Implementing a Recommender System on Spark

Data 612 Project 5

William Outcault

01 July 2020

Contents

Introduction

Assignment

The goal of this project is give you practice beginning to work with a distributed recommender system. It is sufficient for this assignment to build out your application on a single node.

Adapt one of your recommendation systems to work with Apache Spark and compare the performance with your previous iteration. Consider the efficiency of the system and the added complexity of using Spark. You may complete the assignment using PySpark (Python), SparkR (R), sparklyr (R), or Scala.

Please include in your conclusion: For your given recommender system's data, algorithm(s), and (envisioned) implementation, at what point would you see moving to a distributed platform such as Spark becoming necessary?

You may work on any platform of your choosing, including Databricks Community Edition or in local mode. You are encouraged but not required to work in a small group on this project.

My Recommendation Model

I want to setup an experiment which will track the computation times as datasets become much larger. To setup this experiment I worked with the `MovieLense` dataset, `recommenderlab`, and `sparklyr`. This project was modified from my previous project which involved experimenting different schema methods, parameters, and algorithms to best optimize precision. This time around we want to focus on scalability.

In Context

We have two clients, both require instantaneous recommendations and live updates to their recommender engines, however one client has sparse data while the other does not. The goal is to be able to determine when to use `Spark` to make recommendations versus `recommenderlab`. We need to minimize computation time for both parties, so the question is how many ratings can we handle before switching to `Spark`?

Experiment

To do this I tried my best to setup identical models using `Spark` and `recommenderlab`. The time spent creating the recommender and making predictions are recorded over each iteration. In addition, each iteration increases our training set so we continuously work with more and more data. Times and sizes of our matrices are recorded after each test so we can eventually visualize the results.

Data Ingestion

As mentioned previously the data was received from `recommenderlab`'s `MovieLense` dataset. The tricky part of this experiment was getting `recommenderlab` and `Spark`'s data objects to scale with one another. `Spark` works with dataframes with ratings as a column and `recommenderlab` works with realRatingMatrices, this means for each iteration the dataset was manipulated in order to close enough sizes such that it would not skew the results. The code below initializes the dataset as a dataframe but is later manipulated throughout different sections of the code. The data ingestion was aided by Yun Mai, her code is referenced at the bottom.

```
data("MovieLense")
movie_df <- as(MovieLense, 'data.frame')
movie_df$user <- sapply(movie_df$user,function(x) as.numeric(as.character(x)))
movie_df$item  <- sapply(movie_df$item,function(x) as.numeric(as.factor(x)))
movie_mx <- spread(movie_df, item, rating)
movie_mx$user <- sapply(movie_mx$user,function(x) as.numeric(x))
movie_mx[is.na(movie_mx)]<- 0
```

ALS Functions

The model I chose was alternating least squares, I did this because both `recommenderlab` and `Spark` can create these models with relative similarities. Inside each function is a timer, which is recorded and returned after each iteration. My code was supplimented by the works of Ilya Kats. This will be referenced at the bottom.

recommenderlab

For the evaluation scheme I used a standard 80/20 split so it could be repeated using simple code for the Spark model. Parameters were picked to maintain consistency and timers were places precisely to eliminate irrelevant functions from skewing the experiment. The output includes relevant metrics for this study such as; number of ratings, computation time and RMSE.

```
als_model <- function(df){
  # Start computation timer
  df <- as.matrix(df)
  movie_ratings <- as(df[,-1],"realRatingMatrix")
  # Split schema
  schema <- evaluationScheme(data = movie_ratings, method = "split", train=0.80,
                             given = 5, goodRating = 3)
  # Recommender model
  start_time <- Sys.time()
  als_recs <- Recommender(getData(schema, "train"), method = "ALS")
  # Predictions
  als_preds <- predict(als_recs, getData(schema, "known"), type = "ratings")
  # Return times and number of ratings
```

```
  end_time <- Sys.time()
  time <- round(as.numeric(end_time - start_time), 2)
  dim <- dim(movie_ratings)
  dim <- dim[1]*dim[2]
  als_acc <- round(calcPredictionAccuracy(als_preds, getData(schema, "unknown")), 4)
  mylist <- list(time, dim, als_acc[[1]])
  return(mylist)
}
```

Spark

The splits were picked to mimick the `recommenderlab`'s function. Timers were once again placed in order to maintain precision for this experiment and the function ourput is identical to that of the previous function except with the Spark ALS recommender model performance metrics.

```
spark_als_model <- function(spark_df){
  which_train <- sample(x = c(TRUE, FALSE), size = nrow(spark_df),
                        replace = TRUE, prob = c(0.8, 0.2))
  train_df <- spark_df[which_train, ]
  test_df <- spark_df[!which_train, ]
  spark_train <- sdf_copy_to(sc, train_df, "train_ratings", overwrite = TRUE)
  spark_test <- sdf_copy_to(sc, test_df, "test_ratings", overwrite = TRUE)
  # Start computation timer
  start_time <- Sys.time()
  # Recommender Model
  als_recs <- ml_als(spark_train, max_iter = 5, nonnegative = TRUE,
                     rating_col = "rating", user_col = "user", item_col = "item")
  # Predictions
  als_preds <- als_recs$.jobj %>%
    invoke("transform", spark_dataframe(spark_test)) %>%
    collect()
  end_time <- Sys.time()
  # Remove NA
  als_preds <- als_preds[!is.na(als_preds$prediction), ]
  # SPARK Accuracy
  rmseSpark <- round(RMSE(als_preds$rating,als_preds$prediction), 4)
  # Return times and number of ratings
  time <- round(as.numeric(end_time - start_time), 2)
  mylist <- list(time, length(spark_df$rating), rmseSpark)
  return(mylist)
}
```

Calculating Model Performance

This section of code iterates over the functions discussed previously, slowly increasing the size of the datasets, and recording the relevant metrics. The metrics stored in this section include; number of ratings, computation time and RMSE.

```
set.seed(55)
n <- 0
eval_times <- data.frame()
```

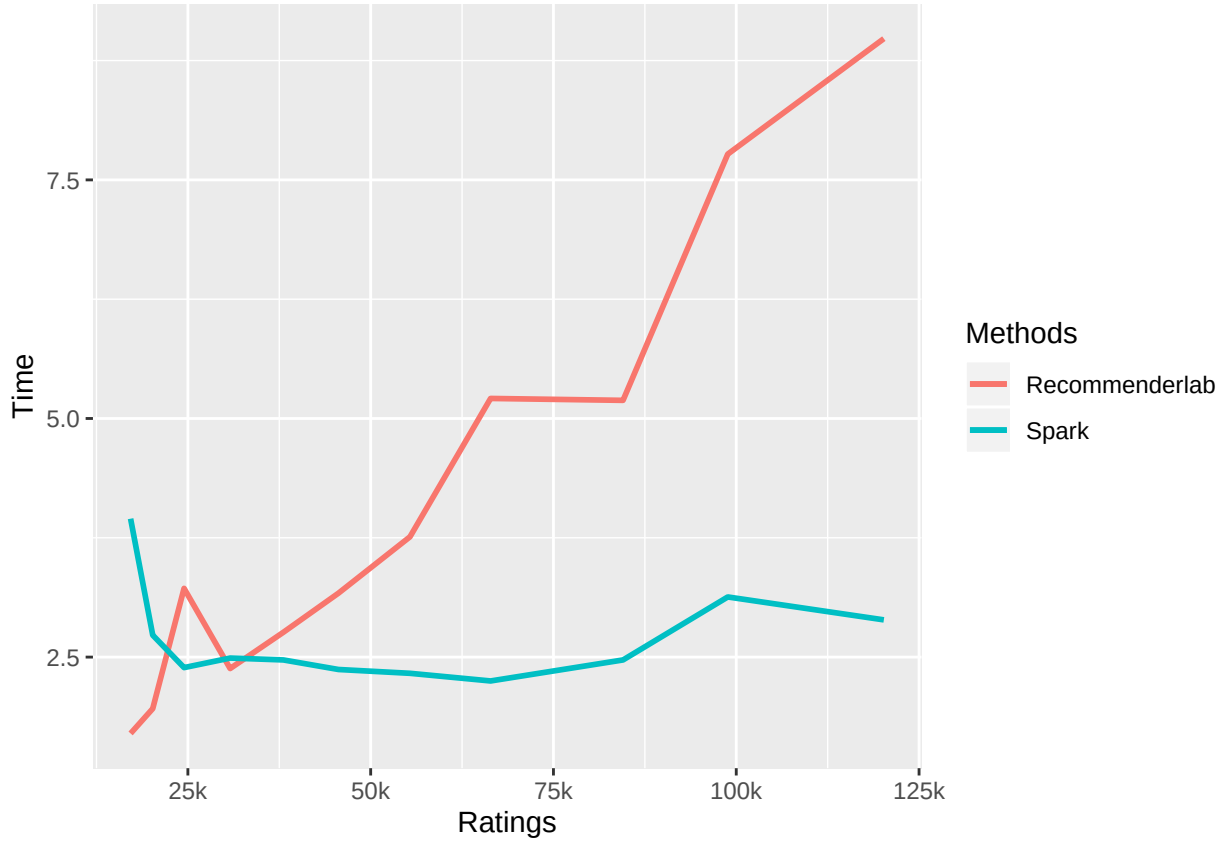
```
spark_eval_times <- data.frame()
sc <- spark_connect(master = "local")
for (i in seq(200,100,-10)){
  n <- n + 1
  # Slowly increase total scores
  df <- movie_mx[rowSums(movie_mx!=0) > i, colSums(movie_mx!=0) > i]
  # Prepare Spark df
  df_colnames <- colnames(df)
  spark_df <- gather(df,item,rating,df_colnames[2:length(df_colnames)])
  spark_df$item <- as.numeric(spark_df$item)
  # Run Recommenderlab ALS and store times
  eval_times <- rbind(eval_times, c(als_model(df)[[1]],als_model(df)[[2]],
                                    als_model(df)[[3]]))
  # Print Recommenderlab results
  print(paste("Time: ",eval_times[n,1],", Ratings: ",eval_times[n,2],
              ", RMSE: ",eval_times[n,3],", - Recommenderlab", sep=""))
  # Run SPARK ALS and store times
  spark_eval_times <- rbind(spark_eval_times, c(spark_als_model(spark_df)[[1]],
                                                spark_als_model(spark_df)[[2]],
                                                spark_als_model(spark_df)[[3]]))
  # Print SPARK results
  print(paste("Time: ",spark_eval_times[n,1],", Ratings: ",spark_eval_times[n,2],
              ", RMSE: ",spark_eval_times[n,3],", - Spark", sep=""))
}
spark_disconnect(sc)

eval_times <- setNames(eval_times, c("Time", "Ratings", "RMSE"))
spark_eval_times <- setNames(spark_eval_times, c("Time", "Ratings", "RMSE"))
```

Results

Spark surpasses recommenderlab after working with about 38k ratings. This does not take into account any effects that the dimensions of the matrices might have on computation time. As seen below

```
ggplot()+
    geom_line(data=eval_times,aes(x=Ratings, y=Time,colour="darkblue"),size=1 )+
    geom_line(data=spark_eval_times,aes(x=Ratings, y=Time,colour="red"),size=1) +
    scale_color_discrete(name = "Methods", labels = c("Recommenderlab", "Spark")) +
    scale_x_continuous(breaks = seq(0, 150000, 25000) , labels = c("0","25k","50k","75k","100k","125k",
```



We can see the intersection between computation times indicates the moment at which you would switch to Spark. The `recommenderlab` computation time has a positive linear correlation with the number of ratings meanwhile `Spark` has the capability of working with datasets much larger than these before time is significantly effected.

Table 1: recommenderlab

Time	Ratings	RMSE
1.70	17168	2.0351
1.96	20193	2.0407
3.22	24480	2.1218
2.38	30780	2.1585
2.76	38064	2.0943
3.17	45600	2.1056
3.76	55372	2.0794
5.21	66402	2.0480
5.19	84512	2.0000
7.77	98838	2.0078
8.98	120184	1.9191

Table 2: Spark

Time	Ratings	RMSE
3.95	17168	1.6466
2.73	20193	1.6680
2.39	24480	1.6572
2.49	30780	1.6702
2.47	38064	1.6657
2.37	45600	1.6723
2.33	55372	1.6466
2.25	66402	1.6340
2.47	84512	1.6143
3.13	98838	1.5843
2.89	120184	1.5689

```
knitr::kable(eval_times, caption = "recommenderlab")
```

```
knitr::kable(spark_eval_times, caption = "Spark")
```

After each iteration we see that the `Spark` starts off slower but does not hinder as the datasets grow larger. Alternatively `recommenderlab` begins quick and increases linearly as the datasets increase. In addition notice the consistently lower RMSE scores for `Spark`. Both RMSE scores increase as datasets increase which is to be expected, however `recommenderlab`'s RMSE increases slightly quicker.

Reference Code

Ilya Kats - https://rstudio-pubs-static.s3.amazonaws.com/403474_cc6283ecad504d749a5a8fa00ce90f7a.html
Yun Mai - https://rpubs.com/YunMai/DATA643_Project5_Spark