

- Removed missing values (i.e., movies without tags were assigned empty strings),
- Tokenized text into individual words (unigrams),
- Constructed a document-term matrix to represent movies in vector space,
- Calculated pairwise cosine similarity for movie recommendations and diversity scoring.

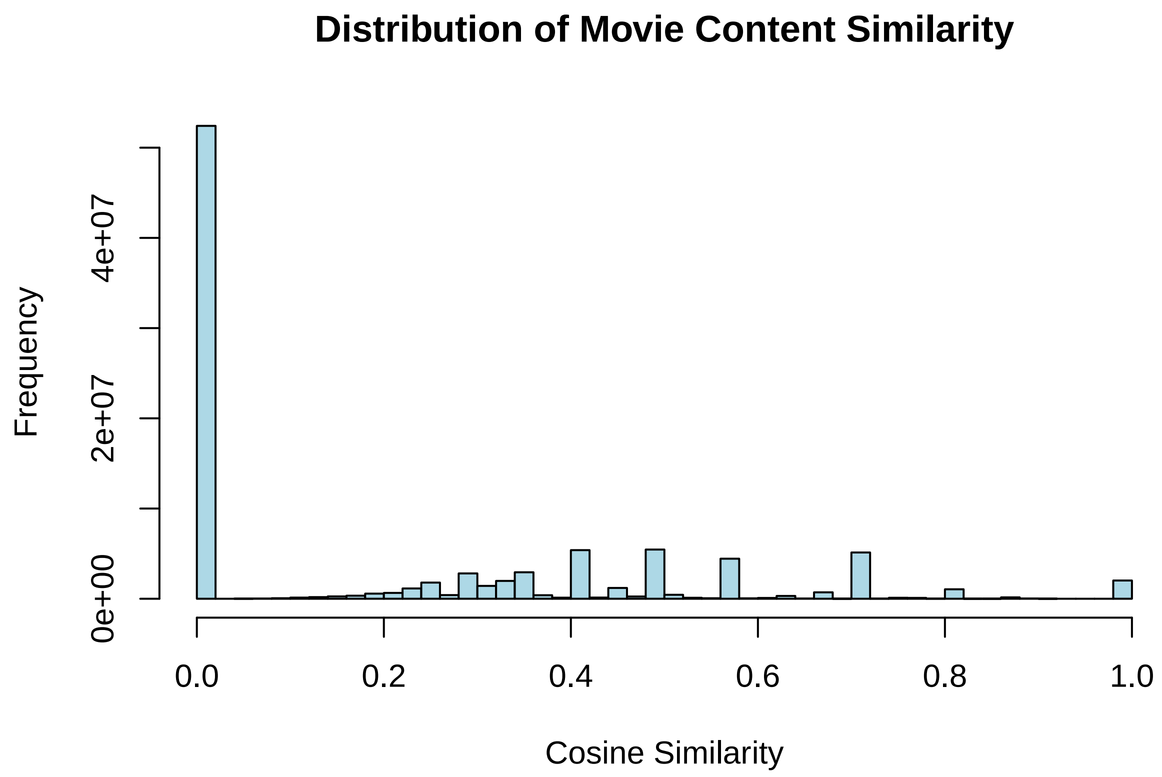
```r
sc <- spark_connect(master = "local")

# 1. Load the csv files locally to the desktop
ratings <- read.csv("/Users/zhianna/Desktop/School/Data612/ratings.csv")
movies <- read.csv("/Users/zhianna/Desktop/School/Data612/movies.csv")
tags <- read.csv("/Users/zhianna/Desktop/School/Data612/tags.csv")
```

```r
# === Process tags: combine all tags per movie === This will concatenate all the unique tags for each m
movie_tags <- tags %>%
  group_by(movieId) %>%
  summarise(all_tags = paste(unique(tag), collapse = " ")) %>%
  ungroup()

# Join tags with movies file
# movie file now contain - movieid, title, genres, all tags
movies <- movies %>%
  left_join(movie_tags, by = "movieId")

# Replace NA tags with empty string
movies$all_tags[is.na(movies$all_tags)] <- ""

# Combine genres and tags into one text feature and label it as content (genre + tags) column
#(movieid, title, genres, tags, content)
movies <- movies %>%
  mutate(content = paste(genres, all_tags, sep = " "))
```

```r
# Create a tokenized version for text features (genres + tags)
# Count how many times a word appears under the content column (genres + tags) for each movie
movie_words <- movies %>%
  select(movieId, content) %>%
  unnest_tokens(word, content)

# Create document-term matrix for content features
# Covert the counts into matrix rows are movies and columns are words and values are counts
dtm <- movie_words %>%
  count(movieId, word) %>%
  cast_dtm(document = movieId, term = word, value = n)

# Convert dtm to a matrix for cosine similarity (content-based filtering)
# Consine similarity measures how close two movies are based on genre and tag words
# Returns a symmetric matrix of similarity scores between all pairs of movies
content_matrix <- as.matrix(dtm)

cosine_sim <- function(x) {
  sim <- x %*% t(x)
  norm <- sqrt(rowSums(x * x))
  sim / (norm %*% t(norm))
}

content_sim_matrix <- cosine_sim(content_matrix)

summary(as.vector(content_sim_matrix))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0000  0.2161  0.4082  1.0000
```

```r
hist(as.vector(content_sim_matrix), breaks = 50, main = "Distribution of Movie Content Similarity",
     xlab = "Cosine Similarity", col = "lightblue")
```



```r
# Preparing Data for Spark
# Copy ratings to Spark
ratings_tbl <- copy_to(sc, ratings, "ratings", overwrite = TRUE)

# Create numeric movie IDs for ALS (Spark ALS needs numeric item IDs)
movie_map <- ratings_tbl %>%
  distinct(movieId) %>%
  arrange(movieId) %>%
  collect() %>%
  mutate(movie_numeric_id = row_number())

movie_map_tbl <- copy_to(sc, movie_map, "movie_map", overwrite = TRUE)

# Join numeric IDs back into ratings in Spark
ratings_numeric_tbl <- ratings_tbl %>%
  inner_join(movie_map_tbl, by = "movieId") %>%
  select(userId, movie_numeric_id, rating)

# Split data into training (80%) and test (20%)
set.seed(123)
splits <- ratings_numeric_tbl %>% sdf_random_split(training = 0.8, test = 0.2)
training_tbl <- splits$training
test_tbl <- splits$test

# Train ALS model
als_model <- training_tbl %>%
  ml_als(
    rating_col = "rating",
    user_col = "userId",
    item_col = "movie_numeric_id",
    rank = 10,
    max_iter = 15,
    reg_param = 0.1,
    cold_start_strategy = "drop"
  )

# Predict on test set
predictions <- ml_predict(als_model, test_tbl)

# Evaluate model with RMSE
evaluator <- ml_regression_evaluator(sc,
                                     label_col = "rating",
                                     prediction_col = "prediction",
                                     metric_name = "rmse")
rmse <- ml_evaluate(evaluator, predictions)

# Evaluate MAE
preds_local <- predictions %>% collect()
mae <- mean(abs(preds_local$rating - preds_local$prediction), na.rm = TRUE)

# Calculate MSE from RMSE
mse <- rmse^2

# Create results table
results_tbl <- tibble(
  Metric = c("RMSE", "MAE", "MSE"),
  Value = c(rmse, mae, mse)
)


# Print the table nicely
kable(results_tbl, digits = 4, caption = "ALS Model Evaluation Metrics")
```

Table 1: ALS Model Evaluation Metrics

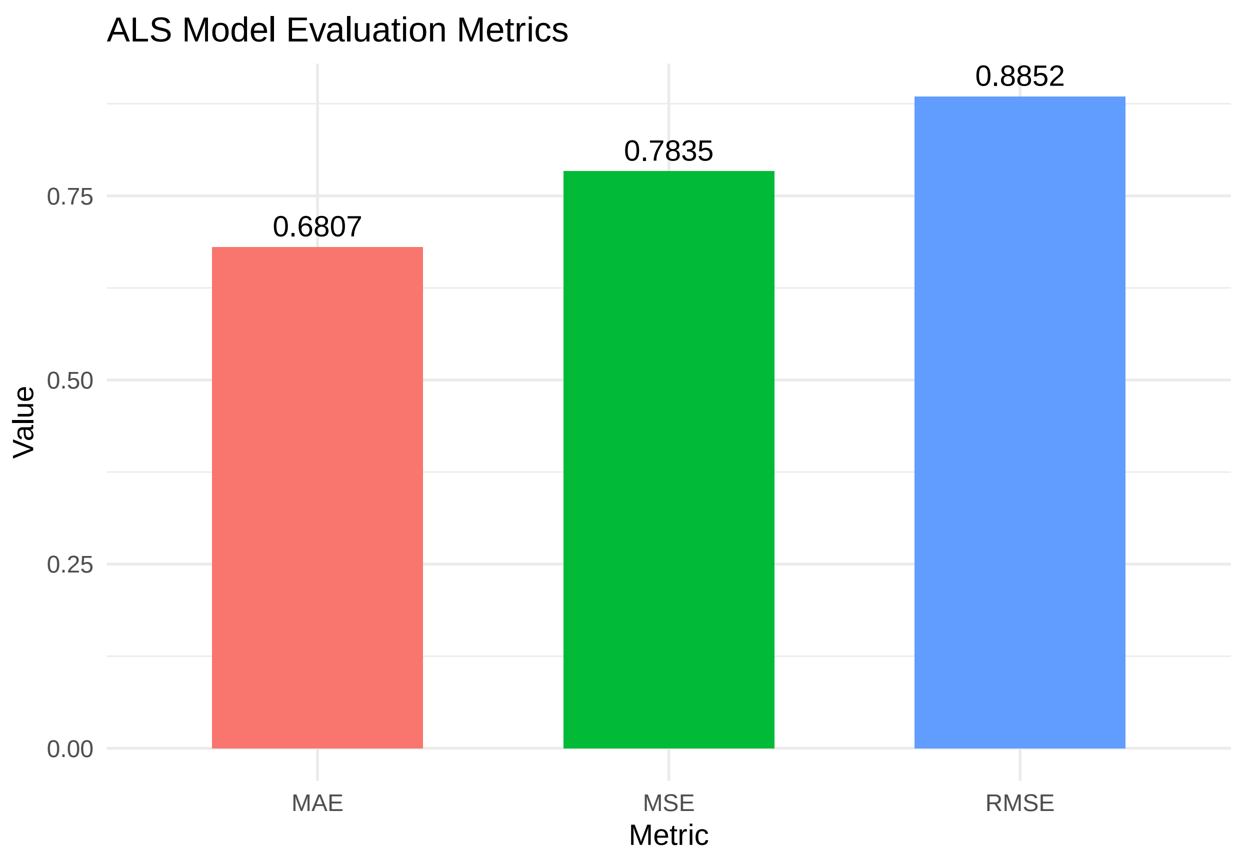
Metric	Value
RMSE	0.8852
MAE	0.6807
MSE	0.7835

```r
# Visualize RMSE, MAE, MSE
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:NLP':
```

```
##
##     annotate
```

```
ggplot(results_tbl, aes(x = Metric, y = Value, fill = Metric)) +
  geom_col(width = 0.6, show.legend = FALSE) +
  geom_text(aes(label = round(Value, 4)), vjust = -0.5) +
  labs(
    title = "ALS Model Evaluation Metrics",
    y = "Value",
    x = "Metric"
  ) +
  theme_minimal()
```



```
# === Step 1: Get ALS model recommendations and collect ===
top_recs <- ml_recommend(als_model, type = "items") %>%
  collect()

# === Step 2: Drop conflicting columns before unnesting ===
top_recs_clean <- top_recs %>%
  select(-movie_numeric_id)

# === Step 3: Unnest recommendations and rename ===
top_10_recs <- top_recs_clean %>%
  mutate(recommendations = map(recommendations, as.data.frame)) %>%
  unnest(recommendations, names_sep = "_") %>%
```

```r
  rename(
    movie_numeric_id = recommendations_movie_numeric_id,
    predicted_rating = recommendations_rating
  )

# === Step 4: Select top 10 recommendations per user ===
top_10_recs <- top_10_recs %>%
  group_by(userId) %>%
  arrange(desc(predicted_rating), .by_group = TRUE) %>%
  slice_head(n = 10) %>%
  ungroup()

# === Step 5: Join back original movieId ===
top_10_recs <- top_10_recs %>%
  left_join(movie_map, by = "movie_numeric_id")

# === Step 6: Build content similarity matrix for recommended movies ===
movies_filtered <- movies %>%
  filter(movieId %in% top_10_recs$movieId)

movie_words <- movies_filtered %>%
  select(movieId, content) %>%
  unnest_tokens(word, content)

dtm <- movie_words %>%
  count(movieId, word) %>%
  cast_dtm(document = movieId, term = word, value = n)

content_matrix <- as.matrix(dtm)

cosine_sim <- function(x) {
  sim <- x %*% t(x)
  norm <- sqrt(rowSums(x * x))
  sim / (norm %*% t(norm))
}

content_sim_matrix <- cosine_sim(content_matrix)

# === Step 7: Define diversity calculation function ===
calculate_diversity <- function(movie_ids, sim_matrix) {
  movie_ids <- as.character(movie_ids)

  # Keep only movie_ids present in sim_matrix
  valid_ids <- movie_ids[movie_ids %in% rownames(sim_matrix)]

  if (length(valid_ids) <= 1) return(NA)

  sim_vals <- combn(valid_ids, 2, function(x) sim_matrix[x[1], x[2]], simplify = TRUE)
  1 - mean(sim_vals, na.rm = TRUE)
}

# === Step 8: Calculate diversity score per user ===
diversity_scores <- top_10_recs %>%
  group_by(userId) %>%
  summarise(diversity = calculate_diversity(movieId, content_sim_matrix), .groups = "drop")

# === Step 9: Calculate novelty ===
movie_popularity <- ratings %>%
  group_by(movieId) %>%
  summarise(rating_count = n(), .groups = "drop")

top_10_with_pop <- top_10_recs %>%
  left_join(movie_popularity, by = "movieId")

novelty_scores <- top_10_with_pop %>%
  group_by(userId) %>%
  summarise(novelty = mean(1 / (1 + rating_count), na.rm = TRUE), .groups = "drop")

# === Step 10: Combine and summarize ===
user_metrics <- left_join(diversity_scores, novelty_scores, by = "userId")

summary_tbl <- user_metrics %>%
  summarise(
    Avg_Diversity = mean(diversity, na.rm = TRUE),
    Avg_Novelty = mean(novelty, na.rm = TRUE)
  )

kable(summary_tbl, digits = 4, caption = "Average Diversity and Novelty Across Users")
```

Table 2: Average Diversity and Novelty Across Users

Avg_Diversity	Avg_Novelty
NaN	0.2644

```r
spark_disconnect(sc)
```

```r
# Sample top 10 recommendations with metadata
sample_recs <- top_10_recs %>%
  left_join(movies %>% select(movieId, title, genres), by = "movieId") %>%
  left_join(movie_popularity, by = "movieId") %>%
  filter(userId == 1) %>%  # You can change this to any user
  arrange(desc(predicted_rating))

kable(
  sample_recs %>% select(title, predicted_rating, genres, rating_count),
  caption = "Top 10 ALS Recommendations for User 1",
  digits = 3
)
```

Table 3: Top 10 ALS Recommendations for User 1

title	predicted_rating	genres	rating_count
Reign Over Me (2007)	5.68	Drama	5

Summary

In our content-based filtering (tags + consine similarity) we are able to measure the similarity between movies based on the genres and tags. Based on the results it shows more than half of the movie pairs have zero similarity. It indicates on average the similarity between movies is relatively low since it is .2161 and 75% of the pairs have similarity less than .4082. Majority of the movie pairs have little or no content similarity between them based on the genre and tags. Only a small number of the movies have some similarity.

Training the ALS model on the MovieLens 100k dataset was relatively fast due to its small size, taking under 1 minute using local Spark. However, as dataset size increases (e.g., MovieLens 1M or 20M), runtime and memory consumption will grow significantly. Spark's distributed computing capabilities make it scalable, especially when deployed on cloud infrastructure like Databricks or Azure HDInsight. The use of Spark was intentional to ensure that the recommender could scale beyond small educational datasets in future applications.

We ran the collaborative filtering with the Alternative Least Squares(ALS) which predicts ratings from user behaviors. The results of RMSE is .8852 and MAE is .6807 which indicated the ratings are about .68 to .88 rating points of the actual user ratings on a scale of 1 to 5. The ALS model is reasonably accurate as the predictions are low but there is room for improvements.

The novelty lets us know how new or surprising the recommended movies are to the user and with the results it shows the novelty scores is .26 which is a good. It is providing a good balance of movies that are new and old movies to the user.

The recommender was not able to calculate the diversity score for the users because it did not have enough detail information such as genres and tags to compare how the different recommended movies are.

Improvements

One of the improvements we can do is blend the predicting ratings from ALS with the content based similarity score to get a balanced recommendations. We could normalize the ratings to improve accuracy to avoid ratings that may be too high or low. Another improvement we can do is change the rank, reg_param, max_iter and test how the values change to see if there are improvements in the RMSE and MAE. RMSE and MAE are used to evaluate the prediction accuracy but it does not measure how satisfy the user might be on the recommended movies.

Potential Errors

We can have overfitting in our ALS model if the the rank is too high for the data and the regularization parameter is too low. In the movielens dataset there are many missing ratings and tags which causes it to get very few ratings and struggle to get meaningful data. There could potentially be more similarities in movies but with the lack of tags it's hard to determine and potentially introduce bias in tags. One of the challenges we face is a cold start problem when there is a new user or movies that don't have much data which makes it difficult to generate an accurate recommendation.