Data 612 - Project 5

Anna Moy & Natalie Kalukeerthie

2025-07-06

```r
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
# Connect to Spark locally
sc <- spark_connect(master = "local")

# Read CSV into Spark
ratings_tbl <- spark_read_csv(sc, name = "movies", path = "/Users/zhianna/Desktop/School/Data612/moviel

# Collect distinct movie IDs locally to create numeric IDs
movie_map_local <- ratings_tbl %>%
  distinct(movieId) %>%
  arrange(movieId) %>%
  collect() %>%
  mutate(movie_numeric_id = row_number())

# Copy movie_map back to Spark
movie_map_tbl <- copy_to(sc, movie_map_local, "movie_map", overwrite = TRUE)

# Join ratings with movie numeric IDs inside Spark
```

```r
ratings_numeric_tbl <- ratings_tbl %>%
  inner_join(movie_map_tbl, by = "movieId") %>%
  select(userId, movie_numeric_id, rating)

# Split train/test (80/20)
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 = 5,
    max_iter = 10,
    reg_param = 0.1,
    cold_start_strategy = "drop"
  )

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

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

rmse <- ml_evaluate(evaluator, predictions)
print(paste("RMSE on test data:", rmse))
```

```
## [1] "RMSE on test data: 0.920818581387218"
```

```r
# Collect predictions to R and join movie titles locally for display
preds_local <- predictions %>%
  collect() %>%
  left_join(movie_map_local, by = "movie_numeric_id") %>%
  select(userId, movieId, rating, prediction)

print(preds_local)
```

```
## # A tibble: 19,731 x 4
##    userId movieId               rating prediction
##     <int> <chr>                  <int>      <dbl>
##  1    460 A Chef in Love (1996)      3       3.78
##  2    148 Abyss, The (1989)          4       3.25
##  3    327 Abyss, The (1989)          3       3.10
##  4    343 Abyss, The (1989)          3       3.73
##  5    833 Abyss, The (1989)          2       2.79
##  6    117 Abyss, The (1989)          5       3.90
##  7    151 Abyss, The (1989)          5       3.62
##  8    246 Abyss, The (1989)          3       3.13
##  9    367 Abyss, The (1989)          4       4.18
## 10    377 Abyss, The (1989)          4       3.91
## # i 19,721 more rows
```

```
# Evaluate MSE
mse_evaluator <- ml_regression_evaluator(sc,
                                         label_col = "rating",
                                         prediction_col = "prediction",
                                         metric_name = "mse")

mse <- ml_evaluate(mse_evaluator, predictions)
print(paste("MSE on test data:", mse))
```

```
## [1] "MSE on test data: 0.847906859827969"
```

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

# ---- Results Table ----
library(tibble)
results_tbl <- tibble(
  Metric = c("RMSE", "MSE", "MAE"),
  Value = c(rmse, mse, mae)
)

print(results_tbl)
```

```
## # A tibble: 3 x 2
##   Metric Value
##   <chr>  <dbl>
## 1 RMSE   0.921
## 2 MSE    0.848
## 3 MAE    0.730
```

```
# Disconnect Spark
spark_disconnect(sc)
```

Summary

In Project 2, User-Based Collaborative Filtering (UBCF) and Item-Based Collaborative Filtering (IBCF) were applied to the MovieLens dataset. The RMSE values ranged from 1.15 to 1.52, which is significantly higher than the RMSE of 0.9208 achieved by the Alternating Least Squares (ALS) model implemented in Spark. The ALS model also produced lower MAE and MSE values compared to the UBCF and IBCF models. Overall, Spark provided better prediction accuracy, scalability, and performance—even when running on a local machine—though it introduced some additional complexity.

ALS is well-suited for large, sparse datasets with explicit ratings. It effectively handles missing data through regularization. In this implementation, the cold_start_strategy was used to prevent the model from generating predictions for users or items that were not seen during training, thus avoiding misleading or null predictions.

Using sparklyr in R enables distributed processing and memory optimization. The model executed efficiently on the local machine. However, there were some complexities, such as the need to save data locally before use, and the requirement to convert movie titles into numeric IDs to meet the input expectations of the ALS algorithm

When is Spark necessary?

Spark becomes necessary when working with large datasets that exceed the memory capacity of a single machine. It allows you to distribute the training process, increase computational speed, and support concurrent users. Spark is particularly beneficial for scalable machine learning systems, where training and inference need to operate at scale.

In this project, Spark was not strictly necessary, as the dataset fit comfortably on a single machine. However, implementing the system in Spark provides scalability and flexibility, enabling us to work with much larger datasets in the future with minimal changes to the code. The added complexity of setting up Spark, even in local mode, was small compared to the long-term benefits of accessing distributed computing resources.

Limitations and Improvements

Limitations is ALS can not make predictions for new users or items during training this would be a cold start problem. We may have to come up with ways to address the cold start problem that would best address the issue. Using ALS we have to get the user and items into a integer otherwise we would not be able to run the recommender which causes some complexity. ALS doesn't handle biases like the other matrix factorization which may impact the prediction quality and it uses latent factors which causes it to be hard to interpret why the recommendation was made.

Improvements we can make to the ALS model is playing around with the rank, reg_param and max_iter to see how each of these will impact our predictions and find out what changes will make it better. One of the things we want to consider is to normalize the rating before we train the data sine ALS does not account for biases. We can potentially include re-rank in the model to introduce diversity.