Final Project

Max Wagner

July 12, 2016

Introduction

The MovieLens data was put into a smaller, more compact version that includes only user and movies with suffient information by the University of Minnesota. One of the problems with the data set is that time is not taken into consideration when looking at user's recommendations. The set provides a timestamp for each recommendation, abit in a strange format. The timestamp is in number of seconds from midnight on January 1, 1970. Using this information, it will be possible to weight ratings that are closer to present day higher than older ratings.

The process will be completely done on Apache Spark, which includes the use of `SparkR` and likely a minimal SQL database. Ideally, this will allow for better management of data and queries than pure CSV. Additionally, multiple methods of recommendation will be used to find the most accurate prediction strategy.

Due to ease of use, a local version of Spark was initiated. The Amazon Ec2 version worked fine, but due to being offline often during the process of writing this, the local was more efficient.

Libraries

```
library(SparkR)
library(reshape2)
library(dplyr)
library(recommenderlab)
library(stringr)
library(knitr)
```

Starting a Spark Environment

```
Sys.setenv(SPARK_HOME = "C:/spark")
.libPaths(c(file.path(Sys.getenv("SPARK_HOME"), "R", "lib"), .libPaths()))
sc <- sparkR.init(master = "local")
```

```
## Launching java with spark-submit command C:/spark/bin/spark-submit.cmd   sparkr-shell C:\Users\Max\Ap
```

```
sqlContext <- sparkRSQL.init(sc)
```

Data

The first step is to load in the data from the small data set, and then separate the dates into a separate data frame, and convert the seconds to years. The year of the rating, and the year of the movie will be added to the ratings table. We can then preview the movies and ratings tables to make sure they appear normal.

```
#load in data from github
movies <- read.csv("ml-latest-small/movies.csv", stringsAsFactors = FALSE)
ratings <- read.csv("ml-latest-small/ratings.csv", stringsAsFactors = FALSE)

#see what it looks like in spark
df <- createDataFrame(sqlContext, ratings)
printSchema(df)
```

```
## root
##  |-- userId: integer (nullable = true)
##  |-- movieId: integer (nullable = true)
##  |-- rating: double (nullable = true)
##  |-- timestamp: integer (nullable = true)
```

```
showDF(df, numRows = 5)
```

```
## +------+-------+------+----------+
## |userId|movieId|rating| timestamp|
## +------+-------+------+----------+
## |     1|     16|   4.0|1217897793|
## |     1|     24|   1.5|1217895807|
## |     1|     32|   4.0|1217896246|
## |     1|     47|   4.0|1217896556|
## |     1|     50|   4.0|1217896523|
## +------+-------+------+----------+
## only showing top 5 rows
```

```
#query example if i needed to do it this way
registerTempTable(df, "df")
dates <- SparkR::sql(sqlContext, "SELECT timestamp FROM df")
df <- SparkR::sql(sqlContext, "SELECT userId,movieId,rating FROM df")

#get the rating dates locally
dates <- as.vector(ratings[,4])
ratings <- ratings[,-4]
ratings$ryear <- as.integer(dates/60/60/24/365 + 1970)

#get the movie dates
ratings <- inner_join(ratings, movies[,1:2],by = "movieId")
colnames(ratings)[5] <- "myear"
ratings$myear <- as.integer(str_sub(ratings$myear, -5, -2))

kable(head(movies))
```

movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
2	Jumanji (1995)	Adventure|Children|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama|Romance
5	Father of the Bride Part II (1995)	Comedy
6	Heat (1995)	Action|Crime|Thriller

```
kable(head(ratings))
```

userId	movieId	rating	ryear	myear
1	16	4.0	2008	1995
1	24	1.5	2008	1995
1	32	4.0	2008	1995
1	47	4.0	2008	1995
1	50	4.0	2008	1995
1	110	4.0	2008	1995

The above all seems reasonable for the information we have so far. One interesting approach to take when weighting based on time is to scale the rating itself, instead of scaling the recommender values. I'll take this approach for the purpose of this project. To decide how to weight the rating, we should include both the year of the rating, and the year of the movie. The most straightforward method is to scale based on where the most recent ratings and movies recieve the least penalty to their score. For instance, a rating made in 2016 will be multipled by 1, where a rating made in 1997 might be multiplied by 0.2. Scaling in this method keeps all values above 0, and below 5.

```
# first replace the few rows out the 100000 that had issues with the movie year
ratings$myear[is.na(ratings$myear)] <- mean(ratings$myear, na.rm = TRUE)

# scale it
ratings$scale <- (ratings$myear^25 / max(ratings$myear)^25) * (ratings$myear^25 / max(ratings$myear)^25)
ratings$ratingscaled <- ratings$rating * ratings$scale
summary(ratings$scale)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.05582 0.50940 0.60730 0.59730 0.70570 1.00000
```

```
summary(ratings$rating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   3.500   3.517   4.000   5.000
```

```
summary(ratings$ratingscaled)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.03923 1.45300 2.04900 2.08400 2.68500 5.00000
```

```
# convert to a spark df with a sample
ratings_sample <- sample_n(ratings,2000)
df <- createDataFrame(sqlContext, ratings_sample)
```

The summary of the scale shows that on average a rating will be scaled by 0.67, while the lowest multiplier is 0.22, and the highest is 1.0. The overall ratings dropped dramatically for both mean and median, but the min and max remain within 0 and 5, which is desired.

Next, the ratings table is put into a real ratings matrix format using `recommenderlab`. In partial thanks to Sreejaya Nair and Suman K Polavarapu for the cleaner way to get to a ratings matrix with `acast`. I'll do this twice. Once for the original ratings, and once for the scaled ratings.

```
ratings_hori <- acast(ratings, userId ~ movieId, value.var="rating")
ratings_rm <- as(ratings_hori, "realRatingMatrix")
ratings_rm_r <- ratings_rm[, colCounts(ratings_rm) > 20]
```

UBCF - Original Ratings

First, the original ratings will be placed into a UBCF recommender to get a baseline without any context weighting.

```
movies.recom <- Recommender(ratings_rm_r[1:400], method = "UBCF")
movies.predict <- predict(movies.recom, ratings_rm_r[c(500, 525, 600)],n=3)
movies.predict <- as.data.frame(as(movies.predict, "list"))
kable(movies.predict)
```

X500	X525	X600
318	296	541
356	318	1641
593	356	1213

In this case, for user 500 the top movies suggested are: `Shawshank Redemption, The (1994)`, `Forrest Gump (1994)`, and `Silence of the Lambs, The (1991)`. We can see that each movie recommended is older in this selection.

UBCF - Scaled Ratings

Now to try the same recommender with the weighted ratings.

```
ratings_hori <- acast(ratings, userId ~ movieId, value.var="ratingscaled")
ratings_rm <- as(ratings_hori, "realRatingMatrix")
ratings_rm_r <- ratings_rm[, colCounts(ratings_rm) > 20]
movies.recom <- Recommender(ratings_rm_r[1:400], method = "UBCF")
movies.predict <- predict(movies.recom, ratings_rm_r[c(500, 525, 600)],n=3)
movies.predict <- as.data.frame(as(movies.predict, "list"))
kable(movies.predict)
```

X500	X525	X600
7153	296	50
5952	527	3481
4993	608	4973

With the weighted ratings, the 1st recommendation changes to `Lord of the Rings: The Return of the King, The (2003)`, the 2nd to `Lord of the Rings: The Two Towers, The (2002)`, and the third to `Lord of the Rings: The Fellowship of the Ring, The (2001)`. It is very interesting to see that all three Lord of the Rings movies were suggested. The pattern seems to suggest that the series is HIGHLY recommended for this person to see.

IBCF - Original Ratings

```
ratings_hori <- acast(ratings, userId ~ movieId, value.var="rating")
ratings_rm <- as(ratings_hori, "realRatingMatrix")
ratings_rm_r <- ratings_rm[, colCounts(ratings_rm) > 20]
movies.recom <- Recommender(ratings_rm_r[1:400], method = "IBCF")
movies.predict <- predict(movies.recom, ratings_rm_r[c(500, 525, 600)],n=3)
movies.predict <- as.data.frame(as(movies.predict, "list"))
kable(movies.predict)
```

X500	X525	X600
5	36	172
7	45	372
16	141	405

We can see here that the recommendations are different from the UBCF methods. 1st is `Father of the Bride Part II (1995)`, 2nd is `Sabrina (1995)`, and 3rd is `Casino (1995)`.

IBCF - Scaled Ratings

```
ratings_hori <- acast(ratings, userId ~ movieId, value.var="ratingscaled")
ratings_rm <- as(ratings_hori, "realRatingMatrix")
ratings_rm_r <- ratings_rm[, colCounts(ratings_rm) > 20]
movies.recom <- Recommender(ratings_rm_r[1:400], method = "IBCF")
movies.predict <- predict(movies.recom, ratings_rm_r[c(500, 525, 600)],n=3)
movies.predict <- as.data.frame(as(movies.predict, "list"))
kable(movies.predict)
```

X500	X525	X600
4022	293	61240
4993	342	1884
4995	1610	4299

In this case, with a scaled IBCF method, we see that 1st is `Cast Away (2000)`, 2nd is `Lord of the Rings: The Fellowship of the Ring, The (2001)`, and 3rd is `Beautiful Mind, A (2001)`.

UBCF and IBCF Conclusions

From the two different methods we can see a few common trends. The scaled version, as expected, generally gives recommendations that are more modern, and have been rated more recently. In the case of the scaled UBCF, it recommended a user all 3 Lord of the Rings movies. This seems beyond a coincidence and a complete effect of the scaling.

The scaled versions included some of the same movies with both methods, meaning that the accuracy of the recommender was improved by introducing weighting to the ratings.

Further Work

The above is just one version of what could be done. It is also possible to weight by something like genre. I had stated in a earlier project that each user tends to prefer certain genres, and some genres appear more than others. For instance, in this set the following frequencies can be found. It may be viable to scale based on the rarity of the genre as well as the year of the rating and movie.

```
movies.o <- movies[,3]
genres.split <- unlist(strsplit(movies.o, split = "|", fixed = TRUE))
genres.count <- data.frame(base::table(genres.split))
kable(head(genres.count[order(-genres.count$Freq),],10))
```

	genres.split	Freq
9	Drama	5220
6	Comedy	3515
18	Thriller	2187
16	Romance	1788
2	Action	1737
7	Crime	1440
3	Adventure	1164
12	Horror	1001
17	Sci-Fi	860
15	Mystery	675

The point of the above is that there are any number of scaling values, and scaling by just the year isn't a realistic method for getting the best results. Many different aspects should be considered, not just temporal ones.

Citations

https://github.com/srajeev1/MSDA-IS643/blob/master/projects/Project2/DATA643_Project_2.Rmd

https://spark.apache.org/docs/latest/sparkr.html