CIS 133 – NoSQL

Apoorva Sharma (G01110489)

Project Report

Problem Statement

When consumer is making a purchase decision, usually they look for aides that help them in doing so. One of the common aides is product review by other users who have used the product in past. When the product is popular or used by a large number of users, the number of reviews are huge. Generally it is not practical to go through all the reviews to make a decision. It is highly desirable if there is a mechanism to filter out irrelevant reviews. One such tool can be reviewer's rating history.

In this project, average review score is calculated based on reviewer's history of reviews. To understand this better, let us take an example of review website which provides user reviews of some products. User1 is a regular user and provide reviews very often. If User1 on an average gives a rating of 3 out of 5, then User1's average score is 3. Now if User1 has reviewed a product P and provided rating 3, then this review is not much helpful as it falls in User1's average case bucket. Now consider another product Q which is rated by User1 as 1 out of 5, this review is much more helpful as it is outside the average case rating of User1. In other words, the product must really deserve lower rating as the user who generally provided rating of 3 has given a comparatively lower rating.

Motivation

This problem is interesting as it focuses on empowering consumers to make informed decision by filtering noise and providing specific information user is looking for. When someone wants to hire someone or buy something, they want to make sure the product or person or service is as per the expectation. To ensure this, consumer looks for reviews. If the reviews are not helpful, consumer can make wrong choices. The solution proposed provides a way to determine the usability of review based on reviewer's history of reviews.

Project Dataset

The dataset used in this project contains movie reviews provided by users on MovieLens website. The reviews are provided as ratings based on 5 points scale with 0.5 increments. Each user provided ratings to at least 20 movies. The data is generated from 1995 to 2015. Current dataset was generated on October17, 2016. Dataset contains 27278 movies, 138,493 reviewers and 20,000,263 reviews.

Dataset Format

Dataset is a collection of 6 files in CSV format. For the current project only **ratings.csv** is required.

Review data format: *(`userId,movieId,review,timestamp`)*
Example: (1,2,3.5,1112486027)

Dataset Source

The dataset is available for public download from grouplens research group (Social Computing Research at University of Minnesota) website - https://grouplens.org/datasets/movielens/. Detailed description of data set is available on webpage (http://files.grouplens.org/datasets/movielens/ml-20m-README.html).

Dataset Reference

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

Similar Dataset Applications

The current data set is obtained from MovieLens website which provides movie recommendation based on user reviews. The same type of data is also generated by movie streaming service providers like Netflix, amazon who have large number of active users.

The problem under consideration is not limited to movie streaming service providers only. This can be expanded to a variety of domains. For example, any website that allows users to rate a product or service can use this approach to provide consumers with reviewer's average rating. Examples include Yelp, ebay, Walmart, etc.

Project Description

This project focusses on identifying and filtering out non-relevant reviews based on reviewer's history. Given dataset comprises of 20Million+ reviews. For each reviewer, average rating is generated. The relevancy of a rating given by a reviewer is determined by considerable deviation (diff) from his average rating.

$$diff = |AR - CR|, \text{ where}$$

- AR = Average Rating for a reviewer.
- CR = Current rating given by the reviewer.

Since ratings are provided on a scale of 5 with increments of 0.5, deviation is taken as *±1.0* (40% of Range) for this scale. Following conditions are used to mark review helpful or not helpful:

If (diff > 1.0) ➔ Rating is useful

If (diff <= 1.0) ➔ Rating is not useful

An important point to note here is that the proposed method should not be applied directly to a given review set. It should be applied only as an optimization technique, when certain conditions are met, which include:

- Review data set is substantial so that it is beneficial to filter and present only relevant reviews to end user.
- Reviewer's rating history is big enough to determine average case range.

Implementation

Software Tools

Following tools were used while implementing the project –

- **Apache Hadoop Map-Reduce framework**: This framework was used to load and process input dataset. Dataset was loaded from local file system.
- **Apache Hadoop Distributed File System**: The output set generated by Map-Reduce was uploaded to HDFS. Files in HDFS are easily accessible from Pig for performing analysis operation using Pig commands.
- **Apache Pig**: This tool provides SQL like layer on top of Map-Reduce. The dataset generated by Map-Reduce was analyzed using various commands offered by Pig.
- **Eclipse IDE**: Eclipse Development Environment provided as part of Cloudera Virtual Machine was used to create java based Map-Reduce project.

Execution Steps

1. **Pre-processing and Data Loading**: Data is provided in Comma Separated Values format. Data file '*ratings.csv*' is stored directly inside Cloudera VM's local file system.

2. **Data Processing**: Input dataset is processed in a number of steps utilizing different tools offered by Apache Hadoop framework. Sample subset of input dataset to be used for explaining the process:

Table 1: Sample data set

UserId	MovieId	Ratings (0 to 5)	Timestamp
2	260	5.0	974821014
2	480	5.0	974820720
2	541	5.0	974821014
2	589	5.0	974820658
2	891	2.0	974820969
5	104	2.0	851526992
5	140	2.0	851527012
5	224	2.0	851527650
5	235	3.0	851527723
5	260	5.0	851527608

AR Calculation: The first step was to obtain Average Rating (AR) for each user. Each user provided a number of reviews and to identify each user's average rating following calculation was required –

```
//Pseudocode
For each user U{
        Let $S^U$ = sum of all reviews R from dataset where userId=U, $0<=R<=5$
        Let $C^U$ = total number of reviews from dataset where userId=U
        $AR^u = S^U/C^U$, where $AR^U$ = AR for U
}
```

To implement this procedure **Map-Reduce framework** was used.

- **Map Phase**:- Mapper class was defined as follows:

  - *UserToRatingMapper* class extends Hadoop Mapper class and provides a *map* function.
  - This map function takes in lines from ratings.csv in format *(userId,movieId,review,timestamp)* as value.
  - From value text, user and review fields are extracted.
  - *map* emits out (key=user, value=review) pair.

```
//Mapper class implementation
public class UserToRatingMapper extends Mapper<LongWritable, Text, Text,
DoubleWritable> {
        private Text user = new Text();
        private DoubleWritable rating = new DoubleWritable();

        @Override
        public void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
                StringTokenizer itr = new StringTokenizer(value.toString());
                while (itr.hasMoreTokens()) {
                        String [] ratingTuple=itr.nextToken().split(",");
                        user.set(ratingTuple[0]);
                        rating.set(Double.parseDouble(ratingTuple[2]));
                        context.write(user, rating);
                }
        }
}
```

- **Explaining Map phase via sample subset** :
    - Dataset (userid, movieid, review score, timestamp) is provided as input to Map phase.
    - Output is generated as a set of key-value pairs.
    - UserId is emitted as key and current review score as value.

Input		Map output
2, 260, 5.0, 974821014		key= 2, value= 5.0
2, 480, 5.0, 974820720		key= 2, value= 5.0
2, 541, 5.0, 974821014		key= 2, value= 5.0
2, 589, 5.0, 974820658		key= 2, value= 5.0
2, 891, 2.0, 974820969	Map	key= 2, value= 2.0
5, 104, 2.0, 851526992		key= 5, value= 2.0
5, 140, 2.0, 851527012		key= 5, value= 2.0
5, 224, 2.0, 851527650		key= 5, value= 2.0
5, 235, 3.0, 851527723		key= 5, value= 3.0
5, 260, 5.0, 851527608		key= 5, value= 5.0

- **Magic Shuffle**: Data with same key gets lumped (shuffled) together.

key= 2, value= 5.0

key= 2, value= 5.0

key= 2, value= 5.0

key= 2, value= 5.0

key= 2, value= 2.0

key= 5, value= 2.0

key= 5, value= 2.0

key= 5, value= 2.0

key= 5, value= 3.0

key= 5, value= 5.0

Shuffle

key=2, values=[(5.0), (5.0), (5.0), (5.0), (2.0)]

key=5, values=[(2.0), (2.0), (2.0), (3.0), (5.0)]

- **Reduce Phase**:- Reducer class was defined as follows:
  - Reducer class *UserRatingReducer* is extended from Hadoop's Reducer class.
  - Reducer method *reduce* is written. It accepts user as key and Iterable object of all reviews given by user.
  - All reviews are traversed and average review score is obtained.
  - *reduce* emits out key=user and value=average review score.

```
//Reducer class implementation
public class UserRatingReducer extends Reducer<Text, DoubleWritable, Text,
DoubleWritable> {
    private DoubleWritable average = new DoubleWritable();

    @Override
    public void reduce(Text key, Iterable<DoubleWritable> values, Context context)
throws IOException, InterruptedException {
            double sum = 0;
            double avg=0;
            int count=0;
            for (DoubleWritable val : values) {
                    sum += val.get();
                    count+=1;
            }
            avg=sum/count;
            average.set(avg);
            context.write(key, average);
    }
}
```

- **Explaining Reduce phase via sample subset**: In this phase, a reducer function is invoked for each key. Input to each reducer task is provided as a (key, list of values) pair. Average rating given by a reviewer (key) can be calculated by traversing over this list. Output of this phase is average rating given by each reviewer *(userId, AR)*:

key= 2, values=[(5.0), (5.0), (5.0), (5.0), (2.0)] $\xrightarrow{\text{Reducer 1}}$ (2, 4.4)

key= 5, values=[(2.0), (2.0), (2.0), (3.0), (5.0)] $\xrightarrow{\text{Reducer 2}}$ (5, 2.8)

- **Hadoop Program Execution**

A main Driver java file is then created that sets mapper and reducer class for Hadoop jobs. All files are compiled and jar file is created. Hadoop command is then executed as-

```
hadoop jar jar/user-rating-average.jar Driver /user/cloudera/dataset/ratings.csv
/home/cloudera/workspace/NoSQL/output
```

Output of this stage is set of tuple in format **(userId, AR).** This output file is dumped to local file system from where it was uploaded to **HDFS** using put command –

```
hdfs dfs -put /home/cloudera/workspace/NoSQL/output/out  /user/cloudera/mapr_out
```

Table 2: Output of MapReduce for sample subset - User wise Average Rating Score

UserId	AR
2	4.4
5	2.8

Data Analysis

The data obtained after Map-Reduce processing was in required format. It contained user id and corresponding Average Rating (AR) for that user. Next step was to analyze this data based on proposed method –

$$diff = |AR - CR|, \text{ where}$$

- AR = Average Rating for a reviewer.
- CR = Current rating given by the reviewer.

Deviation window is considered as *±1.0* (40% of range). For the current project:

If (diff > 1.0) ➔ Rating is useful

If (diff <= 1.0) ➔ Rating is not useful

Objective of analysis is to answer questions like –

i. How many reviews in given dataset can be filtered out based on proposed method?
ii. Determine for each movie if a substantial number of reviews can be filtered out?
iii. Determine for each user if a substantial subset of reviews can be filtered out?

Apache Pig is used to load both initial dataset file (ratings.csv) and output of Map-Reduce phase. Following operations are then performed using **grunt shell** to determine answers to above three questions –

i. **How many reviews in given dataset can be filtered out based on proposed method?**

Analysis

//load map-reduce output file as out alias
out = LOAD 'hdfs://quickstart.cloudera:8020/user/cloudera/mapr_out/out' AS (user:int, average:double);

//load ratings.csv file as ratings alias
ratings = LOAD 'hdfs://quickstart.cloudera:8020/user/cloudera/dataset/ratings.csv' USING PigStorage(',') AS (user:int,movie:int,rating:double);

//join the two loaded aliases
out_ratings = JOIN out BY user,ratings BY user;
//delta is the final joined alias with diff column
//diff is the absolute difference between current review and average review
delta = FOREACH out_ratings GENERATE out::user,out::average,ratings::movie,ratings::rating,ABS(average - rating) AS diff:double;

Table 3: Sample subset representation of delta

Reviewer ID	Average Rating(AR)	Movie ID	Current Movie Rating (CR)	Diff =|AR – CR|
2	4.4	260	5.0	0.6
2	4.4	480	5.0	0.6
2	4.4	541	5.0	0.6
2	4.4	589	5.0	0.6
2	4.4	891	2.0	2.4
5	2.8	104	2.0	0.8
5	2.8	140	2.0	0.8
5	2.8	224	2.0	0.8
5	2.8	235	3.0	0.2
5	2.8	260	5.0	2.2

//delta_filter is the final joined filtered alias where deviation window = ±1.0
delta_filter = FILTER delta BY diff > 1.0;

//verify schema of delta_filter
grunt> describe delta_filter;
delta_filter: {out::user: int,out::average: double,ratings::movie: int,ratings::rating: double,diff: double}

Table 4: delta_filter (Derived from delta table (Table 3) after applying filter= diff>1)

Reviewer ID	Average Rating(AR)	Movie ID	Current Movie Rating (CR)	Diff =|AR – CR|
2	4.4	891	2.0	2.4
5	2.8	260	5.0	2.2

//count the number of useful reviews after applying filter
filter_group = GROUP delta_filter ALL;
filter_count = FOREACH filter_group GENERATE COUNT(delta_filter);
DUMP filter_count;
//number of useful reviews found after processing total reviews (20000263)
(5462580)

ii. **Determine for each movie if a substantial number of reviews can be filtered out?**

Analysis
//create movie first alias - movie id and filtered_count
//filtered_count is number of useful reviews for each movie
filtermovie_group = GROUP delta_filter BY movie;
filtermovie_count = FOREACH filtermovie_group GENERATE group AS movie, COUNT(delta_filter) AS filtered_count:int;

//create movie alias two - movie id and total count
//total count is total number of reviews for each movie
movie_group = GROUP ratings BY movie;
movie_count = FOREACH movie_group GENERATE group AS movie, COUNT(ratings) AS total:int;

//join 2 movie aliases to get final movie alias - movie_data (movie, total count, filtered count)
movie_join = JOIN movie_count BY movie LEFT OUTER, filtermovie_count BY movie;

```
movie_data = FOREACH movie_join GENERATE movie_count::movie AS movie,
movie_count::total AS total, filtermovie_count::filtered_count AS filtered_count;

//verify schema
grunt> describe movie_data;
movie_data: {movie: int,total: int,filtered_count: int}

//create subset of movie_data of size 10
movie_top = LIMIT movie_data 10;
DUMP movie_top;
```

```
//subset of first 10 movies from output
(1,49695,12986)
(2,22243,5137)
(3,12735,3485)
(4,2756,979)
(5,12161,3237)
(6,23899,6218)
(7,12961,2944)
(8,1415,319)
(9,3960,975)
(10,29005,4916)
```

Table 5: Representation of movie_top

Movie ID	Total Reviews	Filtered Useful Reviews
1	49695	12986
2	22243	5137
3	12735	3485
4	2756	979
5	12161	3237
6	23899	6218
7	12961	2944
8	1415	319
9	3960	975
10	29005	4916

The dumped subset is of **format {movie: int,total: int,filtered_count: int}**. It is seen that total number of reviews are substantially reduced to smaller set of filtered useful reviews for all movies.

iii. Determine for each user if a substantial subset of reviews can be filtered out?

Analysis

```
//create user alias one - user id and filtered count
filteruser_group = GROUP delta_filter BY user;
filteruser_count = FOREACH filteruser_group GENERATE group AS user,
COUNT(delta_filter) AS filtered_count:int;

//create user alias two - user id and total count
user_group = GROUP ratings BY user;
user_count = FOREACH user_group GENERATE group AS user, COUNT(ratings) AS
total:int;

//join 2 user alias to get final user alias - user_data (user, total count, filtered count)
user_join = JOIN user_count BY user LEFT OUTER, filteruser_count BY user;
user_data = FOREACH user_join GENERATE
user_count::user,user_count::total,filteruser_count::filtered_count;

//verify schema
grunt> describe user_data;
user_data: {user_count::user: int,user_count::total: int,filteruser_count::filtered_count:
int}

//create subset of user_data of size 10
user_top = LIMIT user_data 10;
DUMP user_top;
```

//subset of 10 users from output

```
(1,175,4)
(2,61,8)
(3,187,42)
(4,28,3)
(5,66,16)
(6,24,9)
(7,276,67)
(8,70,23)
(9,35,17)
(10,38,5)
```

Table 6: Representation of user_top

User ID	Total Reviews	Filtered Useful Reviews
1	175	4
2	61	8
3	187	42
4	28	3
5	66	16
6	24	9
7	276	67
8	70	23
9	35	17
10	38	5

Here, dumped output is of the **format {user_count::user: int,user_count::total: int,filteruser_count::filtered_count: int}.** It is seen that total number of reviews from each user are substantially reduced to smaller set of filtered useful reviews.

Observations

[1] Initial dataset contained 20Million+ reviews. After filtering out non-relevant reviews, dataset was reduced to ~5.5Million reviews.

Table 7: Dataset Reduction

Dataset Reviews (Initial Size)	Filtered Useful Reviews	Useful Dataset Size after Filtering
20,000,263	5,462,580	27.31% of Original Size

[2] **Movie specific results:** For each movie, it is observed that a large number of reviews can be filtered out (refer Table 5). This is further demonstrated using Table 8 for reviews given to a movie with MovieId=5089. From table 8, original review set of size 25 is reduced to useful review set of size 14, reducing the actual set to 56%.

Table 8: Review Set Analysis for Movie 5089

#	User Id	Average Rating	Movie Id	Current Rating	Diff	Is Review Helpful?
1	741	2.803345388788427	5089	0.5	2.303345388788427	**Yes**
2	25669	2.726525821596244	5089	3.0	0.273474178403756	No
3	32221	3.557758620689655	5089	1.0	2.557758620689655	**Yes**
4	37530	3.5155817174515236	5089	2.0	1.5155817174515236	**Yes**
5	39647	3.6394910461828465	5089	3.0	0.6394910461828465	No
6	46951	3.8132250580046403	5089	4.0	0.1867749419953597	No
7	48405	3.4861239592969473	5089	0.5	2.9861239592969473	**Yes**
8	54259	3.608240223463687	5089	3.5	0.108240223463687	No
9	60887	1.7342057761732852	5089	1.0	0.7342057761732852	No
10	66533	3.5295378751786566	5089	4.5	0.9704621248213434	No
11	67346	2.415994840374073	5089	0.5	1.915994840374073	**Yes**
12	83090	2.4049139098471657	5089	2.0	0.4049139098471657	No
13	86839	3.1233766233766236	5089	3.0	0.12337662337662358	No
14	89364	3.1106017191977076	5089	2.0	1.1106017191977076	**Yes**
15	91867	3.5285761480013216	5089	1.0	2.5285761480013216	**Yes**
16	91904	2.4789915966386555	5089	1.0	1.4789915966386555	**Yes**
17	107326	3.5010982114841545	5089	2.0	1.5010982114841545	**Yes**
18	107640	2.9950980392156863	5089	3.5	0.5049019607843137	No
19	109310	3.8714285714285714	5089	5.0	1.1285714285714286	**Yes**
20	118205	3.2790685109141995	5089	2.0	1.2790685109141995	**Yes**
21	123352	3.2332463011314188	5089	3.0	0.23324630113141875	No
22	124585	2.642857142857143	5089	4.0	1.3571428571428572	**Yes**
23	128893	2.7453846153846153	5089	1.0	1.7453846153846153	**Yes**
24	130459	3.0638433981576254	5089	2.5	0.5638433981576254	No
25	130809	3.2639939485627836	5089	2.0	1.2639939485627836	**Yes**

[3] **User specific results**: It is observed that total number of reviews given by each user are substantially reduced to number of useful reviews (refer Table 6). This is further demonstrated using Table 9 for reviews given by user with userId=9. Original review set of size 35 is reduced to useful review set of size 17, reducing the actual set by ~50%.

Table 9: Review Set Analysis for User 9

#	User Id	Average Rating	Movie Id	Current Rating	Diff	Is Review Helpful?
1	9	3.057142857142857	858	5.0	1.942857142857143	**Yes**
2	9	3.057142857142857	356	4.0	0.9428571428571431	No
3	9	3.057142857142857	1219	3.0	0.05714285714285694	No
4	9	3.057142857142857	1911	3.0	0.05714285714285694	No
5	9	3.057142857142857	1923	4.0	0.9428571428571431	No
6	9	3.057142857142857	1997	5.0	1.942857142857143	**Yes**
7	9	3.057142857142857	2279	3.0	0.05714285714285694	No
8	9	3.057142857142857	2605	2.0	1.057142857142857	**Yes**
9	9	3.057142857142857	2683	3.0	0.05714285714285694	No
10	9	3.057142857142857	2688	3.0	0.05714285714285694	No
11	9	3.057142857142857	2706	4.0	0.9428571428571431	No
12	9	3.057142857142857	2710	3.0	0.05714285714285694	No
13	9	3.057142857142857	2719	2.0	1.057142857142857	**Yes**
14	9	3.057142857142857	2722	2.0	1.057142857142857	**Yes**
15	9	3.057142857142857	2840	3.0	0.05714285714285694	No
16	9	3.057142857142857	2841	4.0	0.9428571428571431	No
17	9	3.057142857142857	2959	5.0	1.942857142857143	**Yes**
18	9	3.057142857142857	3016	2.0	1.057142857142857	**Yes**
19	9	3.057142857142857	3785	2.0	1.057142857142857	**Yes**
20	9	3.057142857142857	3798	5.0	1.942857142857143	**Yes**
21	9	3.057142857142857	3857	3.0	0.05714285714285694	No
22	9	3.057142857142857	3908	2.0	1.057142857142857	**Yes**
23	9	3.057142857142857	3979	3.0	0.05714285714285694	No
24	9	3.057142857142857	3994	3.0	0.05714285714285694	No
25	9	3.057142857142857	3999	2.0	1.057142857142857	**Yes**
26	9	3.057142857142857	4022	4.0	0.9428571428571431	No
27	9	3.057142857142857	4030	2.0	1.057142857142857	**Yes**
28	9	3.057142857142857	4034	1.0	2.057142857142857	**Yes**
29	9	3.057142857142857	4148	5.0	1.942857142857143	**Yes**
30	9	3.057142857142857	4369	4.0	0.9428571428571431	No
31	9	3.057142857142857	4483	2.0	1.057142857142857	**Yes**
32	9	3.057142857142857	4502	1.0	2.057142857142857	**Yes**

33	9	3.057142857142857	4509	3.0	0.05714285714285694	No
34	9	3.057142857142857	4519	2.0	1.057142857142857	**Yes**
35	9	3.057142857142857	4533	3.0	0.05714285714285694	No

Conclusion

- If the number of reviews for a product or service is substantial, it is highly desirable to provide a subset of only helpful reviews to consumer. A method is proposed to achieve this goal in the current project.
- As shown in observations, a large set of reviews can be substantially reduced to more relevant data set. This reduction parameter is dependent on size of deviation window.
- From the dataset used (size ~20Million+) and deviation window taken as 40%, useful review set is obtained as ~27% of actual data size.
- If size of the deviation window is increased, less number of outliers are obtained (which are identified as useful reviews), further reducing the number of useful reviews.
- The proposed method can be applied to other applications where end user (or customer) generated review content is present in substantial size, for example, Amazon, eBay, Yelp, etc.
- Framework like Hadoop along with other tools used in this project like HDFS, Apache Pig provides an edge when it comes to Big Data processing. These tools made technical implementation of the current project much simpler to execute.