Assignment 1: Movie Lens Analysis using Pig and Hive

Anthony Reidy, 18369643

October 19, 2021

1 Introduction

Movie Lens is a web-based movie recommendation service developed by researchers at the University of Minnesota. The service contains about 11 million ratings for about 8500 movies. They provide a zip folder, Ml-latest-small , to foster research on personalised recommendations. The datasets, generated on September 26, 2018, detail 100836 ratings and 3683 tag applications across 9742 movies. Here, is a brief overview of the CSV files and their associated features.

- **Ratings.csv**: userId, movieId, rating, timestamp
- **Tags.csv**: userId, movieId, tags, timestamp
- **Movies.csv**: movieId, title, generes
- **Links.csv**: movieId, imdbId, tmdbId

In this investigation, we seek to use Pig and Hive to clean and analyse the described datasets. This report is organised into the following over-arching sections:

1. **Prerequisites:** This section details the Hadoop Bash commands required to transport the dataset files from local to HDFS storage

2. **Pig:** The required Pig commands and outputs are displayed here. Only documentation of the code and minor observations of these outputs are reported.

3. **Hive:** Hive queries and their corresponding outputs are detailed in this section. Again, documentation of the queries and only minor observations are discussed.

4. **In-depth Analysis** Finally, we use the Python programming language to visualise both the Hive and Pig queries. We also provide more-in depth analysis compared to the previous sections.

2 Prerequisites

As Pig and Hive are key components of the Hadoop ecosystem, the aforementioned CSV files need to be copied to a location on HDFS. The bash commands are contained in the prerequisites.sh file. *Start-all.sh* is a bash script that starts all Hadoop daemons, the namenode, datanodes, the jobtracker and tasktrackers. The *-mkdir* flag is utilised to create an input directory, input/movielens/, where all CSV files will be stored. Next, the *-put* flag transfers a CSV file from the local filesystem to HDFS. Figure 1 verifies the correctness of the above bash commands.

3 Pig

Pig is an open-source project, initially developed at Yahoo, that provides a data flow engine in Hadoop using a high-level scripting language. Pig offers substantial parallelisation, a very useful tool for analysing and cleaning large datasets.

Figure 1: Grunt output of $HADOOP_HOME/bin/hdfs dfs -ls input/movielens/

3.1 Data cleaning

Before any analysis, data cleaning should be performed to ensure accurate results. The primary cleaning operations performed are:

Split multiple genres: Currently, the genre(s) that describe a movie are represented by a pipe separated string e.g. Adventure|Animation|Children|Comedy|Fantasy. The $STRSPLIT()$ function is used to split the genres string into separate substrings where only one genre is contained in a substring.

Extract year from title: Generally, the title of a movie comes in the following format: $<Title\ (Year)>$. However, there are exceptions. Twelve movies do not have any year attached. We are still interested in such movies, so we keep them in. In addition, one movie has a year range rather than a single year , Death Note: Desu nôto (2006–2007). In this case, we replaced (2006–2007) with (2006), as according to Wikipedia, that is the date of release. The $REGEX\_EXTRACT()$ function with the regular expression parameter \\d((\\d \\d\\d\\d)\\), is utilised to extract the year. Finally, we use the $REPLACE()$ function to remove the year from the title string and $TRIM()$ to remove any trailing whitespace.

Merge ratings and movies datasets: All of the queries listed on the lab sheet require either the ratings dataset or a combination of the ratings and movies datasets. Therefore, we decide to inner join ratings and movies on their common feature, movieID. It should be noted that there were eighteen movies that did not have any user ratings attached. For the purposes of our analysis, we are not interested in such occurrences and drop them.

Change Delimiter: The merged ratings movies dataset is stored as a CSV file with tab characters as delimiters. This will enable hive loading of the dataset when the genres feature will be used to split a single row into multiple rows. Thus, allowing analysis of ratings by genre.

Remove unnecessary column: The timestamp column is not used in this investigation. Thus,it is dropped.

We define two different alias for the CSVExcelStorage UDF as we need to use different constructor parameters for different calls (loading & storage). The CSVExcelStorage removes a lot of problems such as skipping input header and removing unnecessary quotation marks in tags. The tmdbId of the links dataset contains missing values. We are not interested in this column. Therefore, no cleaning actions are performed. **Note** The data_cleaning.pig script contains the relevant Pig commands for this sub-section.

Figure 2: Grunt output of the bottom 5 rows of the last realtion in the data cleaning script

3.2 Pig Queries

As part of an assignment, we were tasked with performing a set of queries (Set A) using Pig:

1. What is the title of the movie with the highest number of ratings (top-rated movie)?

2. What is the title of the most liked movie (e.g. only 5 stars ratings (1) OR only ratings >4 (2) OR majority of 5 star ratings (3))?

3. What is the User with the highest average rating?

Pig-anaylsis.pig is the script relevant for this section. Query 1 and 3 could be accomplished using the primary relational and arithmetic operators.

Figure 3: Grunt output of query 1

```
(Forrest Gump,329)
grunt>
```

Forest Grump had the highest number of ratings (329).

Figure 4: Grunt output of query 2

```
(Streetcar Named Desire, A,20,5.0|4.0|4.0|4.5|4.5|4.0|4.5|5.0|5.0|5.0|5.0|5.0|5.0|5.0|5.0|5.0|3.0|4.0|3.0|4.0)
grunt>
```

The three filters for query 2 are displayed above. It must be noted that all movies present in the first filter are also present in the second filter. Therefore, we essentially have two filters (2,3). In addition, filter 2 and 3 are not mutually exclusive. We interpret 'majority of 5 starts ratings' to describe movies where at least 50% of their ratings are 5 star ratings. Unfortunately there aren't many convenient ways to work with bags in Pig out of the box [1]. We utilise our self-created UDF (python_udf.py) and the $BAG\_TO\_STRING()$ function to filter the most liked movies. $BAG\_TO\_STRING()$ converts the bag to a pipe separated list. Then, our UDF outputs a Boolean value indicating if the movie is popular. The described query requires a single movie so we order by the number of ratings a movie has received (descending) and choose the top occurrence. We conclude Streetcar Named Desire, A is the most liked movie.

Figure 5: Grunt output of query 3

```
(53,5.0)
grunt>
```

The user who submitted the highest average rating was user 53 (userId) who had an average rating of 5 stars. **Note**, no other users had an average rating of 5.

Figure 6: Grunt output of last 4 rows of prolific raters

```
(610,3.5,81591,Black Swan,2010,(Drama,Thriller),610,1302)
(610,3.5,109187,Zero Theorem, The,2013,(Drama,Fantasy,Sci-Fi),610,1302)
(610,3.5,4901,Spy Game,2001,(Action,Crime,Drama,Thriller),610,1302)
(610,2.0,5128,Queen of the Damned,2002,(Fantasy,Horror),610,1302)
grunt>
```
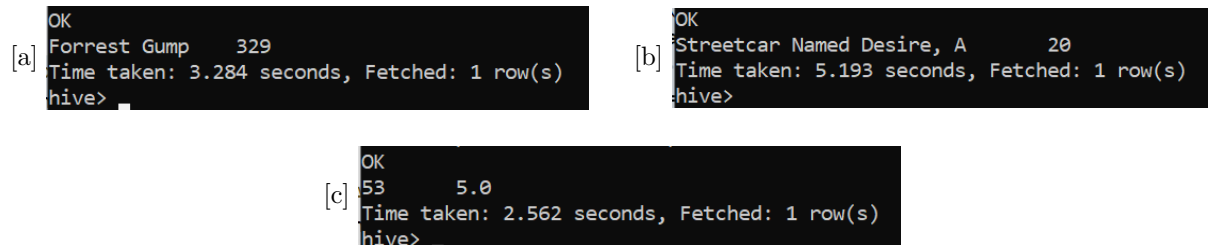
4 Hive

Hive is data warehouse infrastructure tool, initially developed at Facebook, that enables structured data processing in Hadoop via a SQL-like language. Unlike Pig, Hive is not used to clean unstructured data.

---

[1] http://datafu.apache.org/docs/datafu/guide/bag-operations.html

4.1 Hive Queries

We were tasked with performing the same set of queries above (Set A) and an additional set (Set B). The hive-anaylsis.sql file is the Hive script relevant for this section.

[a]
```
OK
Forrest Gump    329
Time taken: 3.284 seconds, Fetched: 1 row(s)
hive>
```

[b]
```
OK
Streetcar Named Desire, A      20
Time taken: 5.193 seconds, Fetched: 1 row(s)
hive>
```

[c]
```
OK
53      5.0
Time taken: 2.562 seconds, Fetched: 1 row(s)
hive>
```

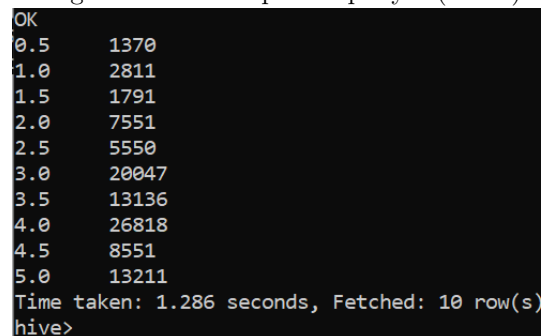Figure 7: Figures displaying Hive output of (a) Query 1 (b) Query 2 (c) Query 3 for Set A

Query 1 and 3 can be achieved by using the primary clauses and aggregate functions. However, query 2 is more complex in nature. A WITH statement is used to give a sub-query block a name (a process known as sub-query refactoring). The benefits of this are twofold; improved readability of the code when compared to nested sub-queries; quicker debugging of queries due to reduced complexity. The two temporary relations, above_4_star_ratings and majority_five_stars, detail all movies that **only** have a rating above 4 stars or all movies that have a **majority** of five stars. CASE statements are used within the HAVING clause to filter group results according to the conditions described above. Finally, we use a FULL OUTER JOIN to get all of the movies than can be described by these conditions as well as the $COALESCE()$ function to get the combined set of values from the duplicate columns of these two temporary relations.

Set B queries:

1. Count the number of ratings for each star level
2. What is the most popular rating?
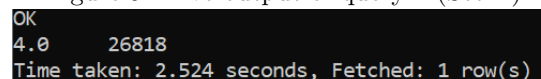3. How are ratings distributed by genre?

Query 1 and 2 can be achieved by using the primary clauses and aggregate functions.

Figure 8: Hive output of query 1 (Set B)

```
OK
0.5     1370
1.0     2811
1.5     1791
2.0     7551
2.5     5550
3.0     20047
3.5     13136
4.0     26818
4.5     8551
5.0     13211
Time taken: 1.286 seconds, Fetched: 10 row(s)
hive>
```

Figure 9: Hive output of query 2 (Set B)

```
OK
4.0     26818
Time taken: 2.524 seconds, Fetched: 1 row(s)
```

The four star rating is most prevalent.

Figure 10: Hive output of query 3 (Set B)

```
2010    Horror  460     3.2728260869565218      1.1023791722086693
2010    IMAX    1849    3.574905354245538       1.0248073882627187
2010    Musical 164     3.5121951219512195      1.07769970068394
2010    Mystery 679     3.696612665684831       1.0084343930482462
2010    Romance 881     3.3552780930760497      1.0905631514788348
2010    Sci-Fi  2497    3.5616740088105727      1.0313473196793792
2010    Thriller        2373    3.4943109987357777      1.0288041242315238
2010    War     265     3.581132075471698       0.9526639122617597
2010    Western 186     3.629032258064516       0.9848399952092313
2010    no genres listed        27      3.2777777777777777      1.227262335243029
Time taken: 2.224 seconds, Fetched: 209 row(s)
hive>
```

The above screenshot is a snippet of the the number of users, average rating, and the standard deviation of ratings grouped by decade and genre. In this instance, grouping by year would have generated too much x tick labels for a plot. The release year of movies in this dataset, ranges from 1902 to 2018. Again, the WITH statement is used to give sub-query blocks a name. In the first temporary relation, parenthesis_removed, we remove the parenthesis around the genres names e.g. (Comedy,Children,War) → Comedy,Children,War. Next (genres_exploaded), we use lateral view, the table generating function $explode()$, and the $split()$ function to convert the rows containing a comma separated list of genres into multiple rows, where every genre gets its own row. The last relation, ratings_and_users_grouped_by_genre_decade, uses the $FLOOR()$ function to cast a year to its corresponding decade, e.g. 1988 -¿ 1980. The in-built agregate functions ($COUNT()$, $AVG()$ and $STDDEV()$) aid us in caluclating number of users, average rating, and the standard deviation of ratings.

We use the INSERT OVERWRITE LOCAL DIRECTORY command to export a query or sub-query result into a file on local storage for visualisation purposes. The final query performs unigram processing which will enable us to compare the most liked and unpopular movie's tags (wordcloud). Hive has three in-built functions $NGRAMS()$, $SENTENCES()$ and $LOWER()$ that aid us here

5   In-depth analysis

First, we are interested in how ratings are distributed by genre.
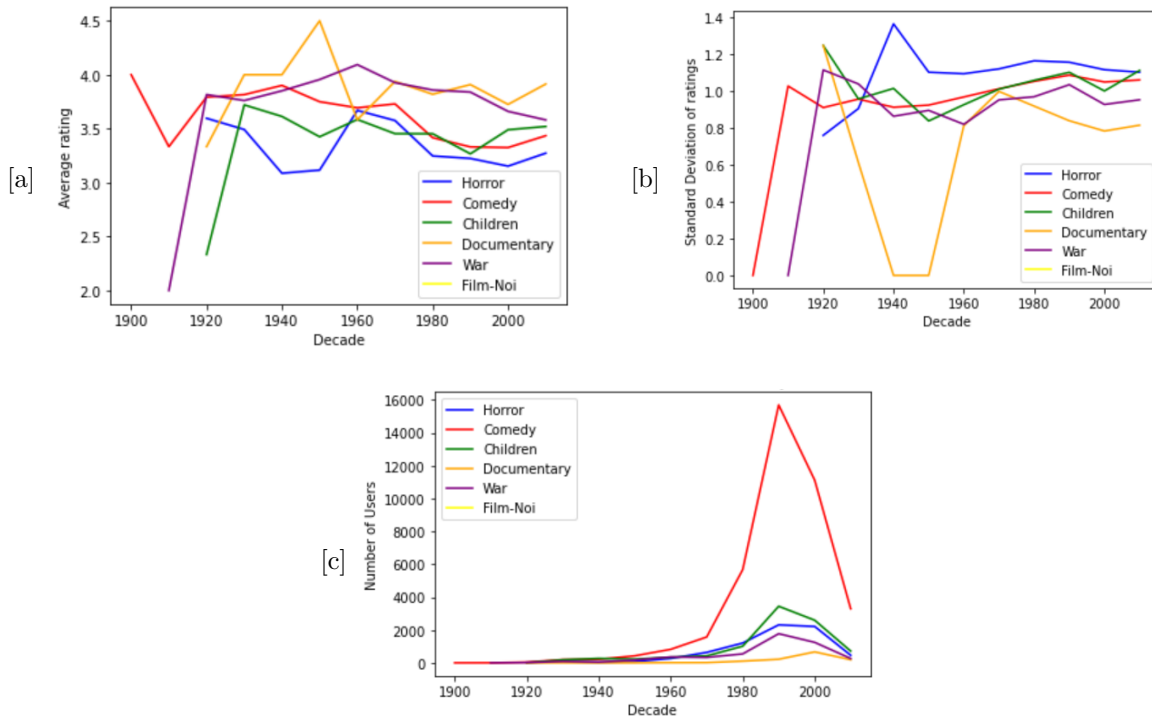
Figure 11: Boxplot of ratings by genre
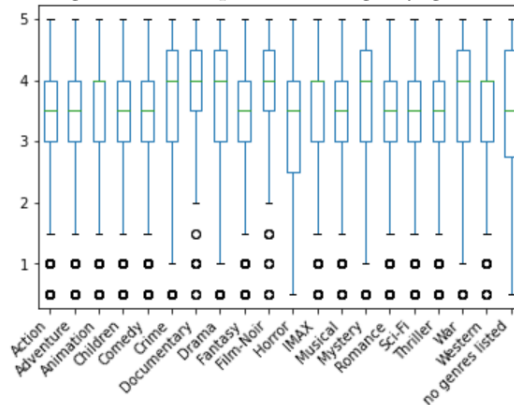
From the boxplots above action, adventure, comedy, crime, musical, scifi, thriller all have similar distributions. 50% of their data are located between 3 and 4 stars, indicating mostly positive reviews. We also observe the 3rd shortest length if we pool these genres together, with "maximum" and "minimum" (according to a normal distribution) being two and five stars. Most users agree movies within these genres tend to be good. In contrast, we observe crime and horror are comparatively tall. This suggests that users hold quite different opinions about the movies of these genres. As expected, films with no

5

genre also show this behaviour as they lack any binding quality! Documentaries, crime, film-noir and wars films seem to be the most well liked. Outliers in this plot represent users who do not agree with the general consensus on a particular genre. The ratings density plot displays the kernel density estimation (KDE) of the ratings for each genre as a single coloured line with a blue histogram of overall ratings in the background. We observe that the genre's are predominantly left-skewed, with the majority of them having a mean at about 3.5 stars. We hypothesis that this phenomenon occurs as users are more willing to log onto Movielens and rate movies that they enjoyed. People are more likely just to switch to a different movie if they dislike it.

Figure 12: Rating Density Plot
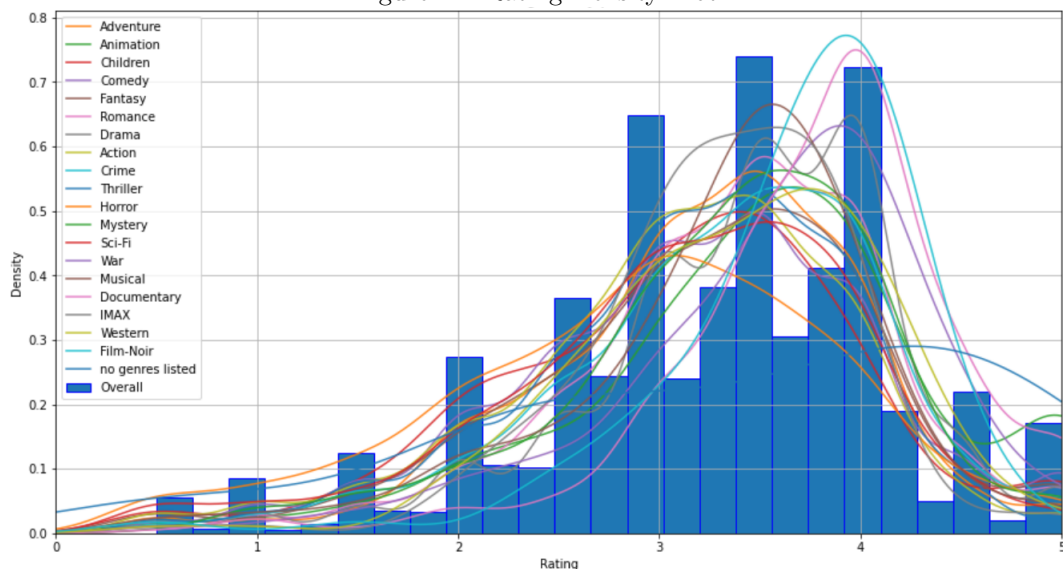
But how doe these scores vary over time? We plot the **top three most liked** genres (Documentary, War, Film-Noi) and **top three unpopular genres** (Horror, Comedy, Children) on figure 13. It is interesting to observe that one popular genre (War) and one unpopular genre (Children) are generally poorly perceived before the 1950s. This could be due to advancements in CGI and Pyrotechnics technologies that contribute to a more realistic viewing experience. We suspect horror movies are reviewed poorly because they tend to typically focused on cheap thrills but are not typically impactful. In general, the standard deviation is higher for unpopular genres indicating a mixed response rather than a negative one given the relative high mean. We observe that the number of reviews for all investigated genres are quite low pre the 1960s. In addition, the number of reviews for each genres tends to peak at around the 1990s to the 2000s, falling drastically afterwards. We hypothesis that the usage of Movielens peaked in the late 2000s, falling considerably there after. We also suspect a young to middle aged demographic uses Movielens given how all the lines in this plot (c) are left-skewed. The vast increase in comedy reviews is likely due to low specificity of the genre which promotes multi-genre classification. Thus, more movies belong to this genre. In contrast, the other genres are relatively niche.

Figure 13: Figures displaying results for the (a)Average rating of genres by decade (b) Standard deviation of ratings of genres by decade (c) Number of reviews who rated each genre by decade

Lastly, we plot the tags of the popular and unpopular movies as word clouds. Tags are typically single words or short phrase that capture the meaning, value, and purpose of a movie as determined by each user. Rather than grouping by the tag values and counting the number of occurrences, we decide to perform unigram processing. As suspected, positively sentimented words are attributed to the most liked movies. Although we observe some negatively sentimented words in the unpopular movies' tags, tags associated with comic book movies (e.g. comic, superhero) are also common place. This is likely due to the high expectations of comic movie fan bases and the mass spamming of negativity we observe when those expectations aren't met.

Figure 14: Figures displaying wordclouds of tags from (a) most-liked movies (b) unpopular movies