Big Data Sample Project

2 Movie Datasets

Table of Contents

Executive summary ***3***

Data Description ***3***

MapReduce (MRJob) ***4***

Hive (SQL Queries) ***7***

Zeppelin (Scala, Spark SQL, Visualization) ***8***

ML in Spark (ALS: MLLib) ***11***

Executive summary

The objective of this Big Data project is to take a chunk of big data of size 10MB+ from any source and demonstrate the usage of Big Data tools and technologies (as many as possible) on it. In our project, two imdb datasets having an overall size of 512MB+ have been used (source: gouplens.org). Big Data tools, namely MapReduce, Hive, PIG, Zeppelin have been applied to perform different operations (i.e. ETL, queries, visualization) on Hadoop, TEZ, and Spark, cluster-computing frameworks using Python and Scala languages. An ML recommender model has also been developed using ALS from Spark MLLib to recommend new movies to an existing member based on his/her and other members' watching habits. Its accuracy has also been compared for a larger and a smaller dataset and as per expectation a higher accuracy has been achieved for the larger dataset due to better training of the model.

Data Description

The bigger dataset has two data files, namely **"rat.csv"** (biggest file: 509MB) and **"mov.csv"** (1.4MB).

The columns for the rat.csv file are as below:

userID- MovieID- Rating- Epoch

The columns for the mov.csv file are as below:

MovieID- Title- Genres

This dataset consists of 20000263(i.e. 2M) ratings and 27278 movies. These data were created by 138493 users between January 09, 1995 and March 31, 2015. This dataset was generated on March 31, 2015, and updated on October 17, 2016. The rating value ranges from 0.5-5. The users have been selected randomly and the movies with at least one rating have been selected.

The second dataset i.e. the smaller one has also two files: **u.data**(2MB) and **u.item**(231kB)

The columns for the u.data file are as below:

userID- MovieID- Rating- Epoch

The columns for the mov.csv file are as below:

MovieID- Title- Release Date- Genres

This dataset consists of 100003(i.e. 0.1M) ratings and 1682 movies. These data were created by 943 users during the seven-month period from September 19th,1997 through April 22nd, 1998. The rating value ranges from 1-5 and each user has rated at least 20 movies.

For both the datasets:

1. users and items are numbered consecutively from 1.

2.Genres have the following categories: Action/Adventure/Animation/ Children's/ Comedy/ Crime/ Documentary/ Drama/ Fantasy/ Film-Noir/ Horror/ Musical/ Mystery/ Romance/ Sci-Fi/ Thriller/ War/ Western/ (no genres listed)

The Headers are removed in Linux before uploading the larger dataset to the Hadoop cluster.

MapReduce (MRJob)

In this MapReduce job, four files have been created. For bigger dataset: Rb11.py and Rb33.py and for smaller dataset: Rb22.py and Rb44.py< Rb11 and Rb22 are similar and so are Rb22 and Rb44 >

Mrjob, Mrstep, and Numpy Python libraries have been used and the programs have been run on Hadoop cluster.

The **Rb11.py,** is created to find the rating breakdown i.e. count of each star rating: a mapper and reducer function have been written. In Mapper, the csv data file is splitted into a tuple of four fields for each line. The "rating" field is appended with 1, for each rating and the (key1, value1) pair is formed. It's then shuffled and sorted based on the key1 values and subsequently passed to the reducer for aggregation operation (i.e. SUM here). The SUM value is then printed for each rating. The result is shown below (note that the result is not sorted):

```
[root@sandbox-hdp ~]# python Rb11.py rat.csv
No configs found; falling back on auto-configuration
Creating temp directory /tmp/Rb11.root.20190410.203530.169539
Running step 1 of 1...
Streaming final output from /tmp/Rb11.root.20190410.203530.169539/output...
"4.0"   5561926
"4.5"   1534824
"5.0"   2898660
"0.5"   239125
"1.0"   680732
"1.5"   279252
"2.0"   1430997
"2.5"   883398
"3.0"   4291193
"3.5"   2200156
```

The **Rb33.py** is created to find the top-rated-movies by movieID and based on the count of ratings against each movieID: a mapper and reducer function have been written at first like Rb11.py. In Mapper, the csv data file is splitted into a tuple of four fields for each line. The "movieID" field is appended with 1, for each movieID and the (key1, value1) pair is formed. It's then shuffled and sorted based on the key1 values and subsequently passed to the reducer for aggregation operation (i.e. SUM here). Next another reducer is written and the (Key2, value2) is passed to it from the first reducer as (count, movies) so that the count value gets sorted while being passed. Finally, the movies with more than 50,000 rating counts are printed in ascending order- the highest rated movie with movieID, 296 has a rating count of 67310. The result is shown below (note that the result is sorted):

```
[root@sandbox-hdp ~]# nano Rb33.py

[root@sandbox-hdp ~]# python Rb33.py rat.csv

No configs found; falling back on auto-configuration
```

Creating temp directory /tmp/Rb33.root.20190410.221519.242759

Running step 1 of 2...

Running step 2 of 2...

Streaming final output from /tmp/Rb33.root.20190410.221519.242759/output...

"527"	50054

"2571"	51334

"589"	52244

"110"	53769

"260"	54502

"480"	59715

"593"	63299

"318"	63366

"356"	66172

"296"	67310

Removing temp directory /tmp/Rb33.root.20190410.221519.242759...

Similar, things have been tried in Zeppelin later with Spark SQL to compare the elapsed time and also to do visualization.

PIG Latin (ETL)

The PIG operation has been run on TEZ. The larger dataset has been used.

Two files have been created, namely **4plusStarRating.pig** and **Less_than_1star_rating**.

In both the programs PIG Latin has been used to do ETL operations.

In 4plus*.pig, the data has been extracted from Hadoop cluster using PigStorage().

After that different transformation operations (in sequence): Group By, Aggregation, Generate, Filter, Join, Generate, Order By and Limit, have been applied to find the movies having 4+ star ratings. In results, the latest and oldest top 5 movies, sorted based on epoch, are shown. It is to be noted that in this dataset, release date is not provided- the earliest epoch has been considered as the release date.

At last the transformed data has been loaded to Hadoop cluster using PigStorage() again.

The result as follows (The first five oldest; last five latest):

```
ratings: {userID: int,movieID: int,rating: float,epoch: int}
intermediateData: {movieId: int,title: chararray,gerne: chararray,avgRating: double,releaseDate: int}
(47,Seven (a.k.a. Se7en) (1995),Mystery|Thriller,4.053492566302111,789652009)
```

(50,"Usual Suspects, The (1995)",4.334372207803259,822873600)
(28,Persuasion (1995),Drama|Romance,4.057545973367152,823255320)
(110,Braveheart (1995),Action|Drama|War,4.042533802004873,824559456)
(111,Taxi Driver (1976),Crime|Drama|Thriller,4.110575548384461,824559459)

(131176,A Second Chance (2014),Drama,4.5,1427692203)
(131082,Playground (2009),(no genres listed),4.5,1427654742)
(131050,Stargate SG-1 Children of the Gods - Final Cut (2009),Adventure|Sci-Fi|Thriller,5.0,1427650612)
(121190,When in Rome (2002),Children|Comedy,4.5,1427642407)
(131027,But Forever in My Mind (1999),Comedy|Drama,4.5,1427612224)

In Less_*.pig, similar operations have been done to get highest rated movies having less than 1-star rating. Filter has been applied to negate movies with only one rating to boost the credibility. Only movies with at least two rating counts are considered.

The result as follows (The highest rated, less than 1-star movie got rated 426 times):

ratings: {userID: int,movieID: int,rating: float,epoch: int}

(6483,From Justin to Kelly (2003),Musical|Romance,0.9730046948356808,426)

(8859,SuperBabies: Baby Geniuses 2 (2004),Comedy,0.8373205741626795,209)

(7282,"Hip Hop Witch, Da (2000)",0.7241379310344828,29)

(107704,Justin Bieber's Believe (2013),Documentary,0.5909090909090909,11)

(32925,Who's Your Daddy? (2004),Comedy,0.9166666666666666,6)

(56835,Pledge This! (2006),Comedy,0.75,6)

(83660,Trog (1970),Horror|Sci-Fi,0.9166666666666666,6)

(71112,Mentiras y gordas (2009),Comedy|Drama|Romance,0.7,5)

(65994,"Dead Calling, A (2006)",0.875,4)

(106688,Crocodile (2000),Horror|Thriller,0.875,4)

(96923,2-Headed Shark Attack (2012),Comedy|Horror,0.6666666666666666,3)

(104677,King Cobra (1999),Horror|Sci-Fi,0.8333333333333334,3)

(107115,"Fairly Odd Movie: Grow Up, Timmy Turner!,0.5,3)

(108146,Rise of the Zombies (2012),Action|Horror|Thriller,0.8333333333333334,3)

(115357,"Chair, The (2007)",0.6666666666666666,3)

(120222,Foodfight! (2012),Action|Animation|Children|Comedy,0.5,3)

(127630,The Legend of Awesomest Maximus (2011),Action|Comedy,0.6666666666666666,3)

(5805,Besotted (2001),Drama,0.5,2)

(80154,Urban Menace (1999),Action|Horror,0.5,2)

(81125,"American Dream, An (1966)",0.75,2)

(91960,"Magic Christmas Tree, The (1964)",0.5,2)

(93484,Princess of Mars (2009),Sci-Fi|Thriller|War,0.75,2)

(95232,Beauty #2 (1965),Drama,0.5,2)

(97283,Creature from the Haunted Sea (1961),Comedy|Horror,0.75,2)

(98030,Atomic Twister (2002),Action|Drama|Sci-Fi,0.75,2)

(98080,"Tomb, The (2007)",0.5,2)

(101728,Miss Castaway and the Island Girls (2004),Adventure|Comedy|Fantasy,0.5,2)

(102176,"Late Great Planet Earth, The (1979)",0.5,2)

(103286,Shark Alarm at Müggelsee (Hai Alarm am Müggelsee) (2013),Comedy,0.75,2)

(104101,Bloody Murder (2000),Drama|Horror|Thriller,0.75,2)

(104746,Killjoy (2000),Horror,0.75,2)

(107079,"Fairly Odd Christmas, A (2012)",0.5,2)

(112776,Bloodline (2011),Horror,0.5,2)

(117065,The Damned (2014),Horror|Mystery|Thriller,0.75,2)

(130398,Transmorphers (2007),Action|Adventure|Sci-Fi,0.5,2)

Hive (SQL Queries)

Initially, the smaller dataset was uploaded to Hive 2.0 for testing but later due to technical issues the larger dataset could not be uploaded on Hive.

In Hive, queries to get top-rated-movies and movies with highest average rating (movies with at least ten ratings are only considered) have been tried.

The queries in Hive 2.0 environment are completely lost. Some backup queries tried in Zeppelin are shown below:

Two tables, ratings (from u.data) and title (from u.item) are created to perform queries on.

Most Popular Movies

REA

```
%jdbc(hive)
create view if not exists topMovieIDs as
select movieID,count(movieID) as ratingCount
from ratings
group by movieID
order by ratingCount desc;

select n.title, ratingCount
from topMovieIds t JOIN title n ON t.movieID = n.movieID
```

Query executed successfully. Affected rows : -1

n.title	ratingcount
Star Wars (1977)	583
Contact (1997)	509
Fargo (1996)	508
Return of the Jedi (1983)	507

Movies with Highest Average Rating

```
%jdbc(hive)

create view if not exists avgRatings as
select movieID,AVG(rating) as avgRating, COUNT(movieID) as ratingCount
from ratings
group by movieID
order by avgRating desc;

select n.title, avgRating
from avgRatings t JOIN title n ON t.movieID = n.movieID
WHERE ratingCount > 10
```

Zeppelin (Scala, Spark SQL, Visualization)

In Zeppelin, Scala and Spark SQL have been used to get similar results as MapReduce. The data is fetched from Hadoop cluster and converted to RDDs. Next, an immutable Scala class object is defined, and the map transformation is performed on RDDs to split each data rows to fields and then those are assigned to the Scala class object. The RDDs are then converted to a DataFrame. Furthermore, different operations are done on the DF to get the top movieIDs. Lastly, the DF is registered as a temporary table to perform JOIN operation with another table in Spark SQL. The time required for getting the final result got reduced significantly against MapReduce. Additionally, visualization have been shown for the results. The file named, **Zeppelin+Scala2.json** is created for the larger dataset. The results are shown below:

Highest rate movies i.e. top-rated-movies:

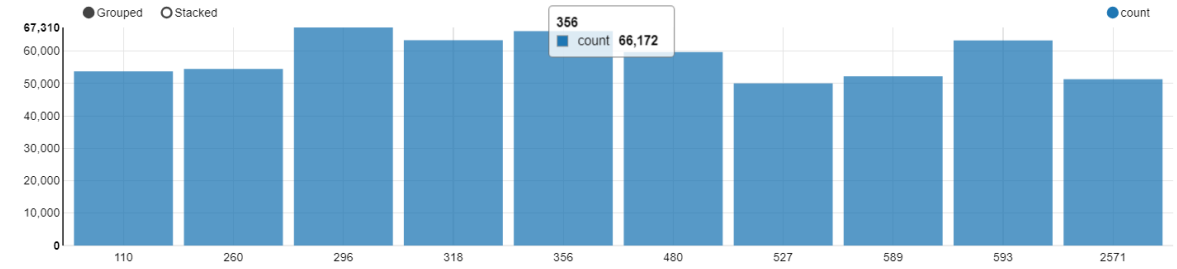
movieID	count
296	67310
356	66172
318	63366
593	63299
480	59715
260	54502
110	53769
589	52244
2571	51334

Took 1 sec. Last updated by anonymous at April 10 2019, 5:33:53 PM. (outdated)

Query the ratings DataFrame via SparkSQL

```
%sql
SELECT * FROM topMov LIMIT 10
```

settings

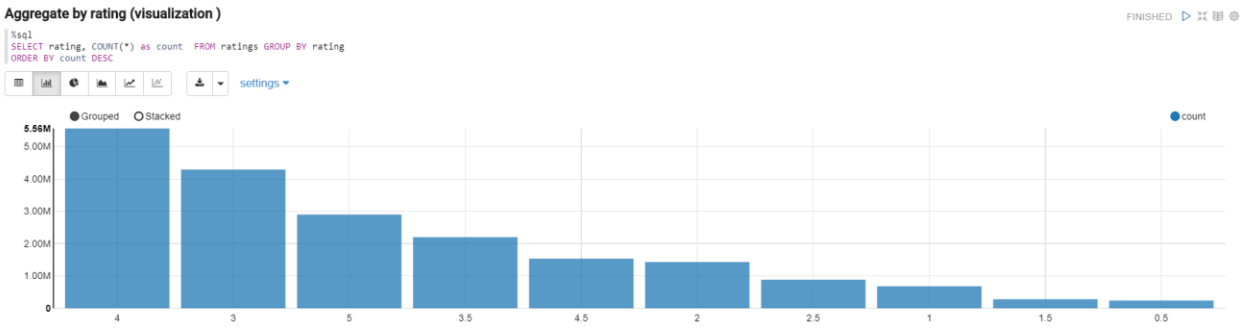


Rating Breakdown:

Aggregate by rating (visualization)

```
%sql
SELECT rating, COUNT(*) as count  FROM ratings GROUP BY rating
ORDER BY count DESC
```

rating	count
4.0	5561926
3.0	4291193
5.0	2898660
3.5	2200156
4.5	1534824
2.0	1430997
2.5	883398
1.0	680732



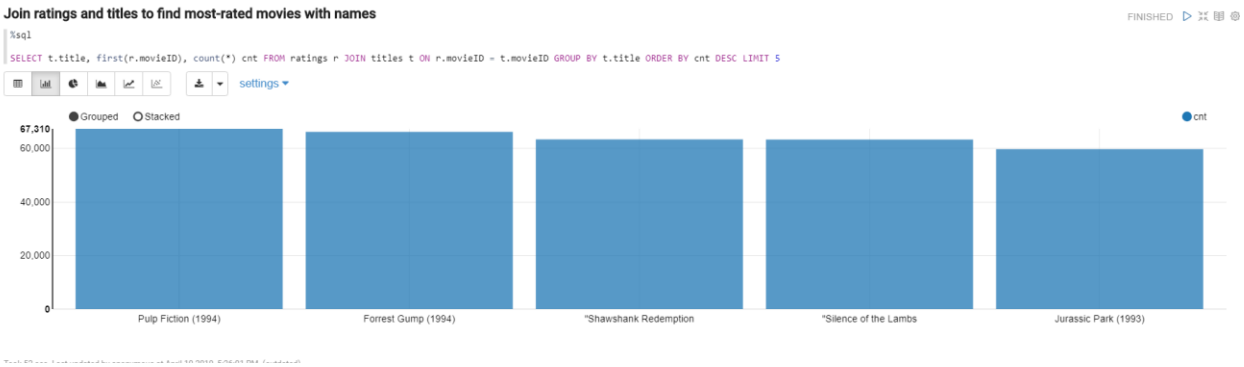
Finally, the two datasets have been joined to find movie names for top-rated-movies (top 5 shown)[Not available in MapReduce job]

Join ratings and titles to find most-rated movies with names

FINISHED

```
%sql
SELECT t.title, first(r.movieID), count(*) cnt FROM ratings r JOIN titles t ON r.movieID = t.movieID GROUP BY t.title ORDER BY cnt DESC LIMIT 5
```

title	first(movieID, false)	cnt
Pulp Fiction (1994)	296	67310
Forrest Gump (1994)	356	66172
"Shawshank Redemption	318	63366
"Silence of the Lambs	593	63299
Jurassic Park (1993)	480	59715



ML in Spark (ALS: MLLib)

Firstly, a comparison of the model accuracy has been done based on rmse(root-mean-square-error) on the larger(rat.csv) and the smaller(u.data) datasets using ALS (Alternating Least Square) algorithm in MLLib library of Spark. The program is written in Python language using pyspark.

The data is splitted into 80:20 ratio as train: test, for both the cases.

Refer, **ML5.json** for smaller dataset and **ML7.json** for larger dataset.

For smaller data, a rmse of 1.07381993168 has been obtained, whereas for larger one, a rmse of 0.815900439836 has been obtained i.e. [**rmse(smaller)] ≈ 1.32 [rmse(larger)]**

From the result it quite evident that a larger dataset produces a robust model for prediction.

[Number of data rows in larger file(rat.csv)] ≈ 200 [Number of data rows in smaller file(u.data)]

Secondly, in ML7.json file (i.e. for larger dataset), a recommender algorithm has been developed at the second part of the script to recommend movies (from the whole database of movies) for any existing user that he/she has not seen yet, **based on his/her and other members watching habits.**

User with userID=1 is taken as an example to justify the model.

The top-10 list shown below from the train data, furnishes: movieID, Title, Genres, and Rating (provided by userID#1). The table is sorted in descending order based on the rating provided by userID#1.

>>>>>>Movies already loved and highly rated by User#1<<<<<<<
(4993, '"Lord of the Rings: The Fellowship of the Ring', ' The (2001)"', 5.0)
(5952, '"Lord of the Rings: The Two Towers', ' The (2002)"', 5.0)
(1196, 'Star Wars: Episode V - The Empire Strikes Back (1980)', 'Action|Adventure|Sci-Fi\r\n', 4.5)
(1198, 'Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)', 'Action|Adventure\r\n', 4.5)
(8636, 'Spider-Man 2 (2004)', 'Action|Adventure|Sci-Fi|IMAX\r\n', 4.5)
(151, 'Rob Roy (1995)', 'Action|Drama|Romance|War\r\n', 4.0)
(293, 'L\xc3\xa9on: The Professional (a.k.a. The Professional) (L\xc3\xa9on) (1994)',
'Action|Crime|Drama|Thriller\r\n', 4.0)
(296, 'Pulp Fiction (1994)', 'Comedy|Crime|Drama|Thriller\r\n', 4.0)
(318, '"Shawshank Redemption', ' The (1994)"', 4.0)
(1036, 'Die Hard (1988)', 'Action|Crime|Thriller\r\n', 4.0)

Now, a test dataset is prepared by filtering the test data by userId#1 and then removing ratings for userID#1 and taking only userID and movieID. The modified test data is then fed to the recommender model to get prediction for userId#1. The result is then sorted in descending order by the predicted ratings for userId#1 to get top-10 prediction for userID#1.

The top-10 recommendations can now be compared to the top-10 train list: It is quite evident that the recommender system has done an excellent job as the predicted movies perfectly matches the favourite genres (and sequel names wherever applicable) of User#1. **userID#1's most loved genres are: sci-fi, action, adventure, thriller.**

>>>>Recomendation for User#1<<<<<
(260, 'Star Wars: Episode IV - A New Hope (1977)', 'Action|Adventure|Sci-Fi\r\n', 4.177889823913574)
(7153, '"Lord of the Rings: The Return of the King', ' The (2003)"', 3.983344316482544)
(1240, '"Terminator', ' The (1984)"', 3.9788243770599365)
(6242, 'Ringu (Ring) (1998)', 'Horror|Mystery|Thriller\r\n', 3.9252049922943115)
(1214, 'Alien (1979)', 'Horror|Sci-Fi\r\n', 3.9051353931427)
(541, 'Blade Runner (1982)', 'Action|Sci-Fi|Thriller\r\n', 3.898895740509033)
(2716, 'Ghostbusters (a.k.a. Ghost Busters) (1984)', 'Action|Comedy|Sci-Fi\r\n', 3.8858866691589355)
(50, '"Usual Suspects', ' The (1995)"', 3.8557701110839844)
(2761, '"Iron Giant', ' The (1999)"', 3.7673757076263428)
(4226, 'Memento (2000)', 'Mystery|Thriller\r\n', 3.7665700912475586)

Lastly, the predicted ratings for the top-10 recommended movies are also compared with actual rating provided by userID#1. The comparison again bolsters the robustness of the recommender model.

>>>>Recommender Predicted Rating and Actual Rating for User#1 side by side <<<<<
(260, 'Star Wars: Episode IV - A New Hope (1977)', 'Action|Adventure|Sci-Fi\r\n', 4.177889823913574, 4.0)
(7153, '"Lord of the Rings: The Return of the King', ' The (2003)"', 3.983344316482544, 5.0)
(1240, '"Terminator', ' The (1984)"', 3.9788243770599365, 4.0)
(6242, 'Ringu (Ring) (1998)', 'Horror|Mystery|Thriller\r\n', 3.9252049922943115, 3.5)
(1214, 'Alien (1979)', 'Horror|Sci-Fi\r\n', 3.9051353931427, 4.0)
(541, 'Blade Runner (1982)', 'Action|Sci-Fi|Thriller\r\n', 3.898895740509033, 4.0)
(2716, 'Ghostbusters (a.k.a. Ghost Busters) (1984)', 'Action|Comedy|Sci-Fi\r\n', 3.8858866691589355, 3.5)
(50, '"Usual Suspects', ' The (1995)"', 3.8557701110839844, 3.5)
(2761, '"Iron Giant', ' The (1999)"', 3.7673757076263428, 3.0)
(4226, 'Memento (2000)', 'Mystery|Thriller\r\n', 3.7665700912475586, 3.5)

Note: For program scripts of all the sections, please refer the attached Resources folder.