A content-based movie recommendation system with massive parallel processing

Team members: Yu Zhao, Ting Wang

Abstract

Recommendation systems have become important information filtering components in our lives, especially in various e-commence applications. It seeks to predict the preferences of a user and rate of items from big data and make suggestions for lots of things, for example books, music, movies or news etc. In this project, we attempt to apply the Hadoop MapReduce program to construct a content-based movie recommendations system based on the content filtering approach. The system collects the information provided by both the user and two different movie datasets (ie, wikidata, movielen) and analyzes them. At each recommender, the system will apply MapReduce to go through all of the data sets and rate each movie by comparing its content with the user input. The recommendation list is sorted according to the ratings given to these movies at parallel processing of MapReduce. Last the system recommends the top 15 movies that is best suited to the users' requirement. We find that for our dataset, using this content-based movie filtering, basically the results of top 15 movies would agree with the users' inputs.

Introduction

Background and Significance

There is a wide variety of web application that called "recommendation system". A subclass of information filtering system that attempts to predict the rating or preference that user responses to options and make suggestions based on these preferences [11]. As the internet has become an indispensable part of human life, we have seen the recommendation system become so popular that it has been applied in a wide variety of areas including movies, music, jokes, garment, Twitter pages. All kinds of websites or web-vendors strive their best to predict each returning users with some suggestions of products that suit their tastes.

Recommendation system use a number of different mechanisms, but overall we could classify these systems into two typical groups: "collaborative filtering" or "content-based filtering".

Collaborative filtering systems[2] focus on the relationship between users and items. Collaborative filtering systems classify users with similar interests or preference on the items, and recommend movies based on similarity measures between users. The recommendation lists are those preferred by similar users. This sort of recommendation system is based on the previous decisions made by similar customer.

In the collaborative filtering system, a key issue is that system constantly expect the good recommendation from similar users [6]. That is, given a certain user, from his/her profile, the system would able to find some number of the most similar users. But in reality every user is a distinct person with different preferences, likes and dislikes. The preference of a user might change a lot with respect of time, his or her education level, living environment or family member etc. Because one man's personality is developing through the whole life time, it is too naive to simply

sort a user into one group of similar users. The collaborative should explore new domain to profile the user in depth and refine the groups.

"Content-based filtering" is another recommendation method making inferences based on the properties of items and a profile of the user's preferences [3]. We profile both the users' interests and the item properties and recommend items based on that matching of profiles and properties. Most obviously, if a user likes a movie on YouTube, then the user can be said to like movies with similar properties in content-based filtering system.

But the issue with content-based filtering is that the system requests the user to provide a profile specifically relevant to features of items. The system has a limitation of the recommendation content. For example, the user's profile of music would be much less valuable when used for other content types of recommending of new browsing, video, products.

The programming model of MapReduce builds on the split-apply-combine strategy for data analysis. Because many information analyses perform the same basic process: a computation processing number of independent data to generate partial results, which are then aggregated in some defined standards. Strategically, MapReduce allows for distributed processing of parallelizable problems across large datasets of the map and reduction operation.

Related work

There also have been developed dozens of movie recommendation system on the web over the past few years, for example Netflix, Rotten Tomatoes, Movielens, Flixster, IMDB etc. The three web application discussed below are typical examples of a movie recommendation system.

Jinni is said as the No.1 popular movie recommendation engine of content-based filtering on the web. Not only they provide the search field of movie for you to profile the preference of yourself in detail, they deploy a method "taxonomy" to analyze metadata and reviews, take your mood, time available, reviews and other parameters into considerations, which is so called a taste-based or mood-based video discovery experience.

Criticker is another type of the movie recommendation engine of collaborative filtering. It does the "Taste Compatibility Index" on finding the movies, by comparing the rating of the movies between different users. If two users have high similarity on rating a certain number of movies of the same type, then the website will consider these two users are a matched peer on their taste of movie. Then one user's preference list of movie will be recommended for the matched user accordingly.

Netflix is another online database providing recommendation of both content based and collaborative filtering for movie [4]. It is called a hybrid recommendation system. Users are invited to give personal movie rating as many fully watched films as possible. The system will sample your tastes and it recommends similar films based on you tastes on rating to films you have seen before (i.e., content-based filtering). Meanwhile users with similar watching and searching history will share their favorite movie list (i.e., collaborative filtering).

Making a recommendation is fairly complicated because it is so difficult to tailor the results to so many different users. Thus the choice of what should be recommended is governed by lots of factors, for example age, gender, job, income, mood, availability etc. The complication of recommendation brings the importance of predicting rating accurately so high. In 2006, to boost

the research of recommendation system, Netflix started events called “CineMatch”, to find the first team that could beat its own recommendation system by 10 %. Over three year of competition, the prize of one million dollars to was finally won by a team of researchers “Bellkor’s Pramatic Chaos,” in 2009 [5].

Motivation

Content-based recommendation system is widely used in many areas nowadays, for example movie, music, book, game etc. Most of the time, the database system of movie, music or book is enormous. The tradition sequential searching or matching method often can cause a significant compromise of the demand and speed. In addition, the current rate of increase in dataset will require us to manage more data in a much shorter time. It is clear that the increasingly large dataset has outgrown the analyzing capabilities of individual computers becoming a serious issue for lots of big data researchers. To maximize the utilization of these bigger resources, and satisfy the demand for more reliability through redundancy, as well as the need for sharing of resources driven by Cloud environments, we have to think of scale-out approach, distributing the computation across multiple cores, processors or machines [7]. Hadoop’s Mapreduce framework provides an attractive and practical programming implementation for developing scalable algorithms.

Problem statement

We will try to construct a movie recommendation system of content-based filtering mechanism. The recommendation system will focus on both properties of movies and the profile of users. The open-source implementation of Apach Hapdoop MapReduce provides a distributed framework to handle large scale distributed computing across a number of node and petabytes of data [8]. It has become so popular in many applications such as machine learning, information retrieval and machine translation etc. Thus in this project, we would like to apply the ability of parallel processing of Hapdoop MapReduce into a movie recommendation with large datasets, to see if the MapReduce would fit to the increasing datasets with ease. The major contributions of this project work are:

1. To the best of our knowledge, this is the first work that shows how to construct a MapReduce framework in a movie recommendation system of content based filtering system.
2. The recommendation list of movies is based on both the movie content and the user ratings.
3. By adjusting the parameters of the evaluation, the top 15 movie are highly relevant to the user’s profile of what movie you like.
4. Two different datasets (i.e., CMU Movie Summary and MovieLens) are merged generate an accurate evaluation of the movie rating.
5. By applying the content based algorithms, we tried our best to offering suggesting movies to users by predicting the movies the user might like.

Materials and Methods

Dataset

Our project is conducted on two real world datasets both the CMU movie summary dataset and MovieLens movie rating dataset for our experiments.

The CMU movie summary dataset used in the project are the movie datasets collected by by David Bamman, Brendan O'Connor, and Noah Smith at the Language Technologies

Institute and Machine Learning Department at Carnegie Mellon University. There are three files including in this dataset:

plot_summaries.txt (Wikipedia movie ID::Movie summary).

This file consists of over 42,306 movie plot summaries extracted from Wikipedia.

movie.metadata.tsv (Wikipedia movie ID::Freebase movie ID::Movie name::Movie release date::Movie box office revenue::Movie runtime::Movie language::Movie countries::Movie genres).

This file consists of aligned metadata extracted from Freebase, including movie box office revenue, genre, release date, runtime, and language

character.metadata.tsv (Wikipedia movie ID::Freebase movie ID::Movie release date::Character name::Actor date of birth::Actor gender::Actor height::Actor ethnicity::Actor name::Actor age at movie release::Freebase character/actor map ID::Freebase character ID::Freebase actor ID)

This file consists of character names and aligned information about the actors who portray them, including gender and estimated age at the time of the movie's release.

download link:

http://www.cs.cmu.edu/~ark/personas/data/MovieSummaries.tar.gz

website:

http://www.cs.cmu.edu/~ark/personas/

movies.csv (userID::movieID::Moviename::Rating::Timestamp)

The MovieLens movie rating dataset used in the project includes all the user rating for each movie collected over various period of time. Ratings are made on a 5-star scale with half-star increments. The dataset consists of 26,000,000 ratings and 750,000 tag applications applied to 45,000 movies by 270,000 users (Last updated 8/2017).

download link:

http://files.grouplens.org/datasets/movielens/ml-latest.zip
website:
https://grouplens.org/datasets/movielens/

Method

Our goal is to calculate how similar these movies in the database are with the features of movie provided by users.

First we construct for each movie a profile by collecting important characteristics of that movie from different datasets. There are many features of movies that could be used, in this project, due to the limitation of datasets, the following properties of movies will be extracted.

The set of summary of that movies. Some users prefer describing the movie content.

The dataset of the cast for that movie. Some users prefer movie with their favorite actors.

The dataset of other general features like the genre, the time frame and the years etc. Some users prefer certain types, latest releases, or have limits on the time.

The user rating of that movie from Movielens.

As the user part, a user profile on interested of movie will be generated by user input. The user's preference for movie will be described in two ways in our recommendation system. one explicit acquisition of user information is by asking the user to describe the features of the movies he or she likes, which includes the movie name, the actor, the language, the summary, the run time, the country and the type of movie. Another way is that if the user is too lazy to input the features, he or she could simply input the name of one of his/her favorite movie.

After the user describes this terms of features or provides a movie name, the system will look into the datasets and search for the most similar movie as the user requesting.

1. Movie Score Algorithm:

In this movie recommendation system, the score of each movie is calculated by string comparison of each feature [10]. The aim of this part is to find the score of each movie relevant to the user input. The attribute used in the project to calculate the score of each movie are:

Movie summary

Movie name

Year of release

Runtime

Language

Country

Type

Actor

If the user decides to leave a blank for any feature, the score of that feature will be 0.

For movie summary part, we could apply a brute force approach, which compares the summary of movie with the summary of user input word by word [9]. Because the longer the summary, there is a higher potential that the number of matched words will be more than that of a shorter movie summary. Based on the number of matched words between the summary of movie and the user input, the grades are added accordingly following a piecewise function.

$$f(x) = \begin{cases} x, & 0 < x \leq 20 \\ 20, & 20 < x \leq 40 \\ 25, & 40 < x \leq 80 \\ 30, & x \geq 80 \end{cases}$$

where x is the number of matching words between the summary in the database and the user input.

To account the lower matching score due to the short summary of user input, we also introduce another piecewise function to scale up the grades if the user just entered a few words for the summary of the input. As long as the short of the length of a summary, the lower the probability to find the same words in the datasets. If the matched words of a movie summary reached certain

percentage of the user input and the user entered a summary less than 40 words, the following function will be used.

$$f(r) = \begin{cases} 4, & r \geq 0.05 \\ 6, & r \geq 0.1 \\ 8, & r \geq 0.25 \\ 10, & r \geq 0.5 \end{cases}$$

where $r = \frac{(summary\ of\ the\ user\ input) \cap (summary\ of\ the\ movie)}{(summary\ of\ the\ user\ input)}$, the ratio of the matched words over the total number of words of user input.

For the part of movie name, quite similar to the summary part, we could apply a brute force approach which compares the movie name in the datasets with the movie name by the user input word by word. Because the longer the name, there is a higher potential that the number of matched words will be more than that of a shorter movie name. Based on the number of matched words between the movie name in the database and the user input, the grades are added accordingly following a piecewise function.

$$f(r) = \begin{cases} 3, & r > 0.1 \\ 5, & r \geq 0.25 \\ 10, & r \geq 0.5 \\ 15, & r \geq 0.75 \end{cases}$$

where $r = \frac{(movie\ name\ of\ the\ user\ input) \cap (the\ movie\ name\ in\ the\ dataset)}{(movie\ name\ of\ the\ user\ input)}$, the ratio of the matched words of movie names over the length of the movie name of user input.

For the part of years of release, we tried to differentiate the movies by the year of release. The closer to the year designated by the user as input, the higher the score of the movie.

$$f(x) = \begin{cases} 5, & 0 < |x - y| \leq 2 \\ 3, & 2 < |x - y| \leq 5 \\ 1, & 5 < |x - y| \leq 8 \end{cases}$$

where x is the year of the year of movie release, y is the year defined by user input.

Besides the equation above , if the user entered a year before 1980, about half a century ahead of current time, we will assume the user really like old movie. For all the movies that are older than 1980, we add 3 points to the total score for that movie.

$$f(x) = 3 \qquad if\ x < 1980$$

For the run time part, we set the standard runtime for each movie is 90 minutes. If a user wants to watched a long movie, any movie longer than 90 minutes will score 5 points. Else if a user prefers a short movie, we assume that the user want to watch a movie that is no longer than 90 minutes.

$$f(x) = \begin{cases} 5, & x > 90, \quad if\ a\ user\ asks\ long\ running\ time\ for\ a\ movie \\ 5, & x < 90, \quad if\ a\ user\ asks\ short\ runnint\ time\ for\ a\ movie \end{cases}$$

For the language part, if a movie have been translated into a language with the same language as the user requesting in the input, we will add 8 point to that score of the movie.

For the country, if a movie produced in a country as the same country as the user requesting in the input, we add 5 point to the score of movie. Since the language always come with the country, so we lower the grades to 5 for country to balance the grades coming from the language.

For the genre of the movie, if the types of the movie matched with the type of user input, 5 points will be added to the score of that movie.

Last part the actor, we want to discriminate between actors of the main cast and actors of walk-on. If the user enter the name of an actor who is listed as the first or second in the cast, we will count that actor plays a lot in that movie. For actors plays key roles, we add 8 point to that movie. If the actor list as the top 4 in the cast, we add 5 point to that movie. Else if the name just showed in the cast, we add 3 point to that movie.

$$f(x) = \begin{cases} 8, & x \leq 2 \\ 5, & x \leq 4 \\ 3, & x > 4 \end{cases}$$

where x is the place of the actor in the cast of that movie.

2. Implment: How to use Mapreduce to recommend a movie in parallel.

We have four MapReduce applications and 3 java applications in total. In one movie recommendation, a MapReduce application or java application may be called more than one time. So the whole system is a little bit complex. In this section, I will introduce my system from simple case to complex case. The simplest case is the movie recommendation based on only wiki data set and feature description about what the user like. It only uses 2 times MapReduces. The most complex case is the movie recommendation based on wiki and movielen data set. The input of user is one movie name. In this case, we uses 5 times Mapreduce and 3 Java applications. Notice that, in the implement, we combine the 3 java applications into a big java application. We use –s 1, -s 2 and –s 3 to choose the function of the big java application.

(1) Case 1: User Input: description. Data Set: wiki data set.

In this case, the user input is the description about what his favorite movie is. For example, a people would like to watch movies which involves some animals as “dog”,”cat” or”elephant”. He also likes the actor “peter finch” and “simon yam”. Some old movies as 1944 movie is his favorite. And he likes thriller and gangster film. He can also input some language, country and runtime to describe his preference. Then the input document format of the case 1 is as below:

Input document format:

Input document format:
name: nothing
summary: dog cat or elephant.
actor: waise lee; simon yam
year: 1990
type: thriller;gangster film
language:english;French
country:hong kong
runtime:long

Our system will find 15 movies which has largest scores to recommend to users.

As discussed above, the wiki dataset contains movie contents information. But the information is not stored in a single file. The file character.metadata.tsv contains the actor information. The file movie.metadata.tsv contains the year, type, language, country, runtime information. The plot_summaries.txt contains the movie summary information. In three files, the wiki movie id is the key attribute. The same wiki movie id in three files points to the same movie as the red underline in the below example.

plot_summaries.txt

```
15585766    Three friends are droving cattle in Australia in 1939: the restless Bluey Donkin, easy-going Milo Trent and Englis
1760737 Set in a lonely city on a rainy night, the film takes place in a bicycle shop  that is closed for the night. In the co
29062594    A group of teenagers at Catholic boarding school in the provinces spend their time trying to adapt to the demands
```

character.metadata.tsv

```
7156316 /m/0h776p   Who Done It?    1942        76.0    {"/m/02h40lc": "English Language"}  {"/m/09c7w0": "United States of Amer
15585766    /m/03mglpl  The Rats of Tobruk  1944        68.0    {"/m/02h40lc": "English Language"}  {"/m/0chghy": "Australia"}
4363038 /m/0bz8c2   Rusted Pieces   1991        42.0    {"/m/02h40lc": "English Language"}  {"/m/09c7w0": "United States of Amer
```

plot_summaries.txt

```
8893822 /m/027nmrt  1983            M           Ray Baxter      /m/0n44h9p      /m/0n44h9s
15585766    /m/03mglpl  1944        1916-09-28  M           Peter Finch 27  /m/040nz_t      /m/016ynj
15585766    /m/03mglpl  1944        1909-03-26  M           Chips Rafferty  34  /m/040nz_z      /m/07kzcy
15585766    /m/03mglpl  1944        1917-12-06  M           Grant Taylor    26  /m/040n_03      /m/08pk7b
9776644 /m/02prw2f  2008-05-29  Kristen McKay   1977-07-01  F   1.78    /m/022dp5   Liv Tyler   30  /m/02vctjh  /m/04hdt78  /m/01rh0w
```

Firstly, we need to compute scores of every movie in the data set using the movie score algorithm. But in the data set, there is no one file contains all the information which can used to score the movies. We need to grade movies on different files using the movie score algorithm.

We use a MapReduce to grade movies on different files in parallel. The figure 1 shows the MapReduce I:wiki_Description_to_Score in the project. In the Map process, the output key is the wikiID of movie and the value is the score of file matching. Using this MapReduce, the user description can be matched on different features of a movie in different files of the data set. The score of the matching can be computed by the movie score algorithm which is discussed in the above section. As in the figure 1, the movie score algorithm can match the user description documents with movie.metadata.tsv to get the scores of year, type, language and runtime matching of every movie's wiki ID. The output of mapping of the movie.metadata.tsv file is <wiki ID, score of file matching >. The score of file matching =year scores+type scores+language scores+runtime score. Similarly the scores of summary can be computed by matching user description with plot_summaries.txt. The output of mapping of the plot_summaries.txt file is <wiki ID, score of file matching> . The score of file matching= summary scores. And the scores of acter can be computed with the file character.metadata.tsv.

In the reduce part of the MapReduce , the output key is the wikiID of movie and value is the score of matching. In the reduce part, all of scores from different files with same wikiID will be added with each other as score of matching=scores of movie.metadata.tsv +scores of plot_summaries.txt +scores of character.metadata.tsv.

After the MapReduce I, we sort the wiki ID based on its scores of matching. The sort is implemented as the linux statement as "sort –n –r –k2 part-r-00000 wikiscore". After the sorting, the 15 largest scores with related 15 WikiIDs will be picked up. Obviously, users will not satisfy the wiki ID we provided. They need the content of movie features behind wiki IDs. Then we have the MapReduce II to extract the movie content of features with the 15 wiki IDs. The features are movie name, summary, actor, runtime, type, language and so on.

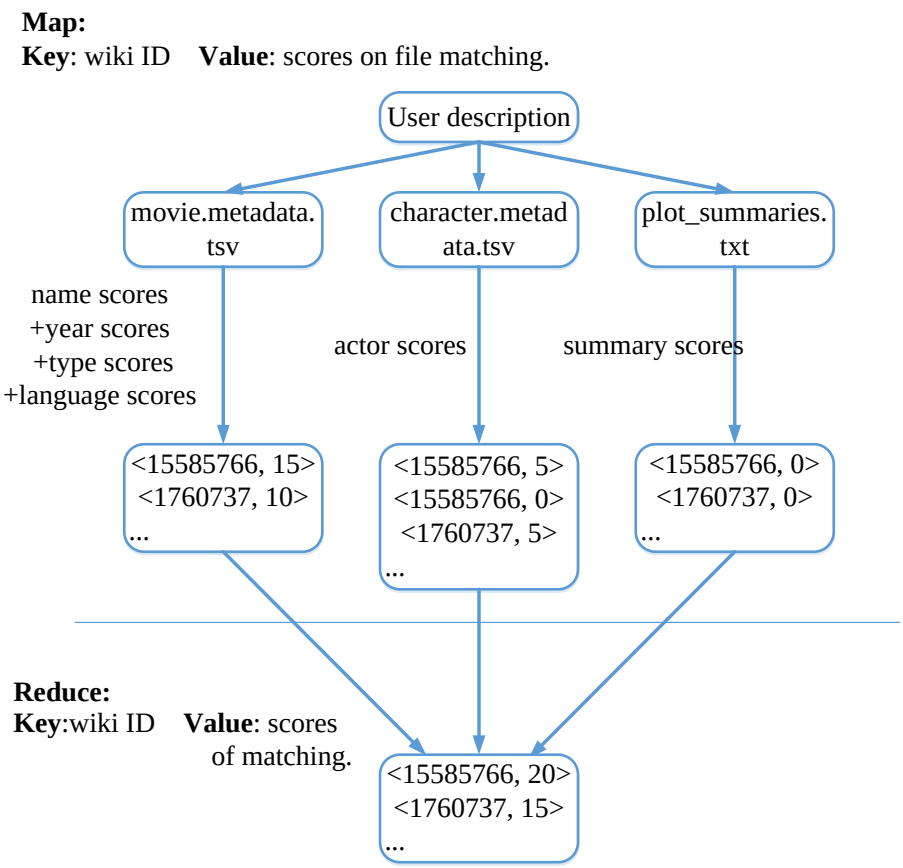

Figure 1, MapReduce I: wiki_Description_to_Score

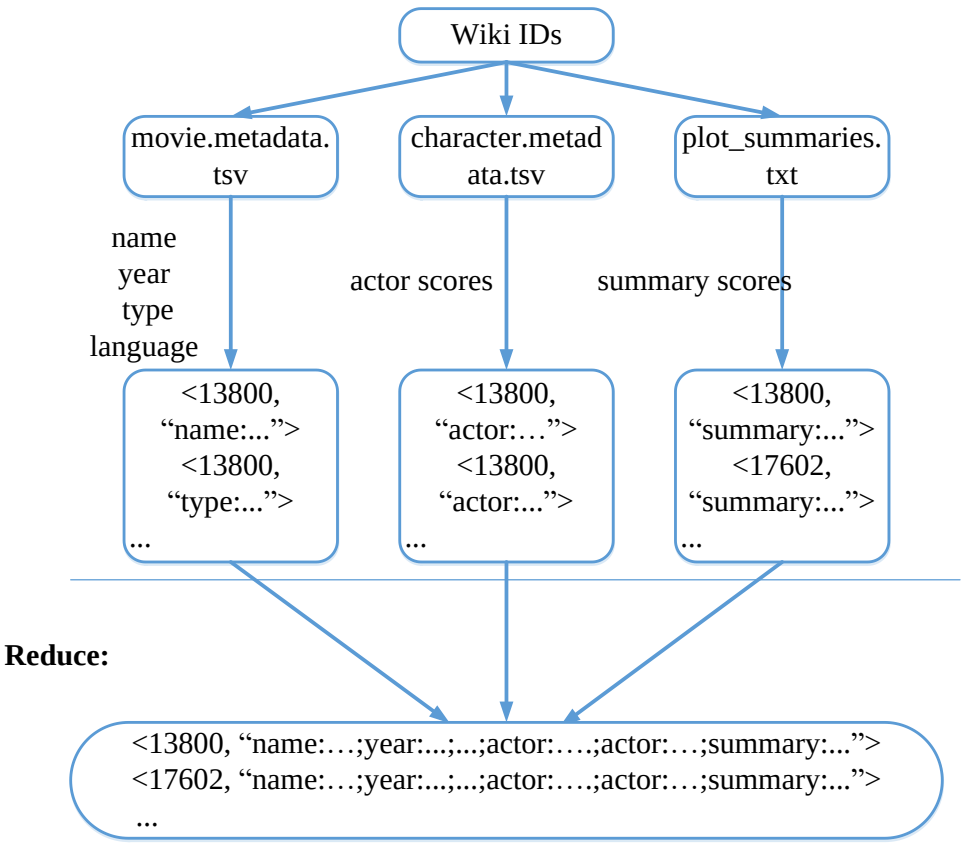

Figure 2, MapReduce II: ID_to_content

We use the MapReduce II: ID_to_content as the figure 2. The input is the top 15 wiki IDs. In the Map process output, the key is wiki ID and value is content of feature. In the map process, we use the Wiki IDs to extract the content of these features from every file. The feature name should be recorded in the output value of map. For example, the wikiID 13800 is in the top 15 wikiIDs. Using wikID=13800, we can extract its year=1990 in the movie.metadata.tsv and output <13800,"year:1990">. We also can extract its type in the movie.metadata.tsv and output <13800,"type:science ficition">. Similarly, the character.metadata.tsv can provide the content of feature "actor" and the plot_summaries.txt can provide the content of feature "summary". Notice that, as in the figure 2, one wikiID may output multi actors in different outputs of the map. It is because the format of character.metadata.tsv is one wikiID with just one actor.

In the reduce part, content of different features of movie with the same wikiID will combine with each other into a single string. As in the figure 2, the wiki ID=13800 may get data from map as <13800:"name:…">,<13800:"year:…">,<13800:"actor:..">. After the combination, the output of reduce will be<13800, "name:…; year:...; ...; actor:….; actor:…; summary:...">.

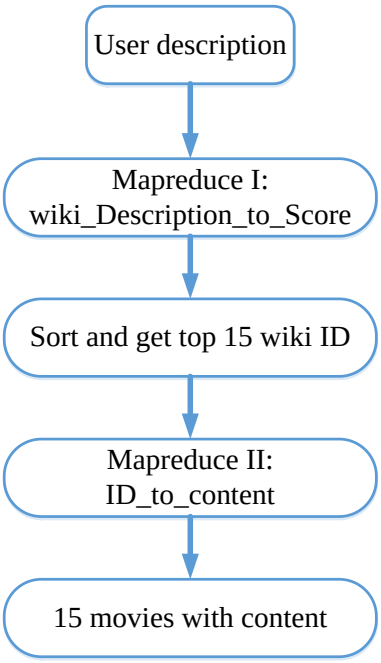

Figure 3 the flow chart of the case I.

After all of above steps are done, the 15 movies are recommended to users with contents of features. The flow chart of the case I is as Figure 3. The case is easiest case of this project, it only use two MapReduces.

(2) Case 2: User Input: movie. Data Set: wiki data set.

In this case, users input only his favorite movie name. Our movie recommendation system can extract content of features of his favorite movie from the wiki data set. The extracted content of movie features are transformed to the input format of case 1. Then we can use the case 1 to recommend 15 movies. We discuss all steps in detail in this subsection.

As we have discussed, the movie information is stored in the wiki data set by unique WikiID. In order to extract the movie information easily, we need to transfer the input movie name to the wiki ID. We use a Java application to transfer it. For example, if user input “Rusted Pieces” as his favorite movie name, this Java application will output 4363038 as the related WikiID based on the file character.metadata.tsv.

character.metadata.tsv

```
7156316 /m/0h776p   Who Done It?    1942        76.0    {"/m/02h40lc": "English Language"}  {"/m/09c7w0": "United States of Amer
15585766    /m/03mglpl  The Rats of Tobruk  1944        68.0    {"/m/02h40lc": "English Language"}  {"/m/0chghy": "Australia"}
4363038 /m/0bz8c2   Rusted Pieces   1991        42.0    {"/m/02h40lc": "English Language"}  {"/m/09c7w0": "United States of Amer
```

With the WikiID, we use the MapReduce II to extract the information of the content of movie features . The Map Reduce II has been discussed in the case 1. In the case1, the Map Reduce II is used to provide the content information of the top 15 recommended movies at the final step. At here, it is used to process data in the middle step of case 2. It is one of our framework’s good characteristics. Some of functions have multiple usages. It will make the whole framework to easily maintained and understood.

The input of the Map Reduce II is the wikiID and output is the features' content of the movie with wikiID. If we input Wiki ID=4363038 into the MapReduce II, the output should be <4363038, "name: Rusted Pieces; year:1991; language:"English Language"; country:"United States of America"; actor:…; summary:...">.

In the case 1, the input document is the user description. At here, the result of Map Reduce II's is enough to fill this document. In the above example, we can fill input document of the case 1 as the below figure.

Input document format:
name: Rusted Pieces
summary: nothing
actor: nothing
year: 1991
type: nothing
language: English Language
country: United States of America
runtime: Short

If the result does not have contents of some features, the feature in the document will be filled as "nothing". As the figure, there are "summary: nothing" and "type: nothing". All of these filling are done by a Java application.

By now, we get an input document of case 1. Then we can use the case 1 to recommend 15 movies to users with the detailed content information of movies. This is a good characteristic of our framework. The whole case can be used by another case. It is good for the software maintain and easy to understand. The flow chart of the case 2 is as Figure 4. In the case 2, there are 3 MapReduce and 2 Java applications have been used.

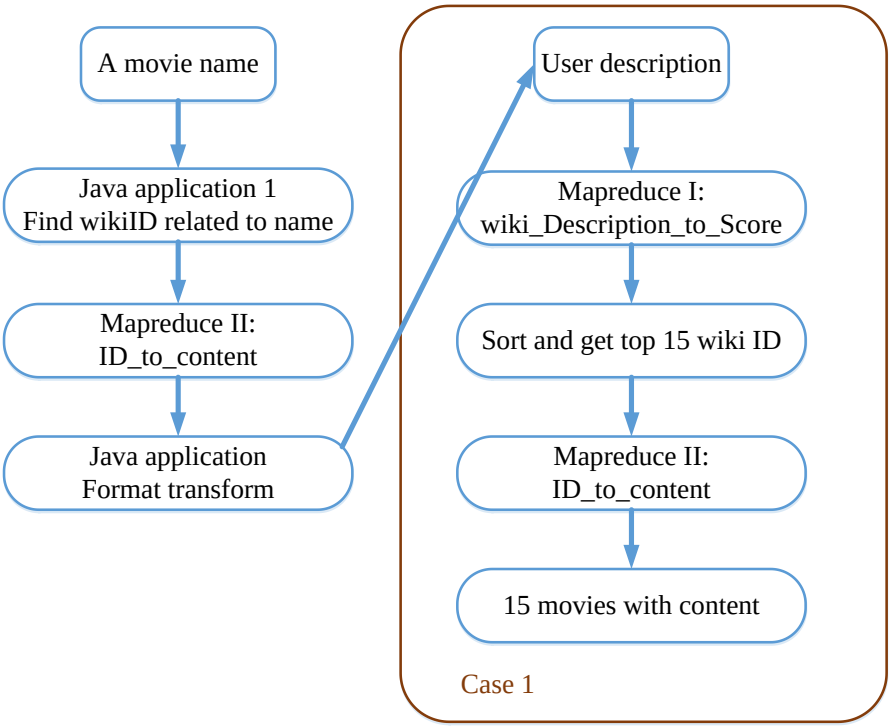

Figure 4. the flow chart of the case 2.

(3) Case 3: User Input: description. Data Set: wiki data set and movieLen data set.

All discussed above are only for a single data set as wiki data set. In the wiki data set, the movie contents are stored. In the case 3, we not only consider about the content match but also consider the rating of movies. Every people likes movies with good rating. In the movieLen data set, people are invited to rate movies. One people may rate more than one movies. We use two files in the movielen data set. The movies.csv records the movielen ID and movie name. And the rating.csv records the rating for every user on movies. Two files are linked by the movielen ID. The below figures are example of the movies.csv and rating.csv.

movies.csv

```
movieId,title,genres
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Romance
4,Waiting to Exhale (1995),Comedy|Drama|Romance
5,Father of the Bride Part II (1995),Comedy
6,Heat (1995),Action|Crime|Thriller
```

rating.csv

```
userId,movieId,rating,timestamp
1,110,1.0,1425941529
1,147,4.5,1425942435
1,858,5.0,1425941523
1,1221,5.0,1425941546
1,1246,5.0,1425941556
1,1968,4.0,1425942148
1,2762,4.5,1425941300
1,2918,5.0,1425941593
```

As shown in the file rating.csv, one user may be invited to rate more than one movies. So we need to use MapReduceIII to compute the average rating for every movielen ID from all of users.

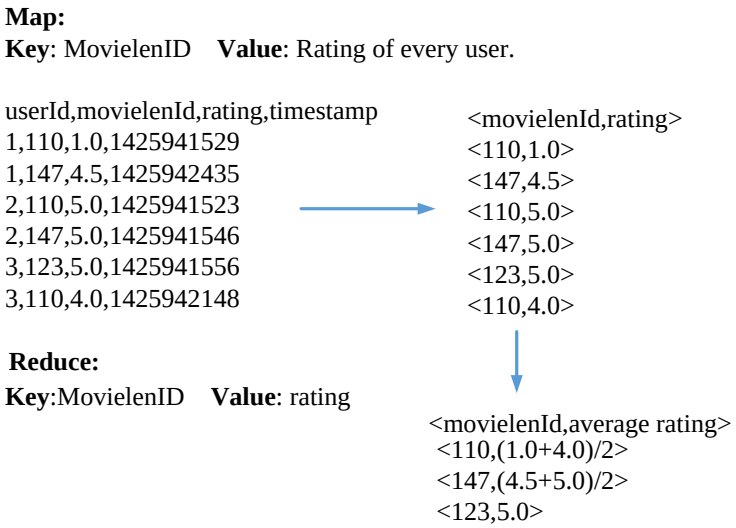

Figure 5. MapReduce III MovieLenID_Rating

As in the figure 5, the MapReduce III: MovieLenID_Rating is used to compute the average rating of every movielen ID. Map output is <MovielenID, Rating of every user>. The Reduce output is<MovielenID, average rating>. In every entry of the rating.csv, the map picks up the movielenId and rating. In the reduce step, ratings of one movielenId will be computed as average.

We need to consider both of content scores in the wiki data set and the user rating in the movielen data set to recommend movies. But the same wikiId and movielen Id may point to different movies in two data set. We need to link wikiId and movielenId in two data set based on the same movie name. We use MapReduce IV: WikiID_MovieLenID to link them.

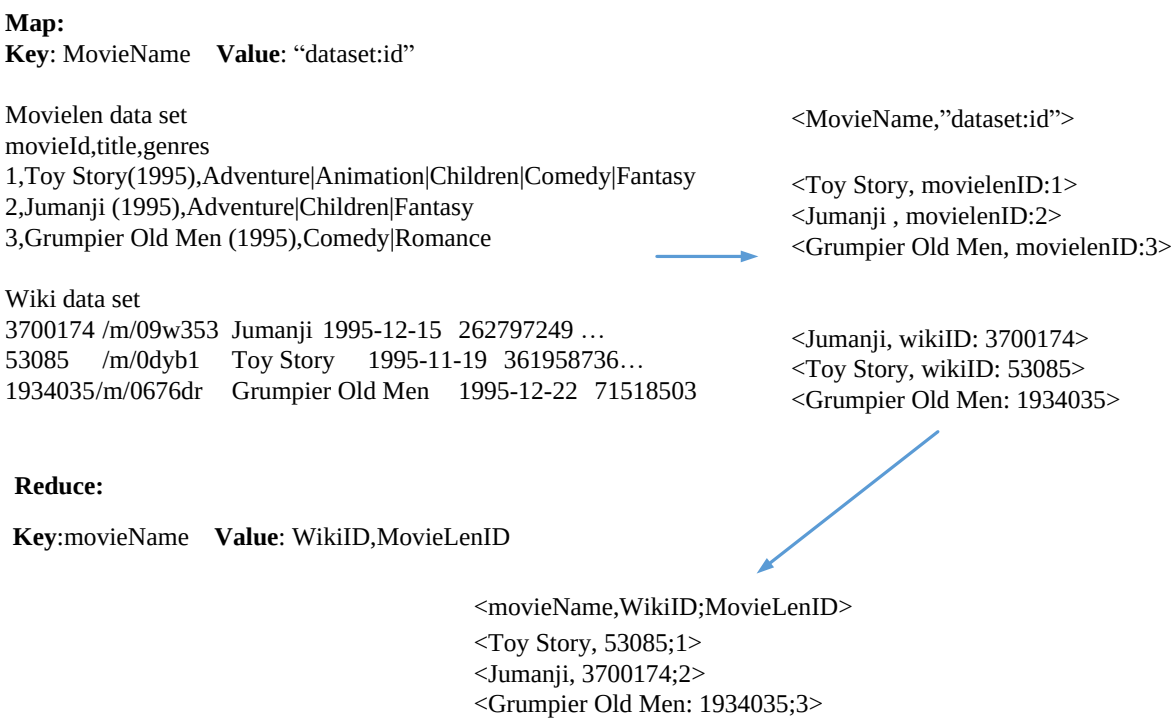

Figure 6. MapReduce IV WikiID_MovieLenID

The Mapreduce IV: WikiID_MovieLenID is shown as the figure 6. The key of the map is the "movieName", because we use movieName to link wikiID and movielenID in two data sets. The value of the map is the "dataset:id". For example, an entry of the moivelen data set is "1. Toy Story (1995), Adventure….". The map output for this entry is<Toy Story, movielenID:1>. There is an entry in wiki data set as "53085 /m/0dyb1 Toy Story 1995-11-19 361958736…". The map output for tis entry is <Toy Story, wikiID: 53085>.

In the reduce phase, a movie name can collect a wiki ID and MovieLenID. Then the wikiID and MovieLenID are pointing to a same movie. The key of the reduce is the "movie Name" and value is "WikiID;MovieLenID". In the above example, the output of the reduce should be <Toy Story, 53085;1>. It means the wikiID 53085 and movieLenID 1 are pointing to the movie "Toy Story".

Now we have two files < MovieLenID, rating> and <movie Name, WikiID;MovieLenID >. To consider both content scores and rating, we need to get the content scores information. By the first part of the case 1, with the user description document and wiki data set, the MapReduce I: wiki_Description_to_Score is used to compute the <WikiID, scores>.

Now we have three files. We need to compute the <WikiID, finalscores>. The finalscores=scores+rating. We need to know the rating that a WikiID is related to, but the rating is with MovieLenID. The solution is finding the MovieLenID in the < movie Name, WikiID;MovieLenID > with a given WikiID and using the found MovieLenID to get the rating information in the<MovieLenID, rating>.

We assume there are N movies in three files. Every WikiID needs to iterate the file <movie Name, WikiID;MovieLenID > with O(N) to find the related MovieLenID. The found MovieLenID needs to iterate the file < MovieLenID, rating > with O(N) to find the related rating. Because there are N wikiIDs, the total computing complex is O(N3).

To reduce the computing time, a O(N) implement has been designed. We extract the <WikiID;MovieLenID> from the file < movie Name, WikiID;MovieLenID >. Then the <WikiID;MovieLenID> are stored into a hash table H1 with O(N). We also store <MovieLenID, rating> in another hash table H2 with O(N). Then every WikiID can find the related movieLenID by H1 with O(1) and every movieLenID can find the related rating with O(1). There are N wikiIDs. So the total computing complexity is O(N3). All of above steps are implemented by a java application.

Now we have the file < WikiID, finalscores >, the left steps of case 1 can be used to recommend 15 moives to users. There are three MapReduce and an application are used in case 3. The chart of the case 3 is as Figure 7. Except the red blocks, all left blocks are same to the case 1.

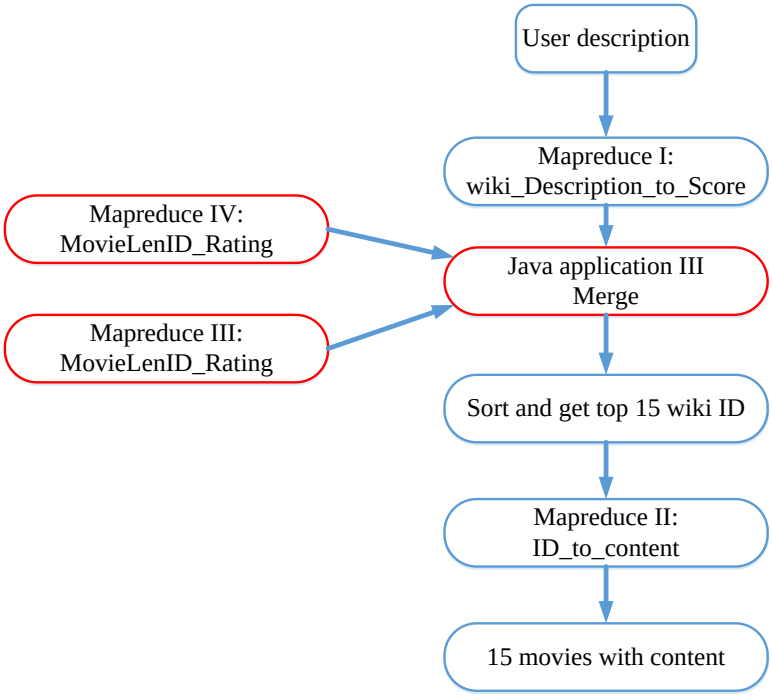

Figure 7 flow char of case 3

(3) Case 4: User Input: movie name. Data Set: wiki data set and movieLen data set.

The implement of case 4 is the combination of the case 2 and case 3. For the case 3, the case 4 adds the part of transform movie name to description input of the case 2. For the case 2, the case

4 adds the part of merge 2 different data sets of the case 3. As in the figure 8, the case 4 uses 5 Mapreduce and 3 java applications.

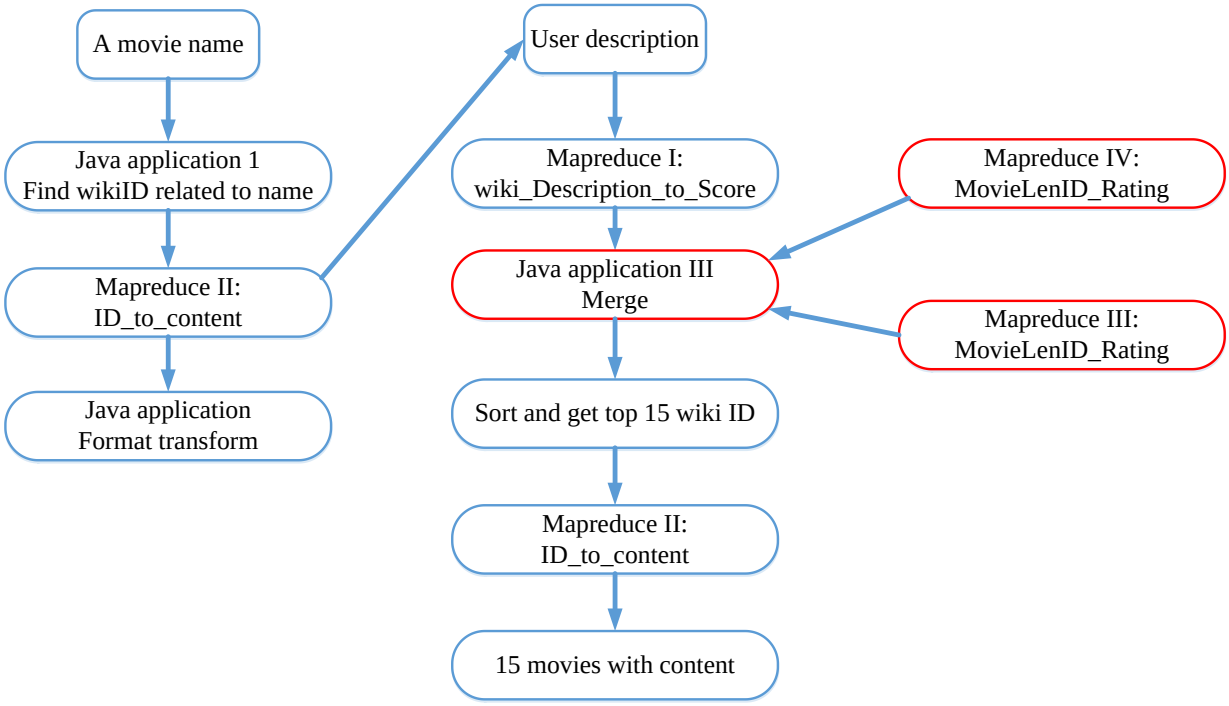

Figure 8 flow chart of case 4

Advantages and innovations of our method:

At each recommendation, our system is reflecting the latest results. There are many new movies released each day, and users can't watch and review all of those. Content based filtering doesn't depend on the user reviews and user similarity. Some rating system could be abused by the Ballot Stuffing.

Each recommendation will go through the all the different datasets. If there are any changes of the datasets affecting the results, the system will find it. The results are much comprehensive than the collaborative filtering.

Very time efficiency. Introduce Mapreduce in combination of Hashmap into the recommendation system to construct the properties of a movie and find out the similarities between each search.

Results (Finding resulted from your methods, Evaluation)

Evaluation Metrics

There are several types of measures for evaluating the success of a recommender system. To check how well the performance of our movie recommender system meets the needs of the user, we deployed a metric named "mean average precision" (MAP) to measure the quality of recommendation [1]. MAP is designed for evaluation of the average percentage of correct items

among a ranked list. MAP is widely used in the information retrieval (IR) to evaluate the top-N recommendation. The MAP is learned in the BMI 733 classes.

First we calculate the precision, P of MAP. Precision is the faction of the relevant parts to the recommended items (eqs 1).

$$precision = \frac{|\{relevant\ movies\} \cap \{recommended\ movies\}|}{|\{recommended\ movies\}|} \qquad (eqs\ 1)$$

Then we calculate the AP. Specifically for a movie recommendation, Average precision (AveP) is the average of the precision value obtained for the top k movies, each time a relevant movie is recommended (eqs 2).

$$AveP = \frac{\sum_{k=1}^{n} P(k)}{number\ of\ relevant\ movies} \qquad (eqs\ 2)$$

Last, mean average precision (MAP) for a set of queries is the arithmetic average of the average precision (AP) for each query (eqs3).

$$MAP = \frac{\sum_{q=1}^{Q} AveP(q)}{Q} \qquad (eqs\ 3)$$

where Q is the total number of queries.

To evaluate the results of our movie recommendation system, we would try to suggest movies for the clients based on their input. We start by simulating some of the user input (i.e., movie properties or movie name) and see if the movies offered by the system are closely relevant to that of user input.

There are two kinds of user input according to the method parts. One is the user defined movie features. The other is the movie name given by the users.

Let's start the evaluation with input of user defined movie features including movie name, summary, actor, year, type, language, country and runtime.

Case 1a: If a user is a big fan of a specific movie, say Jurassic Park.

movie name: Jurassic Park

summary: nothing

actor: nothing

year: nothing

type: nothing

language: nothing

country: nothing

runtime: nothing

Here “nothing” means that the value is null. Because in the datasets some value is null, to prevent the false equal between null of user input and the null of datasets, we set the default null value of users’ input as “nothing” for discrimination.

The output of top 15 by our movie recommendation system is:

4129450	beyond jurassic park
1431847	jurassic park iv
3575300	punishment park
3818512	15, park avenue
473273	jurassic park iii
5051761	park
6143174	mansfield park
6640035	jogger's park
666923	the lost world: jurassic park
68485	jurassic park
7093500	sunset park
761361	dog park
7840287	gandhi park
8138535	barefoot in the park
9916192	luna park

The output file is named as “finalOutput1”. Basically, all the series of Jurissic Park are being selected in the list. Even better all of the Jurassic park are found ranking in the top 10 of the list actually. The other movies are not strictly related with Jurassic park. But since we are searching the data sets by movie name, the others in the list are namely close to Jurassic park somehow.

If we assume that the movie named “*park” are the relevant movie to Jurassic park, the average precision (AP) of the recommendation list would be 1 (100%). If we only consider Jurassic park and its series as the most relevant movie, the precision for the each itme would be 1/1, 2/2, 2/3, 2/4, 3/5, 3/6, 3/7, 3/8, 4/9, 5/10, the average precision (AP) of the list would be (1/1 + 2/2 + 2/3 + 2/4 + 3/5 + 3/6 + 3/7 + 3/8 + 4/9 + 5/10)/10 = 0.6015 (60.15%).

Case 1b: If a user is a big fan of both a specific movie, say Jurassic Park, and an actor named Waise Lee.

movie name: Jurassic Park

summary: nothing

actor: Waise Lee

year: nothing

type: nothing

language: nothing

country: nothing

runtime: nothing

The output of top 15 by our movie recommendation system is:

14129450	beyond jurassic park	
1431847	jurassic park iv	
22607311	overheard	waise lee
26876540	the east is red	waise lee
3105539	wing chun	waise lee
27641142	inspector pink dragon	waise lee
3193452	tricky brains	waise lee
3473999	swordsman ii	waise lee
473273	jurassic park iii	
5051761	park	
6426885	running out of time	waise lee
666923	the lost world: jurassic park	
68485	jurassic park	
89541	a better tomorrow	waise lee
9152153	inner senses	waise lee

The output file is named as “finalOutput2”. As the user requesting, the recommendation list consists both the movies acted by “Waise Lee” and movies namely relevant to “Jurassic Park”. If we assume that the movie named “*park” are the relevant movie to Jurassic park, the average precision (AP) of the recommendation list would be 1 (100%) when the movies are featured by both the actor “Waise Lee” and “Jurassic Park”.

Case 1c: If a user is a big fan of both a specific movie, say Jurassic Park, and an actor named Waise Lee and meanwhile he or she really likes movie about fairy tale, snow queen, price and princess etc. On top of case 1b, this time we add another feature of summary of movies.

movie name: Jurassic Park

summary: fairy tale “the snow queen”, a fareless prince, a beautiful princess, a journey, a rugged iceman, pet reindeer, a naïve snowman, ice power, kingdom in winter

actor: Waise Lee

year: nothing

type: nothing

language: nothing

country: nothing

runtime: nothing

The output of top 15 by our movie recommendation system is:

14129450	beyond jurassic park	
1431847	jurassic park iv	
2287636	sleeping beauty	
33641119	mirror mirror	
473273	jurassic park iii	
4832313	happily ever after	
5051761	park	
525314	shrek 2	
666923	the lost world: jurassic park	
68485	jurassic park	
76361	snow white and the seven dwarfs	
8055250	snow white	
8785217	snow queen	
8833565	the snow queen	1957
903383	shrek the third	

The output file is named as "finalOutput_3". According to the user input, besides the movie name and actor, he or she wants to watch some movie related quen, prince, winter, and fairy tale. Thus the recommendation list represent some movies related with that kind summary, like sleeping beauty, mirror mirror, happily ever after etc. For example, movie "happily ever after" film begins with a recap of what happened in Snow White and the Seven Dwarfs provided by the Looking Glass.

By checking the summary of the movies and the relevant movie to Jurassic park, the average precision (AP) of the recommendation list would still be 1 (100%) when the movie is described by the user defined summary, the actor "Waise Lee" and "Jurassic Park".

Case 1d: If a user is a big fan of a specific movie, say Jurassic Park, and an actor named Waise Lee. And he or she also prefer the movie produced in hong kong and some movie about fairy tale, snow queen, price and princess etc.

movie name: Jurassic Park

summary: fairy tale "the snow queen", a fareless prince, a beautiful princess, a journey, a rugged iceman, pet reindeer, a naïve snowman, ice power, kingdom in winter

actor: Waise Lee

year: nothing

type: nothing

language: nothing

country: hong kong

runtime: nothing

The output of top 15 by our movie recommendation system is:

1431847	jurassic park iv	
2287636	sleeping beauty	
27975847	all's well, ends well	
30574080	the sorcerer and the white snake	
3299612	twins effect ii	
33641119	mirror mirror	
3623596	the grand substitution	
473273	jurassic park iii	
5051761	park	
6309664	a chinese tall story	
666923	the lost world: jurassic park	
68485	jurassic park	
8055250	snow white	
8833565	the snow queen	1957
903383	shrek the third	

The output file is named as “finalOutput_4”. On top of the results of case 1c, this list also includes the movies relevant to hong kong for example “all’s well, ends well”, “the sorcerer and the white snake” and “twins effect ii” etc. By checking the summary of the movies, the country, the actor and the relevant movie to Jurassic park, the average precision (AP) of the recommendation list would still be 1 (100%).

The same kind of user input (featured description) and datasets (CMU movie summary datasets) are used in the set of query from case 1 to case 4, so we could assume that the query of case 1a through 1d is a set of query whose AP could be used to calculate the MAP.

$$MAP = \frac{1 + 1 + 1 + 1}{4} = 1$$

The value of MAP indicates that the results for the set of query 1a-1d is 100% relevant to the user input.

By the nature of our system, it is no easy way to evaluate the performance since the judgment of recommendations varying from person to person. For a practical evaluation of user experience, we asked a user to rate the list of recommended movies by labeling like or dislike to a specific movie

in the list under that a specific search of movie (i.e., case1a-1d). From the user's perspective, an average 77.52 % of the recommendation list would meet their expectations. Table 1 shows the comparison of the average precision between our developer and the user experience for each case specifically.

Table 1. APrecision for each case

	Developer	User experience
Case 1a	1	0.6015
Case 1b	1	1
Case 1c	1	0.8849
Case 1d	1	0.6140
MAP(Mean Average Precision)	1	0.7751

Case 1e: If a user is a big fan of a specific movie, say Jurassic Park, and an actor named Waise Lee. And he or she also prefer the movie produced in hong kong and some movie about fairy tale, snow queen, price and princess etc.

The condition defined by the user is the same as case 1d. The only difference is that we add the user rating from the datasets of movielens to the score of each movie at this time.

The output of top 15 by our movie recommendation system is:

27975847 all's well, ends well

30574080 the sorcerer and the white snake

3299612 twins effect ii

33641119 mirror mirror

3473999 swordsman ii waise lee

3623596 the grand substitution

473273 jurassic park iii

5051761 park

525314 shrek ii

6309664 a chinese tall story

666923 the lost world: jurassic park

68485 jurassic park

76361 snow white and the seven dwarfs

8833565 the snow queen 1957

903383 shrek the third

The output file is named as “finalOutput_7”. Different from the results of case1d, since user rating are accounted, some movies with lower user ratings are removed from the list for example “Jurassic park iv” which is the top 1 of the list in case 1d. Also the list satisfies all the conditions provided by users, including the movies relevant to the actor “Waise Lee” which is not seen in result of case 1d. And by checking the summary of the movies, the country, the actor and the relevant movie to Jurassic park, the average precision (AP) of the recommendation list would still be 1 (100%).

Next we are switched to another kind of user input. When user gets lazy, instead of defining the features of favorite movie, they would simply input his/her favorite movie for example “snow white”. As we described in the method part, the recommendation system will assume that the users satisfied with all the features of that movie. By default, the feature of that specific movie will be used for the query.

Case 2a: If “snow white” is the favorite movie of a user

The output of top 15 by our movie recommendation system is:

12503466 snow white 1902

18445420 happily n'ever after 2

19284991 snow 2008

21414543 snow white and the seven dwarfs 1961

23047923 snow 1981

26244289 snow white and the three stooges

28203984 snow 1963

2826225 snow white 1916

29168368 white as snow

36019569 snow white and the huntsman

4187434 snow white Christmas

76357 snow white 1933

76361 snow white and the seven dwarfs 1937

8055250 snow white 2001

9644100 snow white 1987

The output file is named as “finalOutput_5”. As the user said “snow white” as his or her favorite movie, the movie recommendation system would first extract the features specifically related to “snow white”, for example type of animation or fantasy, country of united states of America, summary related with queen, dwarfs, snow etc. Based on the features of the favorite movie, the system would recommend the movie. Thus in the list above, intuitively, you can tell all the movies are talking about snow white.

Case 2b: If “snow white” still is the favorite movie of a user and the user rating of movielens was taken into consideration.

The output of top 15 by our movie recommendation system is:

11275640	flower drum song
11445373	white heat
12169608	ben 10: secret of the omnitrix
12503466	snow white 1902
14635811	snow 2004
18445420	happily n'ever after 2
2600572	3 men and a little lady
26244289	snow white and the three stooges
29168368	white as snow
341538	the black hole
36019569	snow white and the huntsman
3863417	beer league
4187434	snow white Christmas
540654	body heat
76361	snow white and the seven dwarfs

Some movie not that content relevant to the snow white will be added to the recommendation list after counting the user rating into the score. There are about half of list which are the movies related with the content of snow white, and also some varieties compared with the results of case2a.

Discussion (results, limitations, Future Work)

This project has attempted to identify the movies of interest to users based on the description of features or their favorite movies. The similarity between each different movie are based on the similarity of between the input of user and features of movies in the datasets.

In case 1, when the user is asked to give a description of the preferred feature, we observe that in case 1a, by simply enter a movie name, and leave other features blank. The system will output all the movies which is namely related to the name of the movie. From case1b to case 1e, it is found that as the conditions are added, like the actor, summary, and country, the results could fulfill the requirements of the user. Moreover, to test our system, when we simulate the feature, we tried to make the thing worse. Instead of make a input of features pointing to the same movie, we make the movie name, actor, country, summary opposite to each other. Even if the movie name, the actor, the summary and the county are not related with each other, the top 15 movies from our recommendation list still covered all the aspects of input. This suggested that the contented based filtering algorithm are very useful in providing a recommendation list.

In case 2, when the user is asked to provide one of his or her favorite movies as an input, we observed that in case 2a, that list not only containing the movies that their names are close to the

input movie, the content is also quite close too. Because the system will extract the feature (i.e., actor, genre, name, country, summary etc.) of that input movie, and the evaluation will go through the whole feature comprehensively. The top 15 movie are highly relevant to the user's favorite movies.

In case of the user rating, we observe that the recommended movie list changed significantly after the movie rating being added. By comparing 1d with 1e and 2a with 2b, the rank of each movie changed a lot. Some movie at lower rank at content filtering, but with high user rating in the movielens dataset, get moved forward. Some movie at higher rank at content filtering, might go downgrade if the user rating is bad. Thus the datasets of movie content and user rating are merged successful in our case. And both content datasets and user rating datasets are effective in ranking of movies in the recommendation system.

As for the precision of the movie recommendation system, there is a comparison of the effectiveness of the recommendation in the perspective between developer and user. We observe that in Table 1, precision of top15 movie list based on a real-life user experience is less than the developer. The MAP is 100 % from the develop side, 77 % from side of the real life user. This suggests that our recommendation system did offer suggesting movies to users by predicting the movies the user might like.

Overall, the results are encouraging for the use of Mapreduce in content-based filtering recommendation systems. The Mapreduce is build into the movie recommendation system successfully. The two different datasets (movie description from CMU and user rating from movielens) are merged successfully. We could provide the recommendation list of movies is based on both the movie content and the user ratings. The top 15 movie are highly relevant to the user's profile of movie features. By applying the content based algorithms, we tried our best to offering suggesting movies to users with MAP higher than 70%.

Limitations:

Normally we assume that the user rating will improve the quality of recommendation from the perspective of client. But the datasets in movielens is not complete. There are missing user ratings for lots of movies. Further, some popular movie serious with the duplicate name for example "snow queen" which was release several times with the same name. It is hard to tell which user rating goes to which movie if their name is the same since they didn't specify other important features in their datasets.

Our recommendation only covered some of those important features of movie, probably not comprehensive enough to describe a movie completely and affecting user's choice drastically. There are other features that is not as immediately apparent as actor, movie name, summary. To give a recommendation with high quality, we need to do a research on which are the factors mostly affecting the user's choice of movie. For example, the popularity of the movie, if a user enters a specific year or specific genre of movie, the the blockbusters of that year of that genre should have a higher grade than the other movies.

We should also be aware of a slightly different precision that make sense in many applications. It is very difficult to predict every relevant movie in such a short recommendation list say Top 15. In our applications, the recommendation system won't offer users a ranking of all items, especially you are choosing Top 15 movies out of 42,000. Rather it is only necessary to suggests a few that

the user should value highly. It may not even be necessary to increase the precision to 100 %, but only to find a large subset of those that the user might like.

Future Work

The movie score algorithm is designed by experience. A liner machine learning algorithm should be added in the future. The machine learning algorithm can modify the weight of scores from different features of movie. Then our movie recommendation system will be more robust. To achieve with this, we also need a trainset about the movie watching history of some users.

In this version, we only use some simple word matching methods to match the content of features of movies. The current used NLP methods contains stop list, punctuation remove, uppercase to lowercase. In the future work, some complex NLP method should be added as similarity between two words, the dep and pos of sentence and the word tokenization.

Given-Movie-Name only accept one movie at each search. If our search could accept more movie from the user, the recommendation would be more meaningful.

Challenges Faced

1. The framework of this project is designed very carefully. We tend to repeat to use some MapReduce and Java applications to reduce the complexity of the system.

2. Two different data sets have been computed together in this project. So we need to consider the format uniform of contents between two data set.

3. There are 4 MapReduce and 3 java applications implemented in this project. So we need to build a good framework of the system to find bug easily. We use the shell script to run the MapReduce and Java applications one by one. The middle files are stored to help finding bugs.

4. Because the movie score algorithm is designed by our experience, some parameters in this algorithm need to be optimized. It is a large labor work. We need analyze the results of evaluation and modify the parameters or algorithm based on them. Evaluations should be done again and again.

Conclusion

At first glance, our movie recommendation system looks like every other film search engine. It is simple enough, you input the name of a film or an actor you like, and it returns the results that offers similarities to your favorites. But our recommendation goes well beyond a simple results page. The content-based filtering system are build on top of MapReduce a parallel processing framework from Apach. As the huge volume of both rating data and the movie data stress the recommender system, MapReduce would dramatically improve the scability of recommender systems.

From the perspective of the datasets, the recommendation system take advantage of both the user rating and the movie content for filtering the movies and make recommendations, although these two data sets have different schema and different movie ID. And the user rating could affect some of the movies on the movies list generated by content-based filtering.

From the perspective of client, we provide two ways to have them profile their preferences. One side, the user could provide their preference on the movies to describe their personal tastes, like the actor, the country, the genre and so on. Or if you are too lazy to do that, simply searching "The Godfather", the recommendation will extract the features of "The Godfather" like the actor, the summary, the language, the country etc. Based on the parameters or features that you provided or we extracted, combining with the rating of the movie by other users, our recommendation system immediately refines the best relevant film for you, instead of the blockbusters or some dull and lousy movies.

Team member contributions

Framework Design: Yu Zhao

Coding: Yu Zhao, Ting Wang

Evaluation: Ting Wang

The codes authors:

1. finalProject_java: Yu Zhao, Ting Wang
2. finalProject_movielenID_score: Yu Zhao
3. finalProject_IDtoContent: Yu Zhao
4. finalProject_user_description: Yu Zhao, Ting Wang
5. finalProject_wikiID_name_movielenID: Yu Zhao
6. first.sh of case 1,2,4: Yu Zhao
7. first.sh of case 3: Ting Wang

References

1. https://en.wikipedia.org/wiki/Information_retrieval

2. https://en.wikipedia.org/wiki/Collaborative_filtering

3. https://en.wikipedia.org/wiki/Recommender_system

4. Y. Koren, "The BellKor solution to the Netflix grand prize,"

5. www.netflixprize.com/assets/GrandPrize2009<http://www.netflixprize.com/assets/GrandPrize2009> BPC BellKor.pdf 2009.

6. Sarvdeep Singh Bindra, "Movie Recommendation Using Map Reduce" Rochester Institute of Technology

7. "Write a Hadoop MapReduce program for Movie Recommendation System" https://cdac.in/index.aspx?id=ev_hpc_movie-recommend-mr1

8. ZhiDan Zhao, Mintsheng Shang “User-Based Collaborative-Filtering Recommendation Algorithms on Hadoop,”, WKDD 2010.

9. Izzat Als madi and Maryam Nuser; “String Matching Evaluation Methods for DNA Comparison.” International Journal of Advanced Science and Technology Vol 47. October 2012

10. Mahiye Uluyagmur, Zehra Cataltepe and Esengul Tayfur; “Content Based Movie Recommendation Using Different Feature Sets.” Proceeding of the World Congress on Engineering and Computer Science 2012 Vol I, WCECS 2012, October 24-26, 2012, San Francisco, USA

11. Chapter 9 “Recommendation Systems”, Standford infoLab,

http://infolab.stanford.edu/~ullman/mmds/ch9.pdf