DATA 612 - Project 5

Brian Liles

July 9, 2019

Objective:

The goal of this project is to 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?

Step One: Set Up Libraries & Spark Locally

Following the instructions based on https://github.com/rstudio/sparklyr (https://github.com/rstudio/sparklyr) it was decided to conduct a local spark installation.

Using **spark_connect** we will connect to the local instance of Spark

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

Step Two: Choose dataset

From the **recommenderlab** package we will be using the **MovieLense** dataset

```
data("MovieLense")
MovieLense
```

```
## 943 x 1664 rating matrix of class 'realRatingMatrix' with 99392 ratings.
```

MovieLense rating matrix has 964 rows and 1664 columns

Step Three: Create Training Sets & Models for Comparison

As per my submission in Project #3, when creating a recommender system, a potential customer would feel more comfortable with information from reliable sources. In the text, users who have rated at least 50 movies and watched 100 were used.

```
ratings_movies <- MovieLense[rowCounts(MovieLense) > 50,colCounts(MovieLense) > 100]
ratings_movies
```

```
## 560 x 332 rating matrix of class 'realRatingMatrix' with 55298 ratings.
```

ratings_movies rating matrix now has 560 rows and 332 columns

```
test <- evaluationScheme(ratings_movies, method = "split", train = 0.8, k = 4, given = 15, goodRating = 3)

# method: this is the way to split the data
# train: this is the percentage of data in the training set
# given: number of items to keep
# goodRating: rating threshold
# k: number of times to run the evaluation
```

Based off results in a prior test, the **IBCF** method proved best so we will utilize that technique.

```
ibcfRecMod <- Recommender(getData(test,"train"), "IBCF")
```

Next, we will make predictions

```
ibcfPred <- predict(ibcfRecMod, getData(test, "known"), type = "ratings")
cat("IBCF Method: RMSE, MSE, MAE","\n","\n")
```

```
## IBCF Method: RMSE, MSE, MAE
##
```

```
(ibcf <- calcPredictionAccuracy(ibcfPred, getData(test, "unknown")))
```

```
##     RMSE      MSE      MAE
## 1.412915 1.996329 1.065154
```

```
ibcfResults <- evaluate(test, method  = "IBCF", n = seq(10,100,10))
```

```
## IBCF run fold/sample [model time/prediction time]
##   1  [0.48sec/0.03sec]
##   2  [0.32sec/0.06sec]
##   3  [0.3sec/0.03sec]
##   4  [0.3sec/0.03sec]
```

```
head(getConfusionMatrix(ibcfResults)[[1]])
```

```
##           TP        FP       FN       TN precision     recall        TPR
## 10  2.794643  7.205357 67.68750 239.3125 0.2794643 0.04016984 0.04016984
## 20  5.258929 14.741071 65.22321 231.7768 0.2629464 0.07390874 0.07390874
## 30  7.589286 22.410714 62.89286 224.1071 0.2529762 0.10668712 0.10668712
## 40  9.732143 30.267857 60.75000 216.2500 0.2433036 0.13734679 0.13734679
## 50 11.973214 38.026786 58.50893 208.4911 0.2394643 0.16979585 0.16979585
## 60 14.160714 45.839286 56.32143 200.6786 0.2360119 0.20113629 0.20113629
##           FPR
## 10 0.02888043
## 20 0.05918836
## 30 0.09024749
## 40 0.12214269
## 50 0.15364431
## 60 0.18548352
```

Step Four: Convert MovieLense to Dataframe

```
# convert the MovieLense data into a data frame entitled MovieLenseDF
MovieLenseDF <- as(MovieLense, 'data.frame')
glimpse(MovieLenseDF) # use the glimpse function from the tidyverse to see the data
```

```
## Observations: 99,392
## Variables: 3
## $ user   <fct> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ item   <fct> Toy Story (1995), GoldenEye (1995), Four Rooms (1995), ...
## $ rating <dbl> 5, 3, 4, 3, 3, 5, 4, 1, 5, 3, 2, 5, 5, 5, 5, 5, 3, 4, 5...
```

Next, we will convert the **factor** data to **numeric** variable

```
# convert factor variables into numeric variables
MovieLenseDF$user <- as.numeric(MovieLenseDF$user)
MovieLenseDF$item <- as.numeric(MovieLenseDF$item)
glimpse(MovieLenseDF)
```

```
## Observations: 99,392
## Variables: 3
## $ user   <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ item   <dbl> 1525, 618, 555, 594, 344, 1318, 1545, 111, 391, 1240, 1...
## $ rating <dbl> 5, 3, 4, 3, 3, 5, 4, 1, 5, 3, 2, 5, 5, 5, 5, 5, 3, 4, 5...
```

Step Five: Spark

First, we will copy the **MovieLenseDF** data into spark using the **sdf_copy_to** command

```
start_time <- proc.time()

# sdf_copy_to(sc, x, name, memory, repartition, overwrite, ...)
(MovieLenseSprk <- sdf_copy_to(sc,MovieLenseDF,"spmovie", overwrite = TRUE))
```

```
## # Source: spark<spmovie> [?? x 3]
##     user  item rating
##   * <dbl> <dbl>  <dbl>
##  1     1  1525      5
##  2     1   618      3
##  3     1   555      4
##  4     1   594      3
##  5     1   344      3
##  6     1  1318      5
##  7     1  1545      4
##  8     1   111      1
##  9     1   391      5
## 10     1  1240      3
## # ... with more rows
```

According to the article **Prototyping a Recommender System Step by Step Part 2: Alternating Least Square (ALS) Matrix Factorization in Collaborative Filtering** the author informs readers on how the algorithm was constructed for Apache Spark and does a decent job at solving scalability and sparseness of ratings data.

```
MovieLenseALS <- ml_als(MovieLenseSprk)
summary(MovieLenseALS)
```

```
##                         Length Class      Mode
## uid                     1      -none-     character
## param_map               4      -none-     list
## rank                    1      -none-     numeric
## recommend_for_all_items 1      -none-     function
## recommend_for_all_users 1      -none-     function
## item_factors            2      tbl_spark  list
## user_factors            2      tbl_spark  list
## user_col                1      -none-     character
## item_col                1      -none-     character
## prediction_col          1      -none-     character
## .jobj                   2      spark_jobj environment
```

Next, we will calculate predictions based on the use of **ml_als**

```
# item_factor predictions
MovieLenseALS$item_factors
```

```
## # Source: spark<?> [?? x 12]
##       id features features_1 features_2 features_3 features_4 features_5
##   * <int> <list>        <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
##  1    10 <list [~    -0.153       1.15      -0.840     0.118      -1.08
##  2    20 <list [~     0.0631      0.515     -0.830     0.669      -0.488
##  3    30 <list [~     0.109       0.445     -0.626     0.126      -0.601
##  4    40 <list [~    -0.264       0.573     -0.173    -0.770      -0.497
##  5    50 <list [~    -0.214       0.692     -0.482    -0.0664     -0.591
##  6    60 <list [~     0.0623      0.829     -0.642    -0.0772     -0.800
##  7    70 <list [~    -0.179       0.972     -0.624     0.260      -0.589
##  8    80 <list [~     0.138       0.744     -0.550    -0.214      -0.743
##  9    90 <list [~    -0.534       0.802     -0.633    -0.332      -0.965
## 10   100 <list [~    -0.839       0.805     -0.502    -0.385      -0.892
## # ... with more rows, and 5 more variables: features_6 <dbl>,
## #   features_7 <dbl>, features_8 <dbl>, features_9 <dbl>,
## #   features_10 <dbl>
```

```
# user_factor predictions
MovieLenseALS$user_factors
```

```
## # Source: spark<?> [?? x 12]
##       id features features_1 features_2 features_3 features_4 features_5
##  * <int> <list>        <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
##  1    10 <list [~     0.179       0.995     -0.955    -0.574      -0.870
##  2    20 <list [~    -0.0772      1.08      -0.605    -0.241      -0.764
##  3    30 <list [~    -0.926       0.737     -0.919     0.0637     -1.42
##  4    40 <list [~    -0.230       0.691     -0.791    -0.804      -1.16
##  5    50 <list [~     0.193       0.987     -0.610    -0.845      -0.245
##  6    60 <list [~    -1.32        1.04      -0.999    -0.392      -0.978
##  7    70 <list [~    -0.697       0.666     -0.248    -0.798      -0.390
##  8    80 <list [~    -0.662       1.15       0.341    -0.712      -1.01
##  9    90 <list [~    -0.558       0.983     -0.956    -0.354      -1.10
## 10   100 <list [~    -0.537       0.956     -1.35     -0.154      -1.03
## # ... with more rows, and 5 more variables: features_6 <dbl>,
## #   features_7 <dbl>, features_8 <dbl>, features_9 <dbl>,
## #   features_10 <dbl>
```

```
(sparkPredict <- ml_predict(MovieLenseALS,spark_dataframe(MovieLenseSprk)))
```

```
## # Source: spark<?> [?? x 4]
##     user  item rating prediction
##  * <dbl> <dbl>  <dbl>      <dbl>
##  1   857    12      4       3.30
##  2   868    12      4       4.00
##  3   822    12      1       1.67
##  4   759    12      4       3.51
##  5   141    13      4       3.63
##  6   367    13      2       2.63
##  7   173    13      4       3.76
##  8   503    13      5       4.59
##  9    17    14      5       4.57
## 10   231    14      5       4.43
## # ... with more rows
```

Step Six: Closing Spark

```
(end_time <- proc.time() - start_time)
```

```
##    user  system elapsed
##    2.65    0.08   23.71
```

```
spark_disconnect(sc)
```

```
## NULL
```

The learning curve for **sparklyr** wasn’t as bad as expected, however an extended time would have been beneficial. It is easy to see how the use of the this platform is celebrated in the creation of recommendation systems.Based on the procedure time, it is faster than the **IBCF** method.