Engineering Big-Data Systems INFO 7250

Recommender Movies System Analysis

PRESENTED BY: Weisimin Peng
Northeastern University

problem statement

Database link: https://grouplens.org/datasets/movielens/1m/

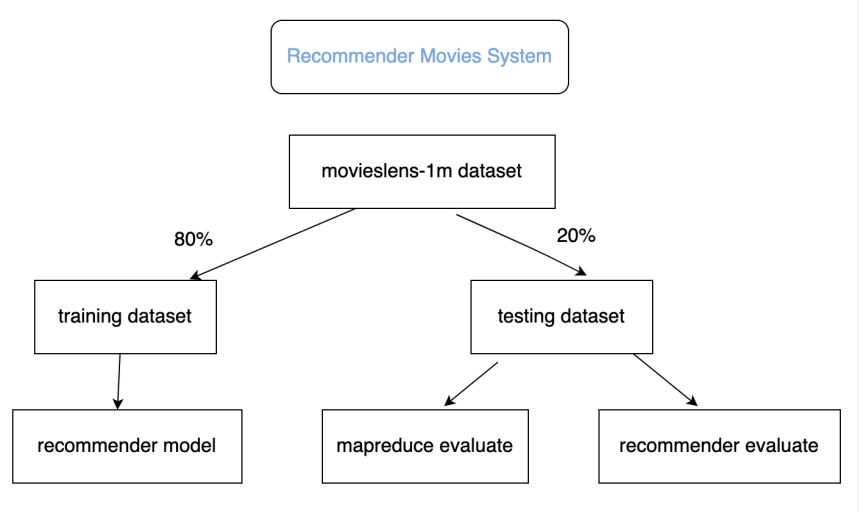
✦ As big data analysis becomes more and more popular, one of the most popular applications is to recommend items for users through collaborative filtering algorithms.

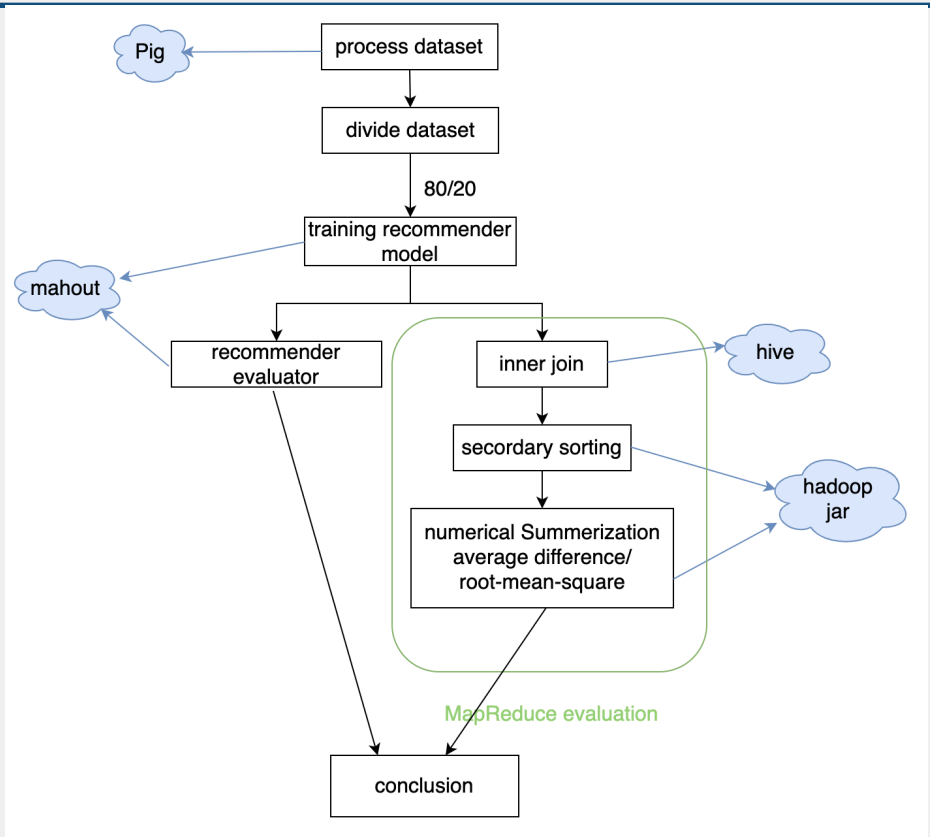
✦ Search for similar users in the data set according to the user's historical information and recommend the historical items of similar users to the target user to explore the user's potential preferences.

✦ This project uses Hadoop tools such as pig, mahout, hive, plus Hadoop MapReduce programmed in the Java programming language to implement a user's movie recommendation system and evaluate the results in multiple ways.

summary& Technical Description

Work flow





Process Dataset

✦ Because recommender model only need 3 variables(UserID, ItemID, Value), but the dataset have four avriables(userID, movieID, rating, timestamp), so I use pig to extract correct input format.

```
grunt> ratingData = load '/Users/weisiminpeng/Desktop/7250/Assignments/FinalProject/ml-1m/ratings.dat'
 using PigStorage(':') as (userid,temp1,movieid,temp2,rating,temp3,timestamp);
2020-08-13 14:56:36,875 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.c
hecksum is deprecated. Instead, use dfs.bytes-per-checksum
grunt> ratingData = foreach ratingData generate $0,$2,$4 ;
grunt> describe ratingData;
ratingData: {userid: bytearray,movieid: bytearray,rating: bytearray}
```

```
grunt> store ratingData into '/Users/weisiminpeng/Desktop/7250/Assignments/FinalProject/ml-1m/ratings-
modify' using PigStorage(',');
```

ratings.dat

```
1::1193::5::978300760
1::661::3::978302109
1::914::3::978301968
1::3408::4::978300275
1::2355::5::978824291
1::1197::3::978302268
```

ratings-modify

Name	Date Modified
_SUCCESS	Today at 15:01
part-m-00000	Today at 15:01

Pig

```
(1,1193,5)
(1,661,3)
(1,914,3)
(1,3408,4)
(1,2355,5)
```

training recommender model

mahout

```java
    public static void main(String[] args) throws IOException, TasteException{
//        load HDFS dataset
        DataModel model = new FileDataModel(new File(
                pathname: "/Users/weisiminpeng/Desktop/7250/Assignments/FinalProject/ml-1m/ratingsModifyTrain1.csv"));
        UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
        UserNeighborhood neighborhood = new NearestNUserNeighborhood( n: 6,
                similarity, model);
        Recommender recommender = new GenericUserBasedRecommender(model,
                neighborhood, similarity);
        PrintWriter writer = new PrintWriter(
                fileName: "/Users/weisiminpeng/Desktop/7250/Assignments/FinalProject/recommenderOutput/RecommenderTrainOutput1.csv");
        for (LongPrimitiveIterator iterator = model.getUserIDs(); iterator.hasNext(); ) {
            long userID = iterator.nextLong();
            List<RecommendedItem> recommendations = recommender.recommend(userID,
                    i: 20);
```

✦ For this project, I choose to use java program to implement recommender model

✦ Mahout is imported by maven;

✦ I choose user-based recommender system because the main purpose of this project is to recommend movies according to similar taste user;

✦For NearestUserNeighborhood, I choose 6 users as a notion of a group. Because for test 4-7, 6 users can have more satisfied result of evaluate score (the table will be showed later);

✦For recommender, I choose 20 movies;

evaluate recommender model

mahout

Using recommender evaluator

```
    public static void main(String[] args) throws Exception, TasteException {
        RecommenderEvaluator evaluator = new RMSRecommenderEvaluator();
//        RecommenderEvaluator evaluator = new AverageAbsoluteDifferenceRecommenderEvaluator();
//        load HDFS dataset
        DataModel model = new FileDataModel(new File(
                pathname: "/Users/weisiminpeng/Desktop/7250/Assignments/FinalProject/ml-1m/ratingsModify.csv"));
        RecommenderBuilder builder = new RecommenderBuilder() {
            @Override
            public Recommender buildRecommender(DataModel model) throws TasteException {
                UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
                UserNeighborhood neighborhood = new NearestNUserNeighborhood( n: 6,
                        similarity, model);
                Recommender recommender = new GenericUserBasedRecommender(model,
                        neighborhood, similarity);
                return recommender;
            }
        };
        double score = evaluator.evaluate(builder,  dataModelBuilder: null, model,  v: 0.8,  v1: 1.0);
        System.out.println(score);
    }
```

✦ 80% of dataset for training and 20% of dataset for testing;

evaluate recommender model

mahout

Using recommender evaluator

Gourp of users	4	5	6	7
Average Absolute Difference	0.846101	0.853719	0.848895	0.863059
RMS	1.126209	1.139984	1.126918	1.145227

✦ As we can see in the table, 6 would be the more satisfied number of group of users.

✦ 5 users and 7 users in a group both have higher average absolute difference and root mean square result.

evaluate recommender model

Using Hadoop jar

0. Code Idea

1. In order to make output more readable and understandable, I using inner join to join movie title and movie genres into training output and test dataset by movie id.

2. I choose userID as natural key, and movieID as composite key to implement secondary sort so that the out put will be more readable;

3. For further evaluating the model, I have selected categories comedy, romance, action and drama of the movie genres, respectively, and calculate the percentages of the above four genres of movies watched by the same user in the test data and training result data. So that I can find the similarity of this two different dataset.

4. CustomOutputTuple writable class to implement calculate average difference and root-mean-square in Reducer; I store ComedyCount, ActionCount, DramaCount, RomanceCount and totalCount for both testing and trainingdata, also I store title and type in writable class. Therefore, in each reducer, I can calculate the local average difference and local root-mean-square by store the count of four genres and the total count of the movies each user.

evaluate recommender model

Using Hadoop jar

0.1. dump data

hive

```
→  sbin hadoop fs -ls /finalproject/dataset
→  sbin hadoop fs -copyFromLocal /Users/weisiminpeng/Desktop/7250/Assignments/FinalProject/ml-1m/ratingsModify-test.csv
 /finalproject/dataset
→  sbin hadoop fs -copyFromLocal /Users/weisiminpeng/Desktop/7250/Assignments/FinalProject/ml-1m/movies.dat /finalproje
ct/dataset
→  sbin hadoop fs -copyFromLocal /Users/weisiminpeng/Desktop/7250/Assignments/FinalProject/recommenderoutput/Recommende
rTrainOutput.csv /finalproject/dataset
→  sbin hadoop fs -ls /finalproject/dataset
Found 3 items
-rw-r--r--   1 weisiminpeng supergroup     393216 2020-08-13 17:52 /finalproject/dataset/RecommenderTrainOutput.csv
-rw-r--r--   1 weisiminpeng supergroup     171308 2020-08-13 17:51 /finalproject/dataset/movies.dat
-rw-r--r--   1 weisiminpeng supergroup    2510681 2020-08-13 17:51 /finalproject/dataset/ratingsModify-test.csv
```

```
→  sbin hadoop fs -tail /finalproject/dataset/movies.dat
ick, The (1940)::Comedy
3930::Creature From the Black Lagoon, The (1954)::Horror
3931::Giant Gila Monster, The (1959)::Horror|Sci-Fi
3932::Invisible Man, The (1933)::Horror|Sci-Fi
3933::Killer Shrews, The (1959)::Horror|Sci-Fi
3934::Kronos (1957)::Sci-Fi
3935::Kronos (1973)::Horror
3936::Phantom of the Opera, The (1943)::Drama|Thriller
3937::Runaway (1984)::Sci-Fi|Thriller
3938::Slumber Party Massacre, The (1982)::Horror
3939::Slumber Party Massacre II, The (1987)::Horror
3940::Slumber Party Massacre III, The (1990)::Horror
3941::Sorority House Massacre (1986)::Horror
3942::Sorority House Massacre II (1990)::Horror
3943::Bamboozled (2000)::Comedy
3944::Bootmen (2000)::Comedy|Drama
3945::Digimon: The Movie (2000)::Adventure|Animation|Children's
3946::Get Carter (2000)::Action|Drama|Thriller
3947::Get Carter (1971)::Thriller
3948::Meet the Parents (2000)::Comedy
```

```
→  sbin hadoop fs -head /finalproject/dataset/ratingsModify-test.csv
3080,537,3
2607,608,5
2734,1302,4
4450,480,3
4471,2794,2
```

```
→  sbin hadoop fs -head /finalproject/dataset/RecommenderTrainOutput.csv
1,3481,5.0
1,1680,5.0
1,110,4.5
1,3259,4.5
1,912,4.5
2,858,5.0
```

evaluate recommender model

hive

Using Hadoop jar

1. Inner join

```
hive> use finalproject;
OK
Time taken: 0.021 seconds
hive> create table recommenderTrainOutput(userid INT, movieid INT, estimateRating float)
 row format delimited fields terminated by ',';
OK
Time taken: 0.054 seconds
hive> describe recommenderTrainOutput;
OK
userid                  int
movieid                 int
estimaterating          float
Time taken: 0.062 seconds, Fetched: 3 row(s)
```

```
hive> select * from recommendertrainoutput;
OK
1       3481    5.0
1       1680    5.0
1       110     4.5
1       3259    4.5
1       912     4.5
2       858     5.0
2       2804    4.5
2       3751    3.5
2       1196    3.5
2       1097    3.0
3       2028    5.0
3       3578    4.5
3       1954    4.0
3       1210    4.0
3       3555    4.0
4       1207    5.0
4       1193    5.0
4       1084    5.0
4       908     5.0
```

- ✦ load data into hive recommendertrainoutput & movies and movies&ratingstest;
- ✦ recommendertrainoutput / movies / ratingstest;
- ✦ only show recommendertrainoutput here

```
hive> load data inpath '/finalproject/dataset/RecommenderTrainOutput.csv' overwrite into
 table recommenderTrainOutput;
Loading data to table finalproject.recommendertrainoutput
OK
Time taken: 0.366 seconds
```

evaluate recommender model

hive

Using Hadoop jar

1. Inner join

```
hive> INSERT OVERWRITE directory'/finalproject/output1/'
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY '%'
    > STORED AS TEXTFILE
    > select r.userid, r.movieid,m.title,r.estimaterating, m.genres from recommenderTrainOu
tput r inner join movies m on(r.movieid = m.movieid);
→  sbin hadoop fs -head /finalproject/output1/000000_0
1%3481%High Fidelity (2000)%5.0%Comedy
1%1680%Sliding Doors (1998)%5.0%Drama|Romance
1%110%Braveheart (1995)%4.5%Action|Drama|War
1%3259%Far and Away (1992)%4.5%Drama|Romance
1%912%Casablanca (1942)%4.5%Drama|Romance|War
2%858%Godfather, The (1972)%5.0%Action|Crime|Drama
2%2804%Christmas Story, A (1983)%4.5%Comedy|Drama
2%3751%Chicken Run (2000)%3.5%Animation|Children's|Comedy
```

- ✦ do inner join to recommendertrainoutput & movies and movies&ratingstest;
- ✦ select r.userid, r.movieid,m.title,r. rating, m.genres from ratingsTest r inner join movies m on(r.movieid = m.movieid);

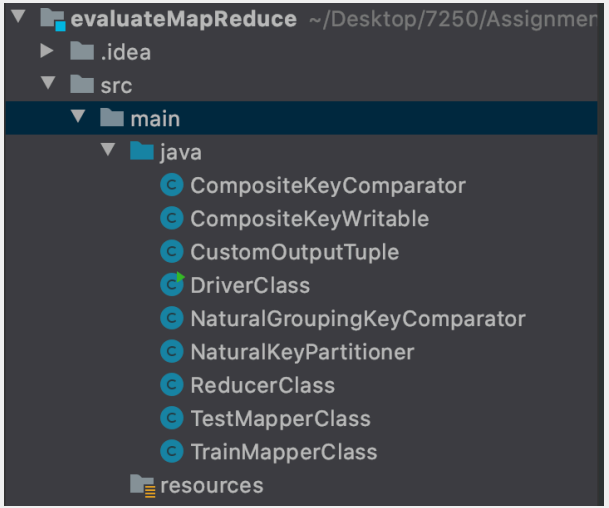
```
hive> INSERT OVERWRITE directory'/finalproject/output2/'
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY '%'
    > STORED AS TEXTFILE
    > select r.userid, r.movieid,m.title,r. rating, m.genres from ratingsTest r inner join
movies m on(r.movieid = m.movieid);
→  sbin hadoop fs -head /finalproject/output2/000000_0
3080%537%Sirens (1994)%3%Comedy|Drama
2607%608%Fargo (1996)%5%Crime|Drama|Thriller
2734%1302%Field of Dreams (1989)%4%Drama
4450%480%Jurassic Park (1993)%3%Action|Adventure|Sci-Fi
4471%2794%European Vacation (1985)%2%Comedy
5795%104%Happy Gilmore (1996)%1%Comedy
4948%1873%Mis?rables, Les (1998)%1%Drama
1001%1894%Six Days Seven Nights (1998)%2%Adventure|Comedy|Romance
```

evaluate recommender model

Hadoop MapReduce

Using Hadoop jar

JAVA classes



✦ secondary sorting:

```
job.setGroupingComparatorClass(NaturalGroupingKeyComparator.class);
job.setSortComparatorClass(CompositeKeyComparator.class);
job.setPartitionerClass(NaturalKeyPartitioner.class);
```

✦ custom writable:

CustomOutputTuple

✦ mapreduce:

```
MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class, TrainMapperClass.class);
MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class, TestMapperClass.class);

job.setMapOutputKeyClass(CompositeKeyWritable.class);
job.setMapOutputValueClass(CustomOutputTuple.class);

job.setReducerClass(ReducerClass.class);

job.setOutputFormatClass(TextOutputFormat.class);
Path outDir = new Path(args[2]);
TextOutputFormat.setOutputPath(job, outDir);

job.setOutputKeyClass(CompositeKeyWritable.class);
job.setOutputValueClass(CustomOutputTuple.class);
```

evaluate recommender model

Hadoop MapReduce

Using Hadoop jar

2. Secondary Sort

```java
public int compare(WritableComparable a, WritableComparable b) {
    CompositeKeyWritable ckw1 = (CompositeKeyWritable) a;
    CompositeKeyWritable ckw2 = (CompositeKeyWritable) b;

    int result = ckw1.getUserid().compareTo(ckw2.getUserid());
    return result;
}
```

```java
public int compare(WritableComparable a, WritableComparable b) {
    CompositeKeyWritable c1 = (CompositeKeyWritable) a;
    CompositeKeyWritable c2 = (CompositeKeyWritable) b;

    int result = -1 * c1.getMovieid().compareTo(c2.getMovieid());
    return result;
}
```

- For secondary sort, I choose userID as natural key, and movieID as composite key;
- Natural Key is on ascending order;
- Composite Key is on descending order;

```java
public int getPartition(CompositeKeyWritable key, CustomOutputTuple value, int noOfPartitions) {
    return key.getUserid().hashCode() % noOfPartitions;
      return key.getUserid().hashCode();
}
```

evaluate recommender model

Hadoop MapReduce

Using Hadoop jar

3. Numerical Summerization

i. CustomOutputTuple

```
public class CustomOutputTuple implements Writable{
    private Double averageDiff;
    private Double RMS;
    private int TrainComedyCount;
    private int TrainRomanceCount;
    private int TrainActionCount;
    private int TrainDramaCount;
    private int TrainTotal;
    private int TestComedyCount;
    private int TestRomanceCount;
    private int TestActionCount;
    private int TestDramaCount;
    private int TestTotal;
    private double rating;
    private String title;
    private String genres;
    private String type;
```

✦ in order to implement numerical summarization, I implemenmted a writable class, CustomOutputTuple

✦ Store count for four different genre and total number of movies each user for train and test separately to calculate the average difference and root mean square

evaluate recommender model

Hadoop MapReduce

Using Hadoop jar

3. Numerical Summerization

ii. Multiple inputs

✦ Also, I use multiple inputs because I need data in the test data set and training output at the same time;

✦ I have selected categories comedy, romance, action and drama of the movie genres, respectively, and calculate the percentages of the above four genres of movies watched by the same user in the test data and training result data;

✦ After that, I can calculate the average difference and root mean square to further analysis the outcome of recommender model;

```
MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class, TrainMapperClass.class);
MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class, TestMapperClass.class);
```

evaluate recommender model

Hadoop MapReduce

Using Hadoop jar

3. Numerical Summerization

iii. MapperClass

```
String[] genres = values[4].split( regex: "\\|");
for(String genre : genres) {
    if (genre != null) {
        if (genre.equals("Comedy")){
            result.setTestComedyCount(1);
        }
        if (genre.equals("Romance")) {
            result.setTestRomanceCount(1);
        }
        if (genre.equals("Action")) {
            result.setTestActionCount(1);
        }
        if (genre.equals("Drama")) {
            result.setTestDramaCount(0);
```

```
public class TestMapperClass extends Mapper<LongWritable, Text, CompositeKeyWritable, CustomOutputTuple> {

    CustomOutputTuple result = new CustomOutputTuple();
    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
```

✦ Because training output and testing data have same data stricture, so the Mapper Class code are almost the same, I just show the testing data Mapper class below;

✦ code to calculate the count of each genre;

evaluate recommender model

Hadoop MapReduce

Using Hadoop jar

3. Numerical Summerization

iii. ReducerClass

✦ initiate result

```
result.setAverageDiff(null);
result.setRMS(null);
result.setGenres(null);
result.setTitle(null);
result.setRating(0.0);
result.setTestActionCount(0);
result.setTestComedyCount(0);
result.setTestDramaCount(0);
result.setTestRomanceCount(0);
result.setTestTotal(0);
result.setTrainActionCount(0);
result.setTrainComedyCount(0);
result.setTrainDramaCount(0);
result.setTrainRomanceCount(0);
result.setTrainTotal(0);
result.setType("none");
```

✦ update result according to data type(e.g train/test/none)

```
    for(CustomOutputTuple text : values){
//          System.out.println(text);
        if(text.getType().equals("train")){
//              System.out.println("111111111111111");
            result.setTrainComedyCount(result.getTrainComedyCount()+text.getTrainComedyCount());
//              System.out.println("%%%%%%%%%%%%%%%%%%%%%");
            result.setTrainActionCount(result.getTrainActionCount()+text.getTrainActionCount());
            result.setTrainDramaCount(result.getTrainDramaCount()+text.getTrainDramaCount());
            result.setTrainRomanceCount(result.getTrainComedyCount()+text.getTrainRomanceCount());
            result.setTrainTotal(result.getTrainTotal()+text.getTrainTotal()) ;
        } else if(text.getType().equals("test")){
//              System.out.println("222222222222");
//              listTest.add(text);
            result.setTestComedyCount(result.getTestComedyCount()+text.getTestComedyCount());
            result.setTestActionCount(result.getTestActionCount()+text.getTestActionCount());
            result.setTestDramaCount(result.getTestDramaCount()+text.getTestDramaCount());
            result.setTestRomanceCount(result.getTestComedyCount()+text.getTestRomanceCount());
            result.setTestTotal(result.getTestTotal()+text.getTestTotal()) ;
        }else if(text.getType().equals("none")){
//              System.out.println("222222222222");
//              listTest.add(text);
            result.setTestComedyCount(result.getTestComedyCount()+text.getTestComedyCount());
            result.setTestActionCount(result.getTestActionCount()+text.getTestActionCount());
            result.setTestDramaCount(result.getTestDramaCount()+text.getTestDramaCount());
            result.setTestRomanceCount(result.getTestComedyCount()+text.getTestRomanceCount());
            result.setTrainComedyCount(result.getTrainComedyCount()+text.getTrainComedyCount());
            result.setTrainActionCount(result.getTrainActionCount()+text.getTrainActionCount());
            result.setTrainDramaCount(result.getTrainDramaCount()+text.getTrainDramaCount());
            result.setTrainRomanceCount(result.getTrainComedyCount()+text.getTrainRomanceCount());
            result.setTestTotal(result.getTestTotal()+text.getTestTotal()) ;
            result.setTrainTotal(result.getTrainTotal()+text.getTrainTotal()) ;
        }
    }
```

evaluate recommender model

Hadoop MapReduce

Using Hadoop jar

3. Numerical Summerization

✦ calculate differences of each genre in two dataset

```
private Double[] diff1 = new Double[]{0.0,0.0,0.0,0.0};
for(int i=0;i<diff1.length;i++){
    if(result.getTrainTotal() == 0 && result.getTestTotal() != 0){
        diff1[i] = new Double(Math.abs(0-(result.getTestComedyCount()/result.getTestTotal())));
    }
    if(result.getTrainTotal() != 0 && result.getTestTotal() == 0){
        diff1[i] = new Double(Math.abs((result.getTrainComedyCount()/result.getTrainTotal())));
    }
    if(result.getTrainTotal() == 0 && result.getTestTotal() == 0){
        diff1[i] = new Double( value: 0.0);
    }
    if(result.getTrainTotal() != 0 && result.getTestTotal() != 0){
        diff1[i] = new Double(Math.abs((result.getTrainComedyCount()/result.getTrainTotal())-
                (result.getTestComedyCount()/result.getTestTotal())));
    }
```

✦ average difference

```
double sum = 0.0;
for(int i=0;i<diff1.length;i++){
        System.out.println(diff[i]);
    sum += diff1[i];
}
  System.out.println("***************************");
result.setAverageDiff(sum/4);
```

✦ root-mean-square

```
double sumSquare = 0.0;
for(int i=0;i<diff1.length;i++){
    sumSquare += diff1[i]*diff1[i];
}
result.setRMS(new Double(Math.sqrt(sumSquare/4)));
```

evaluate recommender model

Hadoop MapReduce

Using Hadoop jar

Example

For user1:

	ComedyCount	RomanceCount	ActionCount	DramaCount	TotalMovieCount
train	3	3	2	5	10
test	2	1	1	2	4

Percentage of each genre:

train	0.3	0.3	0.2	0.5
test	0.5	0.25	0.25	0.5

Count/totalCount

Difference:

difference	0.2	0.05	0.05	0

Absolute average difference: (0.2 + 0.05 + 0.05 + 0) / 4 = 0.375

Root-mean-square(RMS): $\sqrt{((0.2 + 0.05 + 0.05 + 0) / 4)} = 0.6124$

evaluate recommender model

Hadoop MapReduce

Using Hadoop jar

3. Running & Output

✦ running in hadoop

```
→  sbin hadoop jar /Users/weisiminpeng/Desktop/7250/Assignments/FinalProject/codes/innerJoin/target/innerJoin-1.0-SNAPSHOT.jar
 DriverClass /finalproject/output1/000000_0 /finalproject/output2/000000_0 /finalproject/finalOutput
→  sbin hadoop fs -ls /finalproject/finalOutput
Found 2 items
-rw-r--r--   1 weisiminpeng supergroup          0 2020-08-14 07:50 /finalproject/finalOutput/_SUCCESS
-rw-r--r--   1 weisiminpeng supergroup     776242 2020-08-14 07:50 /finalproject/finalOutput/part-r-00000
```

```
→  sbin hadoop fs -head /finalproject/finalOutput/part-r-00000
549,988 averageDiff=1.0, RMS=1.0
1587,988        averageDiff=1.0, RMS=1.0
1306,988        averageDiff=1.0, RMS=1.0
4497,976        averageDiff=0.0, RMS=0.0
5680,976        averageDiff=0.0, RMS=0.0
4688,969        averageDiff=0.0, RMS=0.0
5260,969        averageDiff=0.0, RMS=0.0
475,969 averageDiff=0.0, RMS=0.0
2793,969        averageDiff=0.0, RMS=0.0
1699,969        averageDiff=0.0, RMS=0.0
879,969 averageDiff=0.0, RMS=0.0
1307,969        averageDiff=0.0, RMS=0.0
461,969 averageDiff=0.0, RMS=0.0
2607,969        averageDiff=0.0, RMS=0.0
5054,969        averageDiff=0.0, RMS=0.0
4854,969        averageDiff=0.0, RMS=0.0
1899,969        averageDiff=0.0, RMS=0.0
1717,969        averageDiff=0.0, RMS=0.0
4764,969        averageDiff=0.0, RMS=0.0
```

Conclusion

✦ At first, I consider to using inner join to get the intersection of two data set, training output and testing dataset. However, The intersection is null.

✦ Then I choose to use genres in each movie of each user, to calculate the similarity of percentage of four choosing genre in two dataset. I have selected categories comedy, romance, action and drama of the movie genres, respectively, and calculate the percentages of the above four genres of movies watched by the same user in the test data and training result data;

✦ As we can see in the output in last slide, the average difference and root-mean-square are both not so well of each user, which means that this two dataset is not that similar according to compare genre percentage.

We also can verify this conclusion above when there is no intersection between training output and testing dataset.

✦ Possible reasons:

1. There are few users in the data set and a large number of movies, resulting in very sparse data, which will cause loss of information and reduce the recommendation effect.
2. Some users in the training dataset but they don't have any rating on any movies, in real-time, it is new user, under this circumstance, model cannot find the similar user and do not know how to recommend movies will also have negative influence to evaluate result.

✦ Also, this project can further improvement by further modifying the parameter in model, such as recommender number, and taking more genre kind into consideration instead of four genre.

Thank you!

PRESENTED BY: Weisimin Peng
Northeastern University