

Project Report for Inf 553

Movie Recommendation System

Zhihao Rong
zhihaoro@usc.edu
6829894545

Wei Shi
shi590@usc.edu
6024896192

Jianwen Xiao
jianwenx@usc.edu
3693620642

1 Problem

Nowadays, more and more people watch and rate movies on the website. It is possible that we can recommend movies to users based on ratings and attributes of movies. In this project, we implement three different ways: **content-based, collaborative filtering methods (including neighborhood methods and ALS algorithm)**.

2 Challenges

The key problem for **collaborative filtering methods** is large amount of data. Movie_medium.csv contains 1 million ratings among 20291 users and 10183 movies. Traditional methods are both time and space consuming. Optimization including parallel computation in Spark is needed to raise efficiency. For **Content-based method**, the key point is which attributes should be included in user/movie profile and what data structure should be used. Particularly, How to parse with string/json format data into profiles.

3 Solutions

3.1 Content-Based Recommendation

3.1.1 data set: The inputs include two files, **movies_metadata.csv** and **credits.csv**
Merge these two files by the shared column "id" (movie_id)

Credits.csv contains 45505 rows and 3 columns:

Cast: List of actor-character mappings in json format

Credit: List of person-job mappings in json format (including director, producer..)

	A	B	C
1	cast	crew	id
2	{'cast_id': 14, 'character': 'Woody (voice)', 'credit_id': '52fe4284c3a36847f8024f95', 'gender': 2, 'id': 31, 'name': 'Tom Hanks', 'or_	{'credit_id': '52fe4284c3a36847f8024f49', 'department': 'Directing', 'gender': 2, 'id': 7879, 'job': 'Director',	862
3	{'cast_id': 1, 'character': 'Alan Parrish', 'credit_id': '52fe44bfc3a36847f80a7c73', 'gender': 2, 'id': 2157, 'name': 'Robin Williams',	{'credit_id': '52fe44bfc3a36847f80a7cd1', 'department': 'Production', 'gender': 2, 'id': 511, 'job': 'Executive	8844
4	{'cast_id': 2, 'character': 'Max Goldman', 'credit_id': '52fe466a9251416c75077a8d', 'gender': 2, 'id': 6837, 'name': 'Walter Matth	{'credit_id': '52fe466a9251416c75077a89', 'department': 'Directing', 'gender': 2, 'id': 26502, 'job': 'Director	15602
5	{'cast_id': 1, 'character': 'Savannah 'Vannah' Jackson', 'credit_id': '52fe44779251416c91011aad', 'gender': 1, 'id': 8851, 'name': ''	{'credit_id': '52fe44779251416c91011acb', 'department': 'Directing', 'gender': 2, 'id': 2178, 'job': 'Director',	31357
6	{'cast_id': 1, 'character': 'George Banks', 'credit_id': '52fe44959251416c75039eb9', 'gender': 2, 'id': 67773, 'name': 'Steve Martin'	{'credit_id': '52fe44959251416c75039ed7', 'department': 'Sound', 'gender': 2, 'id': 37, 'job': 'Original Music	11862
7	{'cast_id': 25, 'character': 'Lt. Vincent Hanna', 'credit_id': '52fe4292c3a36847f80291f5', 'gender': 2, 'id': 1158, 'name': 'Al Pacino',	{'credit_id': '52fe4292c3a36847f802916d', 'department': 'Directing', 'gender': 2, 'id': 638, 'job': 'Director',	949
8	{'cast_id': 1, 'character': 'Linus Larrabee', 'credit_id': '52fe44959251416c75039d97', 'gender': 2, 'id': 3, 'name': 'Harrison Ford',	{'credit_id': '52fe44959251416c75039da9', 'department': 'Directing', 'gender': 2, 'id': 2226, 'job': 'Director',	11860
9	{'cast_id': 2, 'character': 'Tom Sawyer', 'credit_id': '52fe46bdc3a36847f810f771', 'gender': 2, 'id': 53283, 'name': 'Jonathan Taylor	{'credit_id': '52fe46bdc3a36847f810f797', 'department': 'Writing', 'gender': 2, 'id': 2075, 'job': 'Screenplay	45325
10	{'cast_id': 1, 'character': 'Darren Francis Thomas McCord', 'credit_id': '52fe44dbc3a36847f80ae0e3', 'gender': 2, 'id': 15111, 'nam	{'credit_id': '52fe44dbc3a36847f80ae0f1', 'department': 'Directing', 'gender': 2, 'id': 37710, 'job': 'Director	9091

Movies_metadata.csv contains 45467 rows and 12 columns (after I remove obviously useless columns):

Id: movie id

Other columns includes but not restricts to: gennr, language, title, overview

	A	B	C	D	E	F	G	H	I	J	K	L
1	budget	genre	id	language	title	overview	popularity	production_c	production_c	release_date	revenue	runtime
2	30000000	{{'id': 16, 'na	862	en	Toy Story	Led by Wood	21.946943	{{'name': 'Pi	{{'iso_3166_	1995/10/30	373554033	81
3	65000000	{{'id': 12, 'na	8844	en	Jumanji	When sibling	17.015539	{{'name': 'Tr	{{'iso_3166_	1995/12/15	262797249	104
4	0	{{'id': 10749,	15602	en	Grumpier Ol	A family wed	11.7129	{{'name': 'W	{{'iso_3166_	1995/12/22	0	101
5	16000000	{{'id': 35, 'na	31357	en	Waiting to E	Cheated on,	3.859495	{{'name': 'Tv	{{'iso_3166_	1995/12/22	81452156	127
6	0	{{'id': 35, 'na	11862	en	Father of the	Just when G	8.387519	{{'name': 'Sa	{{'iso_3166_	1995/2/10	76578911	106
7	60000000	{{'id': 28, 'na	949	en	Heat	Obsessive m	17.924927	{{'name': 'Re	{{'iso_3166_	1995/12/15	187436818	170
8	58000000	{{'id': 35, 'na	11860	en	Sabrina	An ugly duck	6.677277	{{'name': 'Pa	{{'iso_3166_	1995/12/15	0	127
9	0	{{'id': 28, 'na	45325	en	Tom and Huc	A mischievou	2.561161	{{'name': 'W	{{'iso_3166_	1995/12/22	0	97
10	35000000	{{'id': 28, 'na	9091	en	Sudden Deat	International	5.23158	{{'name': 'Ur	{{'iso_3166_	1995/12/22	64350171	106

Data Cleaning:

There are data duplicates. Remove 106 rows after de-duplicating by id.

3.1.2 Algorithm Description

a.No matter user input a single movie or a set of movies, first task is generating a user profile by taking the average of each profile values (by key) from all input movies.

b.Get (compute) each movie profiles

c.Compute cosine similarity between user profile and each movie profiles as a list

- A has watched HP1, Twilight, and SW1

User	Genre_Adventure	Language_English	Release_year_2002	Director_George	...
A	2/3	1	1/3	1/3	...
...

A's profile

Title	Genre_Adventure	Language_English	Release_year_2002	Director_George	...
Harris Porter 1	1	1	1	0	...
Twilight	0	1	0	0	...
Star War I	1	1	0	1	...
Star War II	1	1	1	1	...
...

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

d. Use argsort to get movie indexes in descending order corresponding to top n cosine similarity values, recommend movies matched by these indexes.

3.1.3. About profile generation

- After logical speculation, tests and comparisons, columns used to generating item profile include title, overview, gennr (what's the topic), director, actor (stars user admires), language (user usually prefer movies in his fluent language), released date (reasons for not taking others: budget/revenue column are has some uncollected data; when movie is attractive, user doesn't care much about length, production company)

- b. I used a dictionary to maintain each profile. Matrix is not considered (as that would be too large and sparse).
- c. Columns except for overview and released date
 For each additional title word (say toy) occurrence in movies watched by a user, value of key "title_toy" increase my 1 (first occurrence initialize a {title_toy:1} pair, and in a single movie profile, value 1 is finalized once initialized)
 For director and actor, keys are "director_{director_id}", "actor{actor_id}"

- d. For released date, I leave the precision to decade, e.g.
 1995/04/03=>1990

```
if ("/" in row[9]):
    decade = int(row[9].split("/")[0])
    decade = decade - (decade%10)
    decadeKey="!decade_"+str(decade)
    if decadeKey in profile:
        profile[decadeKey] = profile[decadeKey]+1
    else:
        profile[decadeKey] = 1
```

- e. For overview, I use TF-IDF as measurement (curved down)
 I generate a global dictionary to record each non-stopword word and its inverse document frequency. And when generating item profile, I grab term frequency at the same time to compute TF-IDF value.

```
idfLib={}
for mv in dataList:
    word_only_string=re.sub("[^A-Za-z]", " ",mv[5]).lower()
    word_tokens = word_tokenize(word_only_string)
    wordList = [w for w in word_tokens if not w in stop_words]
    uniqueWordList=list(dict.fromkeys(wordList))
    for w in uniqueWordList:#count once per movie
        wordKey="!overview_"+w
        if wordKey in idfLib:
            idfLib[wordKey] = idfLib[wordKey]+1
        else:
            idfLib[wordKey] = 1
    for key in idfLib:
        idfLib[key]=math.log2(len(dataList)/idfLib[key])
# curve module
maxLibValueDivider=max(idfLib.values()*2)
for key in idfLib:
    idfLib[key]=idfLib[key]/(maxLibValueDivider)
```

However, the TF-IDF value can be significant larger than 1, so that recommendations are unhealthily concentrated too much on overview (while each overview word should be less important than title word). So I curved IDF values down with same ratio so that the maximum IDF would be 0.5. In this case, only when a word appears twice in a movie overview (very rare situation), It is as significant as other attributes.

3.1.4 performance

Generating the model takes 10 seconds, it becomes much quicker if I have saved model in a file.

3.2 Neighborhood Methods

3.2.1 Input dataset

The input of neighborhood methods is rating_medimu.csv which is a subset of the complete rating dataset. It contains 20291 movies, 10183 users and 1 million ratings. Each row is a rating which contains user ID, movie ID and rating score.

The picture on the right is an example of the input dataset.

userid	movied	rating
1	31	2.5
1	1029	3
1	1061	3
1	1129	2
1	1172	4
1	1263	2
1	1287	2
1	1293	2
1	1339	3.5
1	1343	2
1	1371	2.5
1	1405	1
1	1953	4
1	2105	4

3.2.2 Algorithm details

There are two ways of neighborhood methods: find similar users and find similar movies. We did both these ways.

When the input is a user ID, the algorithm finds the similar users based on user's high rating movies and sorts the users in descending order based on Jaccard similarity. Then it picks out the top 25 similar users and movies they have ever rated. It sorts the movies in descending order based on number of reviewers within the 25 similar users. Top 15 movies are recommended to the user.


When the input is a movie ID, the algorithm finds the similar movies compared to the input movie and sorts the movies in descending order based on Jaccard similarity. Then it picks out the top 15 similar movies and recommend these movies to the user. This part recommends movies according to a single movie the user is watching.

Jaccard similarity

Use $\text{Jaccard}(A,B) = |A \cap B| / |A \cup B|$ to reflect the similarity between two users or movies.

Note: A movie may be both reviewed by user A and user B with dramatically different score. Such movies should not be used to contribute to measure of similarity. To solve this problem, we transfer the rating to binary. Only ratings greater or equal than 3.5 are marked as positive rating as 1.

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	



	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	1			1			
B	1	1	1				
C						1	1

$\text{Jaccard}(A,B) = 1 / 4$ and $\text{Jaccard}(A,C) = 0$ after rounding.

3.2.3 Optimization on big data

Computing Jaccard between each pair is time and space consuming. There are two problems about the complete computation method: large sets and large numbers of sets.

Large sets

Large sets mean each set contains lots of items. In this project, some person have reviewed many different movies and some popular movies have been reviewed by many people. We don't want to keep the whole set when computing Jaccard similarity. Here we use minhash signature to represent the set. For the original row index, we use a hash function to provide a new row number. Then it finds the first row with number 1 and records it as the minhash of a set. If the minhash of two sets are same, they are considered as same and checked later. After the permutation of the rows, the probability for two sets to be similar equals Jaccard of the two sets. $\text{Prob}(h(S_i) = h(S_j)) = \text{Jaccard}(S_i, S_j)$. By calculating minhash signature, we save the time of comparing each items in two sets.

Large numbers of sets

Large numbers of sets mean there are a lot of sets to compare with. In this project, there are 20291 movies and 10183 users. Thus, it costs a lot of time to do the comparison. We use locality-sensitive hashing to save the time of comparing sets that are not likely to be similar. It divides the hashes into bands. Each band contains several rows. If the hashes are the same in each row, they are treated as similar in a band. We will consider pairs if they are hashed into the same bucket in at least one band. Such pairs are considered as candidate pairs and we will consider them globally later. Increasing rows will add to the probability of the hashes going to the same bucket, thus increasing the false positive rate. On the other hand, increasing bands will decrease the probability of the hashes going to the same bucket, thus increasing the false negative rate.

In the project, I set 20 total rows and calculate 20 different hashes. It needs to fix the rows and bands to set a balance between false positive and false negative rate. In finding similar users, I set 10 bands and 2 rows while in finding similar movies I set 4 bands and 5 rows. It filters parts of sets that are unlikely to be similar and saves the time of global comparison. Calculation of minhash signature and local-sensitive hashing is done parallelly by Spark. It is finished on different machines to increase the calculation speed. Spark is needed when a single machine suffers from a lack of memory or speed deficiency. To eliminate false positive and false negative rate, we can adjust hash functions to keep more pairs into consideration.

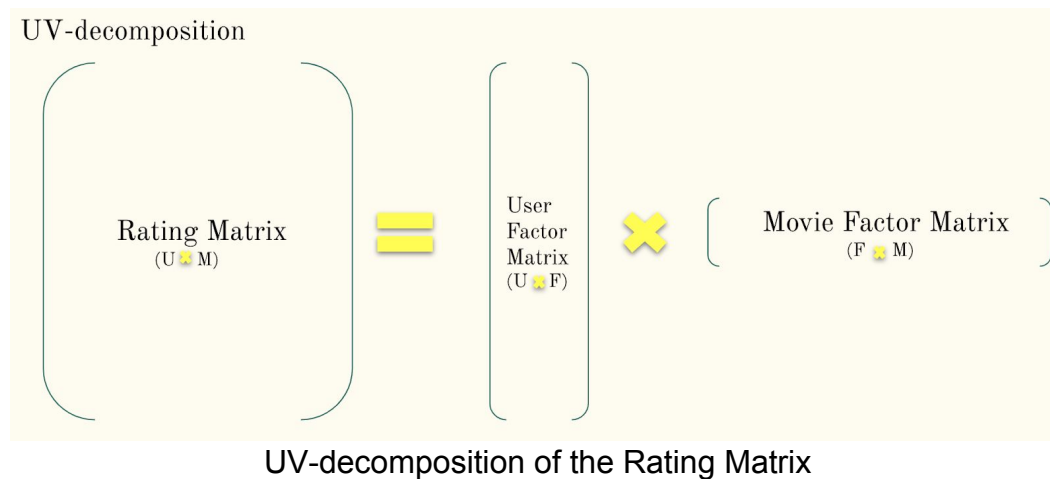
3.3 ALS Algorithm

3.3.1 Input Dataset

The dataset is the same with 3.2.1. Each line represents a rating of a movie from a user. Store the input in the Rating Matrix.

3.3.2 Algorithm Description

The training process in ALS Algorithm is to decompose the Rating Matrix($U \times M$) into two small matrices, the User Factor Matrix($U \times F$) and the Movie Factor Matrix($M \times F$). Minimize the errors between the multiplication of these two matrices and the Rating Matrix by Alternating Least Square method. For each iteration, fix one matrix and update the other matrix alternatively until the RMSE decreases slowly.



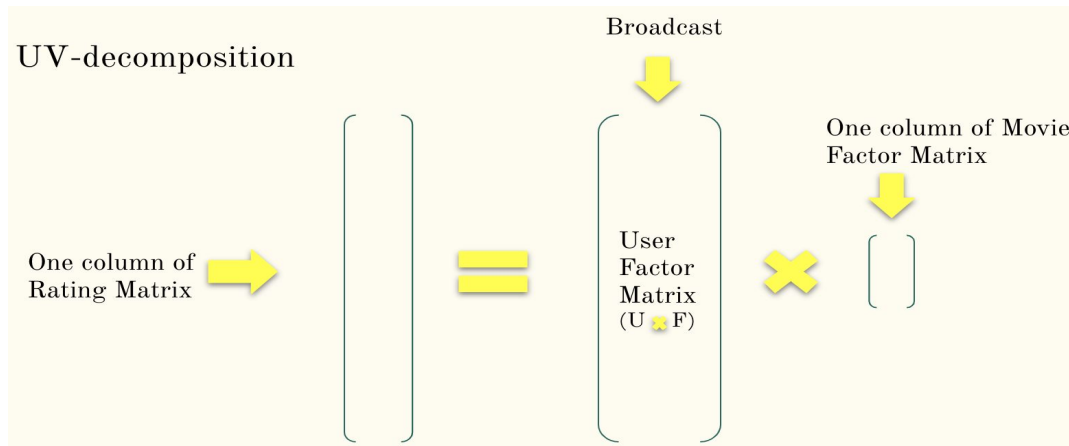
3.3.3 Algorithm Implementation

Initialization

It is crucial to initialize the model in a right way, improper initialization may lead to a terrible result. In the Movie Factor Matrix, set each element in the first column to be the average rating of the movie, put small random numbers in other positions. In the User Factor Matrix, set each element in the first column 1.0, put small random numbers in other positions as well.

Parallel Computing

In order to speed up the algorithm, divide the Rating Matrix and the matrix needed to be updated into columns. Pass these columns and broadcast the fixed matrix in each partition of our spark model, compute solutions parallelly. Finally collect the solutions and update the matrix. For the other half of one iteration, take transpose of the whole equation and follow the same procedure.



Computation for each partition in Spark

Least Squares Method

Write the matrix in this form, $y = Xw$. To compute the solution of w , apply Least Squares Method as follows. Multiply to each side of the equation, add regularization term related to the rank of the matrix to control the complexity of the model and prevent overfitting, solve for w directly and return the solution.

$$X^T y = (X^T X + \lambda I) w$$

Prediction & Recommendation

Utilize two ways to predict and recommend movies. After the execution of the ALS Algorithm, the original sparse Rating Matrix becomes dense. The blank space of the Rating Matrix has been filled with predicted ratings for the corresponding movie of the given user.

For movie recommendation given particular user, construct new Rating Matrix and select movies with top 15 highest score for the user. For movie recommendation given particular movie, compare movie vector in Movie Factor Matrix with cosine distances, output top 15 movies closest to movie.

4 Applications

For both **content based recommendation and collaborative filtering approach(neighbourhood and ALS)**, our solution can make recommendation based on user or based on movie. And in fact, such recommendations are not restricted on movies (like netflix), we can do similar things/recommendation on video website (like youtube, bilibili), technical blog website (Stack Overflow), game engine (Steam App).

For recommendation based on user, it can be used to recommend movies for a user in the rating dataset (when treating the user as a set of his/her high rating movies) or on a new user who claims his preferred movies. An available example is youtube, as user

logged in their account, the “recommended videos” division would include videos with similar attributes based on the user’s watching history/hobby.

For recommendation based on movie, it recommends movie based on a single movie. Say a user is watching a video on a website, he may see such a column on the right side, “Similar movies” (by content-based recommendation) or “People love this video also watch ...” (by collaborative filtering system).

5 Evaluation results

5.1 Comparison -- recommended by movie

5.1.1 Neighborhood method

Recommendation based on movie sorts the movie based on jaccard similarity with respect to input movie. It is likely to recommend movies of the same genre/series/topic. With input Star Wars Episode I, most of the recommended movies are action and Sci-fi movies, like Star Wars Episode I. Star Wars: Episode II, Return of the Jedi (Star Wars Episode VI), The Empire Strikes Back (Star Wars Episode V). which follows my guess.

```
You are watching movie Star Wars: Episode I - The Phantom Menace. Based on this movie, we recommend:
Star Wars: Episode II - Attack of the Clones
Men in Black
Return of the Jedi
The Empire Strikes Back
Ghostbusters
The Fifth Element
The Terminator
X-Men
Total Recall
Indiana Jones and the Temple of Doom
Back to the Future
Spider-Man
Indiana Jones and the Last Crusade
Star Wars
Back to the Future Part III
```

Recommendation result for movie Star Wars Episode I

When setting minhash signature all 0, it is equal to eradicating minhash signature and local-sensitive hashing. In this case, all the candidate pairs are kept and there is no false positive and false negative. (Running time dramatically increased) Thus, it can be used as a validation to the previous outcome. As shown in the pictures below, the it does not differ much as using the signature. Thus, previous outcomes are accurate.

```
You are watching movie Star Wars: Episode I - The Phantom Menace. Based on this movie, we recommend:
Star Wars: Episode II - Attack of the Clones
Men in Black
Return of the Jedi
The Empire Strikes Back
Ghostbusters
The Fifth Element
The Terminator
Star Wars: Episode III - Revenge of the Sith
X-Men
Total Recall
Indiana Jones and the Temple of Doom
Back to the Future
Spider-Man
Indiana Jones and the Last Crusade
Star Wars
```

Recommendation result for movie Star Wars Episode I (when eradicating minhash)

5.1.2 Content-based result and analysis

```
ShideMBP-2:info553project water$ python3 contentRec.py 15 1893
Searching...
Top Recommendations based on movies
['Star Wars: Episode I – The Phantom Menace']
Movie 1 : Star Wars: Episode II – Attack of the Clones
Movie 2 : Star Wars: Episode III – Revenge of the Sith
Movie 3 : Star Wars: The Clone Wars
Movie 4 : MacGyver: Lost Treasure of Atlantis
Movie 5 : Behind Enemy Lines
Movie 6 : 地球[テラ]へ...
Movie 7 : Robo Warriors
Movie 8 : Raiders of the Sun
Movie 9 : Star Trek: Insurrection
Movie 10 : Skyscraper
Movie 11 : Star Kid
Movie 12 : Mars
Movie 13 : Journey to the Center of the Earth
Movie 14 : Captain America
Movie 15 : Empire of Dreams: The Story of the Star Wars Trilogy
```

It is expected that star wars movies are in top recommendation because their title words are highly similar. Among these three, Star War Episode II and Episode III shares more actors with Star War Episode II than Star Wars: The Clone Wars.

When I print and compare item profiles, I found following phenomenon:

- Overview should be the weakest (and rarest) significant measurement. Usually only 3+ words duplication can be identified as an “obviously” similar, which is quite unlikely to happen (only between star wars).
- Tailing movies usually share 2+ genres and 2+ actor with input movie.
- Movie 15 is an exception. Empire of Dreams: The Story of the Star Wars Trilogy has a different genre “Documentary”, but two words matched in title and 4 matched actor_id 11184,7908,6,130 gave it credit.
- There are two “bad” result: Movie 6 and Movie 14. Their actor lists are super small (size 0, 2). They match input movies only on all three genres, but as they don’t have many “alien” actors, their cosine similarity with respect to input movie is still comparatively high (because denominator/vector size is small).

5.1.3 ALS algorithm

Parameters: 10 Iterations, 10 Latent factors, 20 partitions.

After 10 iterations, the image below shows the rmse of the first, fifth and final round of iteration. It’s clear that the the model has began to converge after the fifth round.

```
rmse for round 1: 0.248934837353
rmse for round 5: 0.213759273352
rmse for round 10: 0.213420259653
```

RMSE of the ALS algorithm

Compare movie vectors to output top 15 movies closest to the movie by cosine similarity. It's not surprising that there are lots of sci-fi movies in the recommendation.

```
You are recommending 15 movies for Movie: Star Wars: Episode I - The Phantom Menace
```

Similarity	Movie Name
0.9794	Men in Black
0.9547	Indiana Jones and the Temple of Doom
0.9360	Ghostbusters
0.9348	Back to the Future Part III
0.9306	X-Men
0.9295	Indiana Jones and the Last Crusade
0.9237	The Fifth Element
0.9226	Back to the Future Part II
0.9007	Contact
0.8924	Back to the Future
0.8818	The Mummy
0.8758	Return of the Jedi
0.8728	Armageddon
0.8686	Galaxy Quest
0.8667	Star Wars: Episode II - Attack of the Clones

Recommendation movies & similarities for Star Wars

5.1.4 overall comparison

Results by two Collaborative filtering methods (ALS and neighborhood method) are quite similar. Jaccard similarity is $10/20=0.5$, which indicates LSH results generally agree with ALS results and they didn't decrease much accuracy.

However content-based results are quite different from the other. There is only 1 common recommendation: Star War Episode II. The main reason is content-based method is based on different measurement, movie attribute, instead of rating table used by collaborative filtering method. For example, by Collaborative filtering, user tends to watch a series of movies Star War Episode I,II,III,IV,V. But on content level, Star War Episode IV,V have very different title, actor, director, (content data tables don't even know Return of the Jedi and The Empire Strikes Back are actually in Star War series.

5.2 Comparison -- recommended to User 10000

5.2.1 Neighborhood method recommendations to user

Recommendation based on user sorts the movie based on the times that have been watched by the top 25 similar users. Thus, popular movies tend to be recommended since they are watched frequently by similar user.

```
Hello, user 10000, based on the movies you review, we recommend:
Forrest Gump
Star Wars
Toy Story
The Shawshank Redemption
The Silence of the Lambs
Inception
The Lord of the Rings: The Fellowship of the Ring
Schindler's List
The Godfather
Raiders of the Lost Ark
The Lord of the Rings: The Two Towers
Ratatouille
Whiplash
The Usual Suspects
Pulp Fiction
```

5.2.2 Content-based Recommendation to user

```
ShideMBP-2:info553project water$ python3 contentRec.py 15 105 122 157336 275 544 550 603 329
Searching...
Top Recommendations based on movies
['Back to the Future', 'The Lord of the Rings: The Return of the King', 'Interstellar', ' Fargo', "T
Movie 1 : 地球[テラ]へ...
Movie 2 : The Perfect 46
Movie 3 : InAlienable
Movie 4 : Joni
Movie 5 : The Lord of the Rings: The Fellowship of the Ring
Movie 6 : Игра на выбывание
Movie 7 : The 27 Club
Movie 8 : Kara
Movie 9 : Gerry
Movie 10 : Pixels
Movie 11 : The Lord of the Rings: The Two Towers
Movie 12 : Humanity's End
Movie 13 : Captain America
Movie 14 : Don Quixote
Movie 15 : 11 Minutes Ago
```

5.2.3 ALS Recommendation movies to user

Construct new Rating Matrix multiplied by User Factor Matrix and Movie Factor Matrix, select movies with top 15 highest score. Since the matrix is extremely sparse, redistribute the ratings according to the minimum and maximum value of a given user.

```
You are recommending 15 movies for User 10000:
-----
Ratings      Movie Name
-----
4.6218       The Shawshank Redemption
4.5336       The Matrix
3.8179       Fight Club
3.5699       Forrest Gump
3.5401       Pulp Fiction
3.4782       The Lord of the Rings: The Fellowship of the Ring
3.4101       The Silence of the Lambs
3.2775       Schindler's List
3.1811       American Beauty
3.1434       The Lord of the Rings: The Return of the King
3.0029       The Lord of the Rings: The Two Towers
2.9984       Star Wars
2.8994       The Empire Strikes Back
2.7542       The Sixth Sense
2.6952       Raiders of the Lost Ark
```

The top 15 rated movies are all classic and popular, their average ratings are quite high. We can conclude that User 10000 like to watch drama films like The Shawshank Redemption, The Lord of the Rings and sci-fi movies like The Matrix and Star Wars.

5.2.4 overall comparison

Again results from two Collaborative filtering methods (ALS and neighborhood method) are similar with Jaccard similarity=9/21=0.43.

Results from Content-based method are quite different from the other two results. (Only match on two “The Lord of the Rings” movies)

5.3 Combine Content-based recommendation with neighborhood method

Input a movie (say 1893), use neighborhood method find candidate similar movies (match on at least one band). When applying content-based method, only produce item profile and calculate cosine similarity among these candidates (about 50 candidates). This time, the top 10 results matches on 6 movies (including all top 4), jaccard is 42.8%. Once search space is reduced by LSH, (among 50 candidates), content approach has much more agreement on what voting indicates.

```
ShideMBP-2:toProcess water$ python3 processAfterCol.py bymovie.txt
Searching...
Top Recommendations based on movie Star Wars: Episode I - The Phantom Menace
Movie 1 : id[1894] title[Star Wars: Episode II - Attack of the Clones]
Movie 2 : id[36657] title[X-Men]
Movie 3 : id[861] title[Total Recall]
Movie 4 : id[607] title[Men in Black]
Movie 5 : id[563] title[Starship Troopers]
Movie 6 : id[1891] title[The Empire Strikes Back]
Movie 7 : id[11] title[Star Wars]
Movie 8 : id[329] title[Jurassic Park]
Movie 9 : id[1892] title[Return of the Jedi]
Movie 10 : id[817] title[Austin Powers: The Spy Who Shagged Me]
```

Content-based result on 50 LSH candidates(top10)

```
You are watching movie Star Wars: Episode I - The Phantom Menace. Based on this movie, we recommend
Star Wars: Episode II - Attack of the Clones
Men in Black
Return of the Jedi
The Empire Strikes Back
Ghostbusters
The Fifth Element
The Terminator
Star Wars: Episode III - Revenge of the Sith
X-Men
Total Recall
```

Result from pure neighborhood method(top10)

6. Conclusion

In this project, we implement three ways to recommend movies to a user. Two collaborative-filtering approaches (neighborhood method and ALS) produce quite similar results. However content based approach is based on attribute table and provides relatively different outcomes. This follows our prediction since objective attributes may not always compromise with subjective votes, especially when dataset is enough massive.