Recommender System Project Report

November 8, 2022

1 Background

We now rely more and more on the online platforms and applications such as Netflix, Amazon, Youtube etc. We may find ourselves face a wide range of options. However, *a large array of options may discourage consumers because it forces an increase in the effort that goes into making a decision.* This is why recommender systems have become crucial for those platforms. Our project builds a recommender systems using two approaches: collaborative filtering and content based filtering. The dataset we will be using is the the small MovieLens dataset, and we will show how to recommend movies to a user based on their futures.

Types of recommender systems

- **Collaborative filtering:** The main idea behind these methods is to use other users' preferences and taste to recommend new items to a user. The usual procedure is to find similar users (or items) to recommend new items which where liked by those users, and which presumably will also be liked by the user being recommended.
- **Content-Based:** Content based recommenders will instead use data exclusively about the items. For this we need to have a minimal understanding of the users' preferences, so that we can then recommend new items with similar tags/keywords to those specified (or inferred) by the user.
- **Hybrid-Based:** Content-based include techniques combining collaborative filtering, content based and other possible approaches.

2 Process Design

2.1 Flow chart

We implemented both content based recommender system and collaborating filtering recommender system. In our program, these two options can be chosen separately and they will give the different results.

We loaded the data first. After data pre-processing, we used the user id, movie id and users' ratings to generate the weighted matrices according to the tf-idf values of the movies and the ratings of movies given by different users. In this way, it can be reused and save the computational time. (The details are interpreted in the implementation part).

Then, we implemented content based and collaborating filtering recommender algorithm separately. The content based algorithm ranks the movies based on the cosine similarities. The collaborative filtering algorithm predicts the ratings that users may give to the unseen movies according to the similar users.

The system will give the top k movies with the highest probability that the target user may like.

In addtion, we implemented three optimization methods as comparison. 1). Locality Sensitive Hashing (LSH). 2). Faiss similarity vector search engine. 3) Singular value decomposition (SVD) for matrix factorization. We used LSH and Faiss as the methods of retreival part, and used it to find the similar movies. They can dramatically speed up the program.

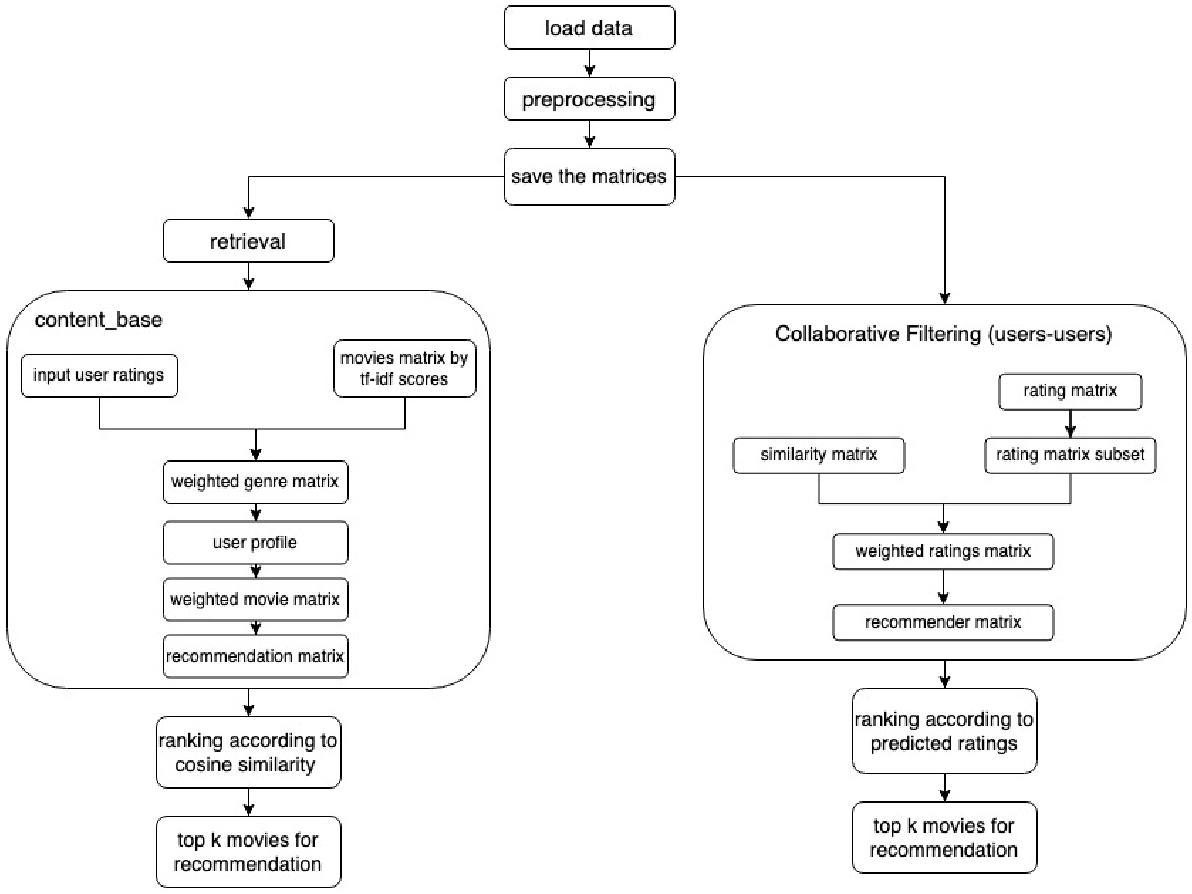
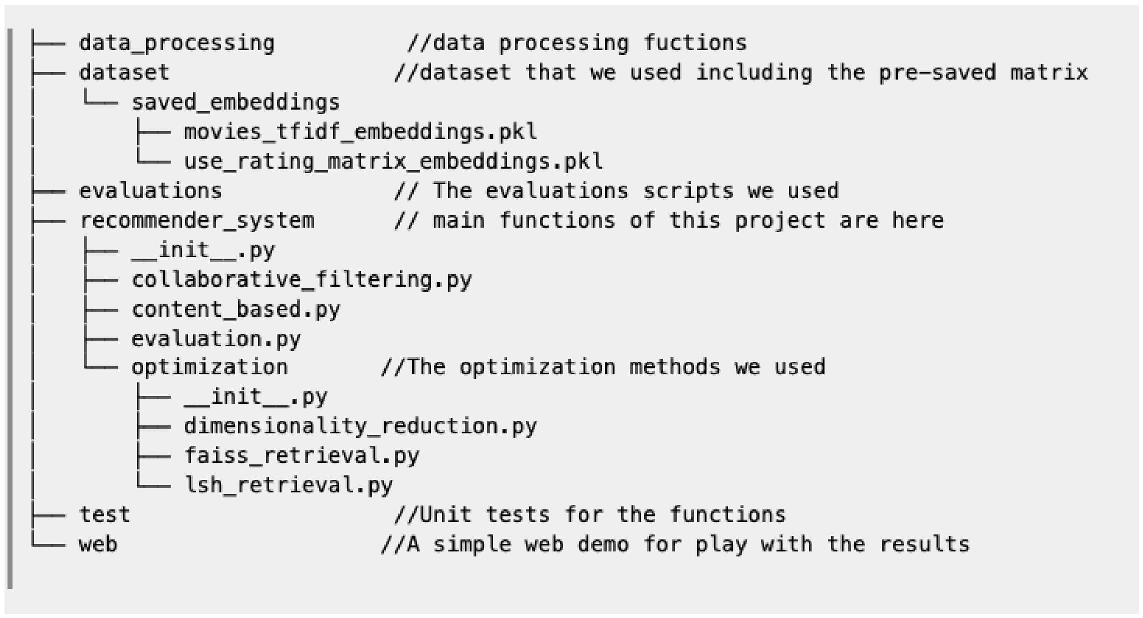


Figure 1: The flow chat of our process design of the project

2.2 Code structure



```
├── data_processing          //data processing fuctions
├── dataset                 //dataset that we used including the pre-saved matrix
│   └── saved_embeddings
│       ├── movies_tfidf_embeddings.pkl
│       └── use_rating_matrix_embeddings.pkl
├── evaluations             // The evaluations scripts we used
├── recommender_system      // main functions of this project are here
│   ├── __init__.py
│   ├── collaborative_filtering.py
│   ├── content_based.py
│   ├── evaluation.py
│   └── optimization       //The optimization methods we used
│       ├── __init__.py
│       ├── dimensionality_reduction.py
│       ├── faiss_retrieval.py
│       └── lsh_retrieval.py
├── test                    //Unit tests for the functions
└── web                    //A simple web demo for play with the results
```

Figure 2: The code structure of the project

The code structure is shown above. There are six main folders including the pre-processing, dataset, evaluations, unit test, and recommender system where the main algorithm has been implemented. It includes the content based and collaborative filtering algorithms and three optimization methods mentioned above. In addition, we added a simple web demo to displace the results conveniently.

3 Data Processing

One of the most used datasets to test recommender systems is the MovieLents dataset, which contains rating data sets from the MovieLens web site. The small ML dataset we are using contains 9724 movies rated by 610 users. Let's first get a glimpse of the data. We have three .csv files: ratings, tags, and movies. The filed will be loaded as pandas dataframes. In this project, we have only used ratings.csv and movies.csv. The distribution of the data is shown as follow:

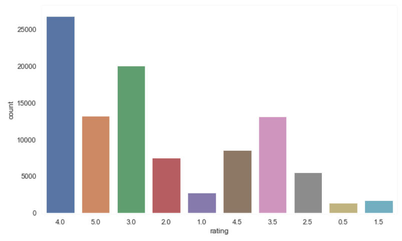


Figure 3: Counts of Moving rating score by the user

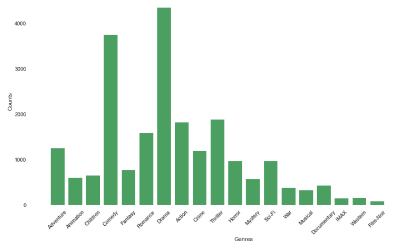


Figure 4: The most frequent genres

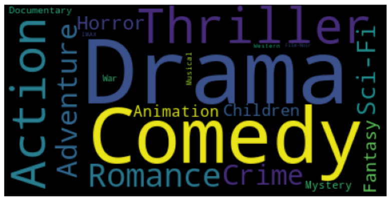


Figure 5: Words cloud for popular genres

4 Implementation

4.1 Content-based

1. generate the movie matrix based on the genres using tf-idf.
In this project, we will use a tf-idf vectorized matrix to buid a content-based recommender. The expression is defined as follows:

$$tfidf_{i,j} = tf_{i,j} \times log(\frac{N}{df_i}) \tag{1}$$

Which in our case:
$tf_{i,j}$ = total number of a given genre occurs in genres of a movie
df_i = total number of genres of movies containing a given genre
N = total number of genres of movie
Thus, the lesser the amount of movies that contain a given genre(df_i), the higher the resulting weight. As a result, tf-idf will help capture the important genres of each movie by giving a higher weight to the less frequent genres, which we won;t get in the other way.
The output of this step will be a matrix of size $numberofmovies \times genresofmovies$

2. generate the users ratings towards different movies, to get a weighted genre matrix based on the movies that the user has watched.
In this part, we will output a matrix of size $1 \times numberofmovies$, with the ratings of the movies for

each user.

3. aggregate the weighted genres and then normalize them to get the user profile.
In this part, we will apply dot multiplication on the $tf - idf$ matrix and the vector of the users rating to get the predicted user profile matrix. The output will be a 1-d vector with size $1 \times numberofgenres$

4. get the movies that have not been watched by the user, get the candidate movies matrix. We don't want to recommend to the users the movie that they have already watched. Therefore, this part we will create a matrix that only contains the candidate movies. The following part which calculate the con-sinus similarity will thus only applys to those movies.

5. multiply user's profile with the movie matrix to get the weighted movies matrix and recommendation matrix which is the recommendation list, return the movies with higher scores.
After encoded each movie's genre into its $tf - idf$ representation, the next step will be to find similar vectors (movies).

This similarity measure owns its name to the fact that it equals to the cosine of the angle between the two vectors being compared. The lower the angle between two vectors, the higher the cosine will be, hence yielding a higher similarity factor. It is expressed as follows:

$$similarity = cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} \tag{2}$$

So here, we'll be obtaining the cosine by taking the inner product between both vetors, and normalising by their norms.

We then multiply the user's ratings for the movie with the candidate movie matrix and we obtain a matrix of size of the number of the unseen movie. The recommendation will thus based on the ranking of the co-sinus score.

4.2 Collaborative Filtering

The difference between the collaborative filtering and content based filtering is that here we recommend the movies by taking into the consideration of the profile of other users. We predict the rating of the movies for each user by applying the similarity matrix between users.

1. generate users ratings matrix based on the movies that different users have watched.
This part we will retrieve a matrix with dimension $numberofusers \times numberofmovies$, the movies that users haven't seen will be replaced by zero

2. calculate the similarity matrix between different users based on the movies that they have watched.
This part, we will input a 1-d vector of size $lengthofmovies$ for a given user id and calculate the co-sinus similarity with all the other users in order to pick similar users for the given input user id. The output of this step is a vector of size $numberofusers$

3. get the subset of ratings matrix from the users who have watched recommended candidates movies.
This part we retrieve a subset of candidate matrix in which contains the movies that the user haven't seen, the output matrix size is $thenumberofusers \times numberofcandidatemovies$

4. multiple the subset ratings matrix and similarity matrix, to get the weighted ratings matrix.
In this part, we will out put a matrix of same size as the step three, but with weighted ratings.

5. sum the weighted ratings for recommended candidates movies according to different users.

Since we have obtained a matrix of a selected user for all the weighted ratings of all the movies based on the similarity between other users, we sum them in order to have a final ratings for the user.

6. normalize the values by diving the sum with the sum of cosine similarity from different users.
In this part, we normalize the value so the rating will be in the range of zero to five.

4.3 Analysis of the data structures

4.4 Optimizations

4.4.1 Locality Sensitive Hashing (LSH)

The basic idea of LSH in retrieval process is that We first perform hash mapping on the vectors in the original data space to obtain a hash table. Then two adjacent vectors in the original data space will be mapped to the same bucket after the same hash transformation. The probability of non-adjacent vectors being mapped to the same bucket is small.

Therefore, in the retrieval process, we can map all the item vectors into different buckets, and then map the user interest vectors into the buckets. At this time, we only need to take the inner product of the user vector and the item vector in the bucket. In this way, the calculational time is greatly reduced.

4.4.2 Faiss similarity vector search engine

Faiss is short for facebook AI Similarity Search developed by facebook, which is a library for efficient similarity search and clustering of dense vectors.

For similarity search, Faiss builds a data structure in RAM from a set of vectors x_i in dimension d. After constructing the structure, when given a new vector x in dimension d, it can efficiently perform the operations:

$$j = argmin_i ||x - x_i|| \tag{3}$$

where $|| - ||$ is the Euclidean distance. The data structure is an *index*, an object that has an *add* method to add x_i vectors. Computing the argmin is the *search* operation on the index.

Faiss has been used a lot in the industry as the retrieval process of the recommender system.

4.4.3 Singular value decomposition (SVD) for matrix factorization

SVD is a method of matrix factorization. Any matrix can be decomposed into the form of multiplying several matrices by the SVD method following the formular:

$$A = U\Sigma V^T \tag{4}$$

In our case, the matrix A of the recommendation system is a sparse matrix, which basically cannot be decomposed and cannot be directly SVD, but we still can use the idea of decomposition. Thus, we use the machine learning method to learn each element of the matrix by minimizing the root mean square error (RMSE) between true values and predicted values.

Specifically, the i row of matrix U is multiplied by the j column of matrix V to obtain the elements of A. Then the $A[i][j]$ element with the true value is used as the label, and the RMSE is continuously used to reduce all the labels in A. After obatining matrices U and V, we use obtained U and V to predict the missing values in A.

Training is implemented in a similar way to other machine learning methods that an algorithm tries to optimize its predictions as close to the true value as possible. We used the "scikit-surprise library" to implement SVD algorithm. It predicts the rating of a certain user-movie combination and compare the predicted value with the true value. RMSE was used as the measure metric.

5 Analysis Experimental Evaluation

In this part, we are going to evaluate the recommender system with two aspects: computational time and accuracy. We will also compare how well the optimization methods have approved the system.

5.1 Computational issues

The problem that arises when we recommend items from large data sets is that there may be too many pairs of items to calculate the similarity of each pair. Also, it's likely that we will have sparse amounts of overlapping data for all items.

In order to save the computational time, we saved the tf-idf values matrix of all movies and users ratings matrix for different movies in advance. Otherwise, it will compute the matrices for every time we launch the prediction. It took around 20s to calculate everything in our case, which is too slow for an online recommender system.

We implemented collaborating filtering algorithm first. However, we can see the computational time is still very slow (around 1.5s). After analyzing the code, we found out it is because when we used numpy for mathematical computations, there were some steps took too long as the dimensions of our matrices for users and movies are big.

Then, we implemented the content based recommender system, which calculates the cosine similarity of different movies. The computational time reduced significantly.

The content based RS relies on the feature of the items, for N items. The collaborative filtering based method is also computationally expensive. For M users and N items, its time complexity is $O(M * N)$, but since the data is mostly sparse i.e. a user only buys a handful of items out of the whole lot, the complexity is approximately $O(M + N)$.

Today, there are different methods of dimensional reduction that converts the hugely interaction matrix to a dense one like the one we have applied: SVD. We also use algorithmic enhancements that vastly narrow down the search space for a vector's k-nearest neighbours which allow it to have a much faster similarity search. The two methods we apply here are LSH and FAISS.

number of k	collaborative filtering (ms)	Content based (ms)	Faiss (ms)	LSH (ms)
5	1558	60	9.4	47
10	1552	130	12	75.5
15	1555	106	21.9	97.4
50	1832	90.7	13.9	70.8
100	1552	90.5	16	83.5

Table 1: Comparison of computation time for one user

After compare the computational time with different k, we can conclude that FAISS improve the result significantly. It almost reduces the computational time by 90 percent.

5.2 Accuracy evaluation

Evaluation will follow the same process above for the prediction, the only difference here is that we will evaluate only for the movie already rated by the user with a test size 0.2. For example, if a user has seen 100 movies, we will use 80 movies to train the model and calculate the difference between the golden data of the rest 20 movies with the prediction.

We will use two methods to evaluate our model:

- **Root Means Squared Error:** RMSE is very popular for the evaluation of the recommender system. It is computed by comparing the predicted rating to the true rating for each user-item pair with a known label. The result is on the same scale as the original ratings.

- **Precision@k:** Precision is classical evaluation metrics in binary classification algorithms and for document retrieval tasks. Here, we need to transform our problem into a binary problem. In the context of recommendation systems we are most likely interested in recommending top-N items to the user. So it makes more sense to compute precision metrics in the first N items instead of all the items. Thus the notion of precision and recall at k where k is a user definable integer that is set by the user to match the top-N recommendations objective.

During our implementation, the content based recommender system will be evaluated by RMSE and the CF recommender system will be evaluated by precision@k. This is because with the first method, we have a matrix of all the weighted ratings, however, in the second method, we have a matrix of cosinus similarity and recommend by the ranking of their similarities, thus we are not able to calculate the RMSE.

5.2.1 Root Mean Square Error(RMSE)

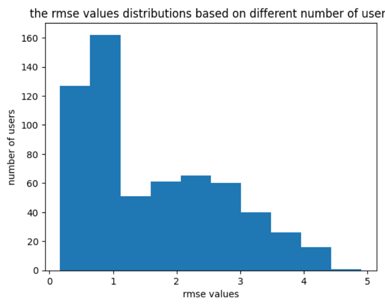


Figure 6: hist representation for RMSE

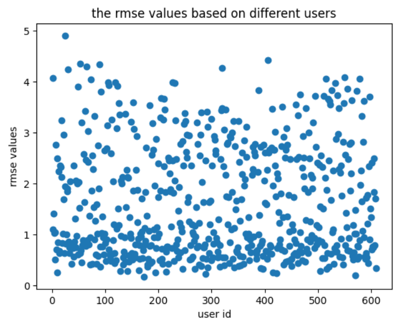


Figure 7: scatter representation for RMSE

As the collaborative filtering algorithm predicts the ratings that a user may have for the unseen movies. Thus, we have calculated the RMSE for different users by diving them into training dataset and test training dataset with the test-size=0.2 as shown above. In Figure 6 and Figure 7, we can observe that the RMSE are between 0-2 for most of the users.

Then we take the mean of all RMSE for 610 users. The final RMSE for the collaborative filtering recommender system is 1.6259.

We also did a test by using the SVD algorithme, the RMSE is improved greatly by 46%, with a RMSE of 0.8754 for all the users.

Collaborative filtering	SVD
1.6259	0.8754

Table 2: Comparasion of RMSE between CF and SVD.

5.2.2 precision@k

Precision @ k = (of recommended items @k that are relevant) / (of recommended items @k) Here we take the intersection of the recommended items with the test data, and we will test for different k with a test size of 0.2.

After adding the optimization method, the accuracy decreased. It is reasonable because we did the trade-off between the accuracy and efficiency. Theoretically, faiss should have performed better than the results we got. However due to the time limitation, we were not able to run more tests to improve the result. We believe that with a larger amount of dataset, the accuracy can be improved. It can be a future experimentation left for us.

number of K	Content based	Faiss	LSH
5	0.3	0	0.2
10	0.3	0	0.24
15	0.08	0	0.067
50	0.08	0.01	0.05
100	0.06	0.01	0.05

Table 3: Comparison of precision@k for different models

6 Conclusion

Overall, we have seen how a simple content based and collaborative filtering recommender system can provide fairly good recommendations. One advantage of content based recommendations is that it will not encounter cold-start problem, since we only need basic information on a user to provide similar recommendations based on the items. An important drawback however is that it tends to recommend the same type of items to the user. With collaborative filtering, the user is able to be recommended with a different type of item, since the match here is done between neighbouring users with similar tastes, but different items rated. Due to the time limit, we have only implemented users-users collaborative filtering, however we can also implement items-items collaborative filtering.

In order to optimize algorithms, we have implemented two retrieval methods: LSH and faiss and one dimension reduction method: SVD. By comparison, we can conclude that different optimization methods improve the recommender system significantly in accuracy and computational efficiency separately.

The implementation of our recommender system is naive but powerful, however, nowadays the recommender system use more and more deep learning model, such as WideDeep, DeepFM, DIN(Deep Interest Network), Multi-task recommenders etc,. But they are not in the range of our experimentation.