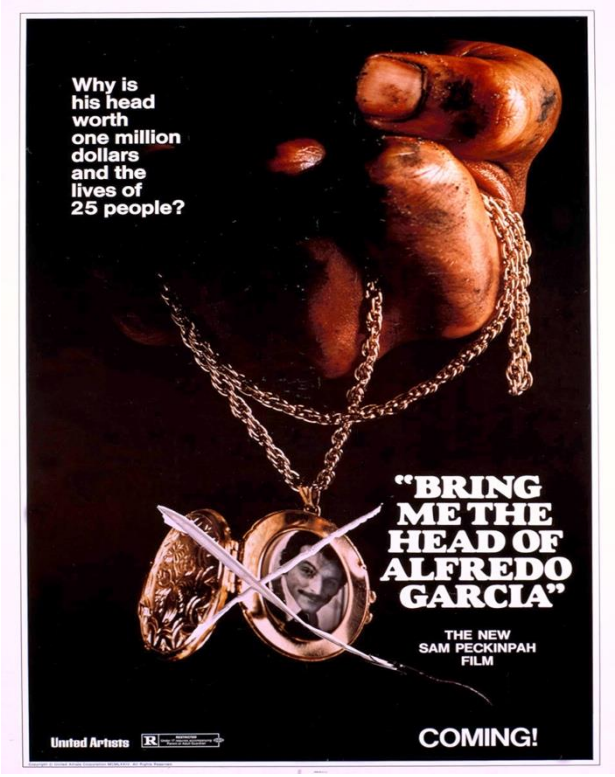
# SEIS 736 PAPER

## An Analysis of Movies Using GroupLens




**Aaron Leff**

# TABLE OF CONTENTS

**Introduction: Data Source and Data Description**

As an avid-movie fan and aspiring software engineer, I was happy to use films as the focus of my big data project. My research is based on data from the MovieLens web site (http://movielens.org) collected by a University of Minnesota researchers from GroupLens Research. Significantly, I used both Hadoop Distributed File System (HDFS) and Hue 2.2.0 Hive interface to complete this project. HDFS and Hue are both provided on our local virtual machine. My seemingly simple goal: can I determine which age groups are most and least likely to watch movies?

I selected the 1M Dataset (permalink: http://grouplens.org/datasets/movielens/1m/). The website elaborates on the dataset: the files contain 1,000,209 anonymous ratings made by 6,040 users of approximately 3,952 movies. This information was contained in three .dat files: ratings, users, and movies.

The ratings file uses the following format: `UserID::MovieID::Rating::Timestamp` As would be expected from the dataset description, UserId's range between 1 and 6040 while MovieId's range between 1 and 3952.

The users file uses the following format: `UserID::Gender::Age::Occupation::Zip-code`. The Age description is most relevant to this paper. According to the website:

```
- Age is chosen from the following ranges:

        *  1:  "Under 18"
        * 18:  "18-24"
        * 25:  "25-34"
        * 35:  "35-44"
        * 45:  "45-49"
        * 50:  "50-55"
        * 56:  "56+"
```

This is, of course, invaluable for correlating age groups with movie users later in the paper.

The movies file uses the following format: `MovieID::Title::Genres`

The website notes that the most movies were entered manually and that errors or inconsistencies may exist.

**Data Pre-Processing**

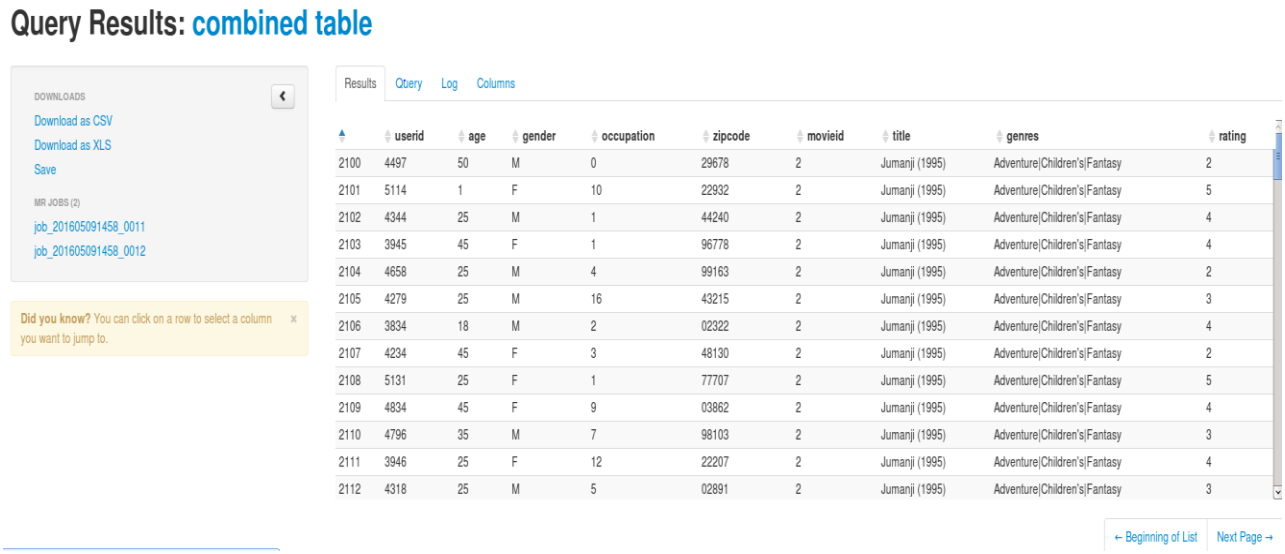
I sought to combine the three files using Hue, with its SQL-like capacity for query and analysis, to correlate age groups with movie users. However, I could not upload and combine the files in a clean, logical manner because of their double colon format ( UserID: :MovieID: :Rating: :Timestamp). I wrote two classes (ModifyFile.java and ModifyFileMapper) to upload and combine the files without the double colons and miscellaneous errors. I instead replaced the necessary demarcations with commas, as the following code demonstrates:

```
String line = value.toString();

        line = line.replaceAll("::", ",");

            text.set(line);
            context.write(null, text);
        }
```

It is, of course, worth noting how I combined these tables. Individually, I had three tables: user, movie, and ratings. I used the following SQL query on the Hue editor to combine the three files:

*SELECT user.userid, user.age, user.gender, user.occupation, user.zipcode,movie.movieid, movie.title, movie.genres, ratings.rating FROM user JOIN ratings ON(user.userid = ratings.userid) JOIN movie ON(movie.movieid = ratings.movieid)*

The end result was the following combined table, with a screenshot taken at random:




This set me up for my MapReduce (MR) algorithm and subsequent analysis.

## MR Algorithm

I wrote a program named MovieLensProject that correlated specific age groups with movieID's. This MR program, like most the programs I wrote during the first half of this course- has three functions: a map function (see MovieMapper code), a reduce function (see MovieReduce code) , and a driver function (see MovieDriver code) that executes the program.

The mapper class uses four parameters that specify the input key, input values, output key, and output value types of the map function. The input key is a long integer offset, the input value is a line of text, the output key is the age group as text, and the output value is the sum as an IntWritable. The mapper focuses on the pertinent combined from the combined file to, among other things, correlate age with the movie id. This is accomplished in the try-catch block,

as seen below:

```
@Override
public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

    try {
    //this examines the relevant info that I used in the SQL statement to
    //combine the data sets (see paper), such as userID and age

        String[] record = value.toString().split(",");

        if (Integer.parseInt(record[5]) > 0){
            age.set(record[1]);
            context.write(age, movieID);
        }
        context.getCounter(Movie.Total_Movies).increment(1);
    }  catch (ArrayIndexOutOfBoundsException e2) {
        context.getCounter(Movie.BAD_RECORD).increment(1);
    }  catch (NumberFormatException e3) {
    }
```

The code contains two counters. One focuses on the total movies watched, an obvious necessity for this exercise. The other counter is for bad records, as required for the project. Also, note that I used “Integer.parseInt(record[5])>0”as part of my “if” statement. The “5” in this scenario refers to the number of columns parsed from the combined tables information, and allows me to hone in on the correlation between age and total movies watched (the first counter).

The reduce function is similarly defined using a Reducer. Four formal type parameters are used to specify the input and output types, this time for the reduce function. As is required, the the input types in the Reducer match the output types of the map function, Text and IntWritable. The two output types of the reduce function are Text and IntWritable, which are outputs of age and maximum number of movies.

```
public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {

    int count = 0;
    for (IntWritable value : values) {
        count = count + value.get();
    }
    intWritable.set(count);
    context.write(key, intWritable);
```

As seen above, an enhanced for loop is used in the reducer to iterate through the data and correlate maximum count with age.

The driver program helps find the maximum number of movies watched per age group by executing the Mapper and Reducer code. Notably, the program implements the Tool interface, so we gain the benefit of being able to set the options supported by GenericOptionsParser. The job name is set and then executed with checks and balances, such as whether the MapReduce job finishes successfully, or if the appropriate amount of command-line parameters are passed (see below).

```
public static void main(String[] args) throws Exception {

    /*
     * The MapReduce job starts, and we wait for it to finish. If it finishes
     * successfully, return 0. If not, return 1.
     */
    int exitCode = ToolRunner.run(new MovieRunner(), args);
    System.exit(exitCode);
    /*
     * Validates that two arguments were indeed passed from the command line.
     */
    if (args.length != 2) {
        System.out
                .printf("Usage: MovieDriver <input dir> <output dir>\n");
        System.exit(-1);
    }
```

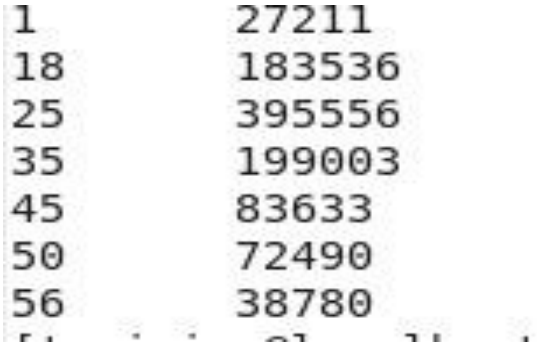
The code for the driver class –like the mapper and reducer classes- was informed by my work on the Maximum Word Length program earlier in the semester.

**Output**

So, after completing the pre-processing and compiling my MovieLensProject Java program, I was ready to find out the relationship between age group and movies watched. I put my Java program into a jar file and ran the necessary Hadoop commands against the dataset. My final command before achieving the results was:

```
[training@localhost ~]$ hadoop fs –cat result9/part-00000
```

The results listed two columns: first, age; second, number of movies watched by the user. The results were as follows.



```
1       27211
18      183536
25      395556
35      199003
45      83633
50      72490
56      38780
```

Thus, people in the age group 18-24 were most likely to watch movies. Those “under 18” were least likely; and, notably, those 56+ were the second least likely. I will discuss these results more in the conclusion.

**Verifying the Output**

In week four of our class, we discussed MRUnit testing which, according to the slides, “brings the JUnit test framework to MapReduce code.” I found this concept very useful and created some tests for my code. My driver test code had three methods to test: “testMapper, ” “testReducer,” and “testMapReduce.” Here is an excerpt from code, showing the “testReducer”:

```
public void testReducer() {


     // For this test, the reducer's input will be : 25 1 45 2.
     //The expected output is: 45 45 2.

    List<IntWritable> values = new ArrayList<IntWritable>();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));
    reduceDriver.withInput(new Text("45"), values);
    reduceDriver.withOutput(new Text("45"), new IntWritable(2));
    reduceDriver.runTest();
```

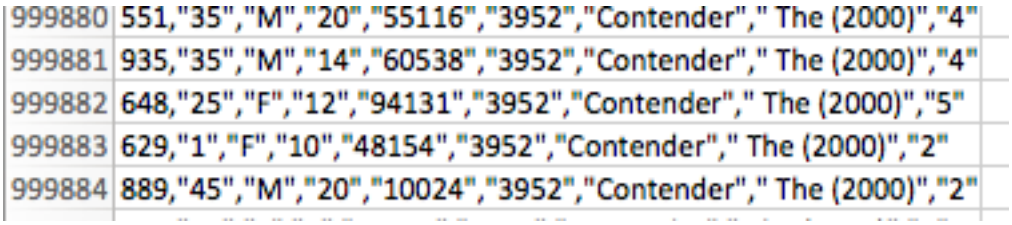
All three tests passed giving me zero errors. Significantly, I entered different inputs in the MRUnit suite and always returned the expected output.

Furthermore, I simply verified that the total ratings. According to the website, and as noted at the beginning of this paper, there are exactly 1,000,209 anonymous ratings in this dataset. It stands to reason that the sum total from the second column of the output results would equal 1,000,209. Indeed, this is the case: 38,780 + 72,490 + 83,633 + 199,003 + 395,556 + 183,536 + 27,211 equals 1,000,209.

**Bad Data Issues**

I mentioned the issue of bad data/records while discussing my MR mapper code. As we discussed in week 5, counters are a lightweight method for gauging errors and are statically defined via an enum. I therefore used an enum and a getCounter method to catch potential bad records. My code worked and I did not find any.

I downloaded my combine table as Excel sheet to further corroborate. The table contained the relevant schema –i.e. userId, age, etc. - and the necessary number of rows (1,000,209). Looking through the data, I noticed data that was not “bad” in the corrupt sense of the word, but worth noting:




In row 999883, the second column, "age," is listed as 1. Granted a one year old could watch the movie *The Contender*-what infant doesn't love Sam Eliot and Gary Oldman? - but I doubt he or she would have written reviews for a movie website. There are more instances of infants/toddlers watching and reviewing movies. Recall, the website itself says that errors "may exist" since the movies were largely entered by hand. This begs the question, is this the only reason the age input was skewered? What's the scale of error input and how does it impact the results?

**What I Would Have Done Differently…**

I started this paper on schedule and started coding for the project first which, in my opinion, was the most important component. However, I encountered numerous family/personal problems this semester-ones which interfered with my ability to complete work assignments to my best ability in the given time-frame. For instance, I was not able to fully resolve all the requirements for the paper, such as a detailed explanation of scale/performance and a better explanation of data pre-processing bad data. Also, I had initially hoped to answer questions beyond age, such as which age group was more likely to watch a particular genre of a movie. Ideally, I would have also liked to test another dataset to see the difference in my results. In any case, the quandary of balancing family and school/work problems is something I will face in the future but hopefully not often. Nevertheless, I learned a lot from this project and enjoyed working on it.

## Conclusion

Based on my dataset, which age group watches the most movies? From most to least: 25-34, 35-44, 18-24, 45-49, 50-55, 56+, and under 18. A couple of issues must be raised. First, as mentioned in the bad data section, there are one year old reviewers mentioned in the dataset. How many other age-related entry errors are there and how would they skew the results? Second, this is an internet-based movie review site from the year 2000. Arguably, especially sixteen years ago, tech-savvy "youth" were far more likely to use a website like MovieLens than were people in their 50s or older. If I wanted a more comprehensive picture, I would have to compare my dataset results with other studies conducted the relationship between movie viewership and age. However, those are questions beyond the scope of this paper. Based on my dataset, the age group 25-34 were the most avid movie reviewers/watchers when this information was compiled.