



CPSC 597 Project

Movie Recommender

Project Advisor:

Dr. Lidia Morrison

Submitted by:

Fawad Ali

Department of Computer Science
California State University, Fullerton
Spring 2022

Table of Contents

1 Abstract	4
2 Introduction	4
2.1 Description of the Problem	4
2.2 Purpose	5
2.3 Objectives	5
2.4 Environment	6
3 File Structure	6
4 Functional Requirements	7
5 UML Diagrams	9
5.1 Use Case Diagram	9
5.2 Activity Diagram	10
6 System Architecture	11
6.1 Architecture Design	11
7 Recommender Systems Evaluation	11
7.1 Dataset	11
7.2 Evaluation Metrics	15
7.3 Group Bayesian Personalized Ranking	16
7.3.1 Overview	16
7.3.2 Algorithm	16
7.3.3 Implementation	18
7.3.4 Testing and Results	19
7.4 LREC	21
7.4.1 Overview	21
7.4.2 Implementation	22
7.4.3 Testing and Results	22
7.5 TSUISIMCF	24
7.5.1 Overview	24
7.5.2 Similarity Metrics	24
7.5.3 Algorithm	25
7.5.4 Testing and Results	26
7.6 Genre Correlation	27
7.6.1 Overview	27
7.6.2 Algorithm	27
7.6.3 Testing and Results	28
7.7 Term Frequency-Inverse Document Frequency	28
7.7.1 Overview	28

	3
7.7.2 Algorithm	29
7.7.3 Testing and Results	30
7.8 TSUISIMCF and TF-IDF	30
7.8.1 Overview	30
7.8.2 Testing and Results	30
7.9 LREC and TF-IDF	31
7.9.1 Overview	31
7.9.2 Testing and Results	31
7.10 Overall Results	32
8 User Interface	35
9 Possible Improvements	38
References	40

1 Abstract

For this project I created a web application called Movie Recommender. This application asks a user for a list of movies they enjoyed and then those movies are used to recommend a list of new movies to the user. The recommendations are made by using a recommender system. Recommender systems are algorithms that recommend items to users. These items include movies, music, and articles.

2 Introduction

One of the most common forms of entertainment for people is watching movies. There are many movies out there for people to watch, but it is difficult for people to decide which one to watch. One way to recommend movies to people is by recommender systems.

Recommender systems are systems that output personal recommendations to a user (Burke, Robin). Recommender Systems can be placed into three categories. One category of recommender systems is called content-based filtering. In these systems, items are recommended to a user by finding similar items the user liked in the past (Adomavicius, Tuzhilin). Another category is called collaborative filtering. Collaborative filtering systems recommend items to a user by using other users that have similar ratings as they do (Sarwar, Karypis, Konstan, Riedl). The final category are hybrid approaches. These approaches combine content-based systems and collaborative filtering systems (Adomavicius, Tuzhilin).

2.1 Description of the Problem

The application I created is called Movie Recommender. What this application does is it gets a list of movies a user enjoyed and uses those movies to recommend new movies for the user. When the user inputs a movie they like they are not asked for a numerical rating and they do not input movies they dislike. Because of this, the recommender systems I worked with involved unary ratings. These are ratings that are assigned to a single class. Usually the class is if the user had a positive view of an item (Schröder, Thiele, Lehner). An example of unary rating is if a user liked an item the rating of the item would be 1. The rating of the item would be 0 if the user either dislikes the item or has no opinion of the item.

2.2 Purpose

For a previous class I read papers on recommender systems and tested them out on a dataset and compared how each system performed. The recommender systems I tested predicted what numerical rating a user would give to a movie. After completing that project, I realized that many people do not give numerical ratings to movies they watch. Another thing I noticed is that many services that may use recommender systems such as Netflix and Hulu, do not ask users to give numerical ratings. These two realizations made me think that recommender systems that use numerical ratings may not be used by many users.

The type of rating that most users give is called implicit ratings. These ratings are given by users based on their actions. Examples of implicit rating would be buying a product or clicking on an article link (Schröder, Thiele, Lehner). Finishing a movie on a streaming service is also an example of implicit feedback. This could be used to indicate that a user had a positive view of the movie. Since implicit ratings are more common, I became interested in testing out recommender systems with implicit ratings. I decided to make an application for users, because I wanted to put everything I learned into practice.

2.3 Objectives

Since Movie Recommender will be using a recommender system to recommend movies to the user, the most important task of this project is to find the best performing recommender system and it needs to use unary ratings. To do this, I will have to read papers on recommender systems that use unary ratings and then test the performance of the algorithms from those papers.

Since this project is a web application, another task will be web-development. When a user enters the website, the user will be able to use the application. The user interface of the web application will allow the user to enter the movies they enjoyed. Users will also be able to remove any movies from the list of movies they enjoyed. When they have entered the movies they enjoyed, they will be able to press a submit button and receive a list of movie recommendations. The user interface will be easy for the user to use. Users will also be able to go on an instructions page that teaches them how to use the application and there will be another page that lets the user learn about the recommender system.

2.4 Environment

The programming languages used to build this application are HTML, CSS, Javascript, and Python. For web development, HTML, CSS, and Javascript are used. Bootstrap and Flask were also used. Bootstrap is a CSS framework that is used for front-end web development. Flask is a Python library that is used to build web applications.

Python will be used to build all of the recommender systems I will test. The main libraries used were Numpy and Pandas. Numpy was used to perform mathematics that involved using matrices and vectors. Pandas was used to load CSV files and it was used to create the matrices needed for the recommender systems.

All of the work that I did for this project was on my MacBook Pro. All of the code for Movie Recommender was written on Sublime Text. The recommender systems that were tested were implemented and tested on Jupyter Notebook. I used Jupyter Notebook for these tasks because I found it to be quicker to edit and rerun code. The best performing recommender system was added to the application's code.

3 File Structure

This project is contained in a folder called “597 Project”. In this folder there are two python files and four folders. One of the Python files is called app.py. This is the file we run to start the application. To run this file, open the project folder in a terminal window and type `python app.py` and press enter. After that type `localhost:5000` on a web browser and you should be able to access the application. The other Python file is called `recommender_system.py`. This file contains the code of the recommender system.

One of the folders in this project is called templates. This folder contains the html files for the application. Another folder is called static. This folder contains Javascript and CSS files of the project. It also contains any images used in this application. There is also a folder called “algorithm test”. This folder contains mostly jupyter notebooks. This is where I implement and test the recommender systems I read about.

There is a folder called data, which contains the files like `like_rating.csv`, `titles.json`, and `links.json`. The file, `like_rating.csv` contains the 610 rows and 9724 columns. Each row is a user and each column is a movie. If a user u likes a movie m , then the value at row u column m is 1.

The value of that element is 0 if they do not like the movie or they did not rate it. The json file called title.json contains the list of titles of each movie. The index of these titles correspond to the column of like_rating.csv. This means that the n-th index of this list is the title to the n-th column of the CSV file. The file links.csv is a list that contains the IMDB link to each movie. The indices of this list also correspond to the indices of movie_list.csv and the columns of like_rating.csv.

4 Functional Requirements

1. The application shall display the program when a user enters the website.

Description

After a user enters the URL to the website on a web browser, the application should display the program that recommends movies.

Pre-condition: User has access to the internet and has the URL to the website.

Post-condition: The application is displayed and the user can start using it.

2. The application shall allow the user to add any movies in the database to the list of movies they enjoyed.

Description

The application will contain a large list of movies. If a movie the user enjoyed is on the list, then they will be able to add it to their list of movies they liked.

Pre-condition: User has a movie they enjoyed and the movie is on the database.

Post-condition: The movie is added to the list of favorite movies.

3. The application shall allow the user to remove a movie from their list of favorites.

Description

As the user begins to add movies to their list, they may want to remove a movie that is in the list. The user will be able to remove a movie by clicking on an 'x' icon next to the movie title. Once a movie is removed from the list, all other movies remain in the list.

Pre-condition: User has at least one movie in their list of favorites.

Post-condition: Movie is removed from their list of favorite movies.

4. The application shall display a list of movie recommendations that are not in the user's list

Description

Once the user has entered their favorite movies, they will be able to click on a submit button. This will result in the list of movie recommendations to appear that will come from an algorithm that uses the user's favorite movies. The movies will not include any movies on the favorites list.

Pre-condition: User's favorite movies are added.

Post-condition: List of movie recommendations is shown.

5. The application shall contain a link to the IMDB page of each movie recommendation

Description

The user will be able to click on a movie recommendation and it will take the user to the IMDB page of that movie to let the user learn more about the movie.

Pre-condition: The user has received a list of movie recommendations.

Post-condition: The link to the movie's IMDB page is opened in a new tab.

6. The application shall allow the user to make changes to their list of liked movies after receiving recommendations.

Description

After a user receives a list of recommendations, they can remove and add new movies to their list of movies they enjoyed. Once changes are made, they can click on submit and receive a new list of recommendations.

Pre-condition: User received movie recommendations.

Post-condition: User's favorite movie list is changed and new recommendations are given.

7. The application shall allow the user to view the page that describes the recommender system used to make recommendations and the dataset.

Description

In the website there will be a button called "Algorithm Description and Dataset". This button will lead to a page that will describe how the recommender system works. Any papers used to make the recommender system will be referenced. The data used in the algorithm will be described and referenced.

Pre-condition: User has the website open in a browser.

Post-condition: Algorithm Description page is displayed.

5 UML Diagrams

5.1 Use Case Diagram

When the user enters the web application, they will have the ability to get recommendations for movies. To get recommendations, users will have to search for movies and add the movie to their list of favorites. Users can also want to remove a movie from their list. Once the user has added all of their favorite movies, they will be able to click submit and receive movie recommendations.

If the user is interested in learning about the process on how the movies are being selected for recommendations, they can click on “Algorithm Description and Dataset”. This will lead them to an article describing the algorithm and the dataset.

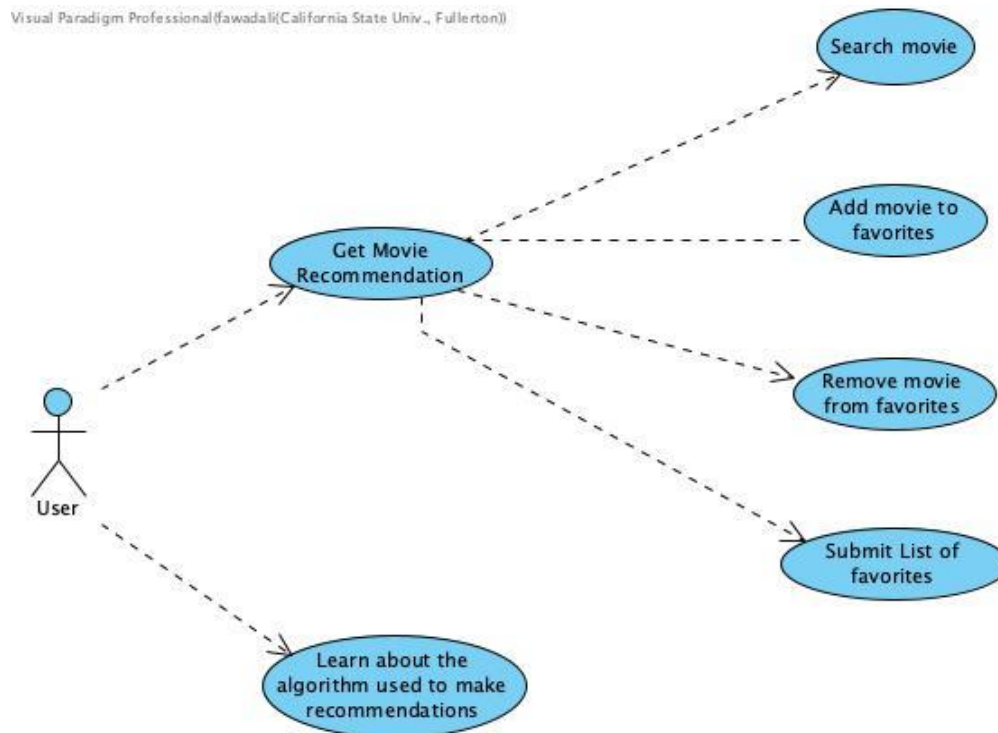


Figure 1

5.2 Activity Diagram

This activity diagram shows how a user will get a list of recommendations. The first thing a user will do is search for a movie in a text bar and then click “add”. If the movie is in the database, it will be added to the user’s list of favorite movies. After the user has a movie in their favorites list, they can add another movie, remove a movie, or they can click submit and receive a list of movie recommendations.

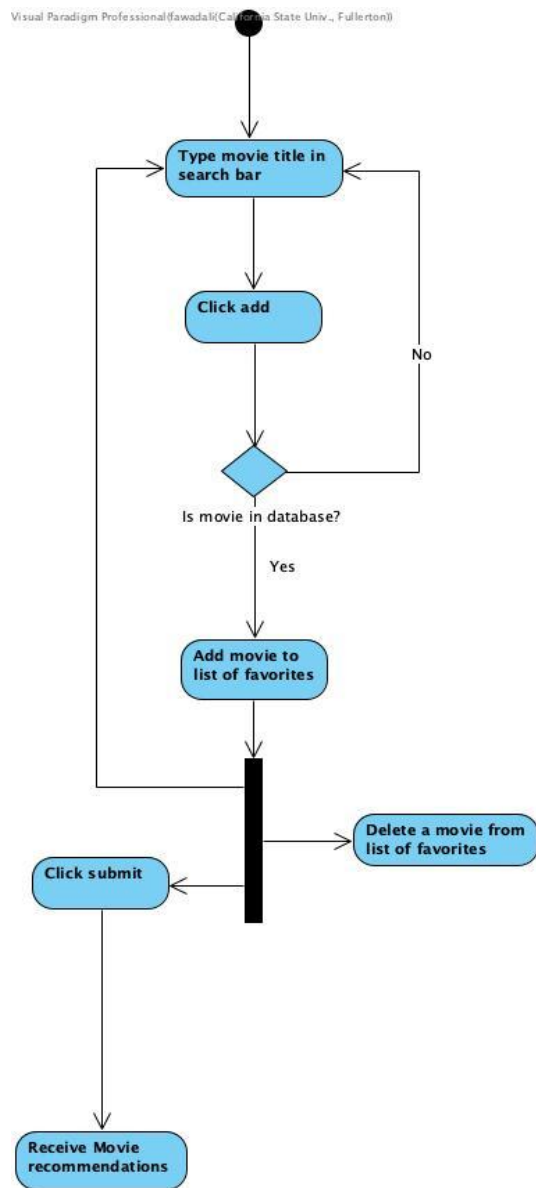


Figure 2

6 System Architecture

6.1 Architecture Design

The web application will allow a user to add movies to a list of movies they like and then they can click “submit” and receive a list of movie recommendations that will be produced using a recommender system which uses the movies the user liked. This program will contain many different components. This includes a text bar, list of movies the user enjoyed, list of movie recommendations, and recommender system.

The text bar is where the user can enter the title of a movie they like and press add to add the movie to a list of movies they liked. When the user starts typing there will be a drop down menu with suggestions. The movies the user enjoyed will be displayed below the search bar. The user will be able to remove any movies from the list by clicking on the ‘x’ next to the title.

The list of movie recommendations will be next to the list of movies the user enjoyed. The recommendations will be generated after the user clicks a “submit” button under the list of liked movies. The recommendations will come from a recommender system which will run in the background. The movies the user liked will be used to make the recommendations.

7 Recommender Systems Evaluation

7.1 Dataset

The dataset I used to test each recommender system is called ml-lastest-small. This dataset was from GroupLens, which is a research lab that is from the Computer Science and Electrical Engineering department of the University of Minnesota, Twin Cities. This dataset contains 100,836 ratings from 610 users on 9742 movies. The files on this dataset are ratings.csv, movies.csv, links.csv, and tags.csv (Harber, Konstan).

The features on ratings.csv are userId, movieId, rating and timestamp. The features userId and movieId are used to identify a user and movie, respectively. Rating is the rating a user gave to a movie. The ratings range from 0.0 to 5.0 in 0.5 increments. Timestamp is the time in which the rating was made. It is in the form of the number of seconds after January 1, 1970 (Harber, Konstan).

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
5	1	70	3.0	964982400
6	1	101	5.0	964980868
7	1	110	4.0	964982176
8	1	151	5.0	964984041
9	1	157	5.0	964984100

Figure 3

The features on movies.csv are movieId, title, and genres. This file contains the titles of the movies that were rated in ratings.csv. Genres is a list of genres of the movie. MovieID corresponds to the movieId of ratings.csv. (Harber, Konstan).

	movieId	title	genres
0	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
1	2	Jumanji (1995)	Adventure Children Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama Romance
4	5	Father of the Bride Part II (1995)	Comedy
5	6	Heat (1995)	Action Crime Thriller
6	7	Sabrina (1995)	Comedy Romance
7	8	Tom and Huck (1995)	Adventure Children
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action Adventure Thriller

Figure 4

The features on links.csv are movieId, imdbId, and tmdbId. ImdbId is what IMDB uses to identify a movie and imdbId is what TMDB uses to identify a movie. These identifiers can be

used to go on the movie page of those websites (Harber, Konstan). For example, the IMDB URL for GoldenEye is <https://www.imdb.com/title/tt0113189>. For TMDB, the URL will be <https://www.themoviedb.org/movie/710>.

	movieId	imdbId	tmdbId
0	1	114709	862
1	2	113497	8844
2	3	113228	15602
3	4	114885	31357
4	5	113041	11862
5	6	113277	949
6	7	114319	11860
7	8	112302	45325
8	9	114576	9091
9	10	113189	710

Figure 5

The features on tags.csv are userId, movieId, tag, and timestamp. The features userId, movieId, and timestamp are the same as ratings.csv. Tag is some form of information about the movie that came from the user. Each tag is either a word or short phrase (Harber, Konstan).

	userId	movieId	tag	timestamp
0	2	60756	funny	1445714994
1	2	60756	Highly quotable	1445714996
2	2	60756	will ferrell	1445714992
3	2	89774	Boxing story	1445715207
4	2	89774	MMA	1445715200
5	2	89774	Tom Hardy	1445715205
6	2	106782	drugs	1445715054
7	2	106782	Leonardo DiCaprio	1445715051
8	2	106782	Martin Scorsese	1445715056
9	7	48516	way too long	1169687325

Figure 6

The recommender systems that were tested used data that was a matrix in which the columns are movies and the rows are users. To get data in this form I used the pivot table function from the pandas library on ratings.csv. This is the code used to get this matrix.

```
ratings = pd.read_csv('ml-latest-small/ratings.csv')
df = ratings.pivot_table(index='userId', columns='movieId', values='rating', fill_value=0, aggfunc='first')

df.index.name = None
df.columns.name = None
```

Figure 7

Looking at the parameters of the pivot table function, we can see that the index are users and the columns are movies. We can also see that the rating values are being aggregated by the first appearance of each userId and movieId pair. Each user can only rate a movie once, so the first instance of a userId and movieId pair is the only pair in the data. If a user did not give a movie a rating, then the value is filled with 0. The resulting data frame is shown below.

	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	193579	193581	193583	193585	193587	193609
1	4.0	0.0	4.0	0.0	0.0	4.0	0.0	0	0.0	0.0	...	0.0	0	0	0	0.0	0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	...	0.0	0	0	0	0.0	0	0.0	0.0	0.0	0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	...	0.0	0	0	0	0.0	0	0.0	0.0	0.0	0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	...	0.0	0	0	0	0.0	0	0.0	0.0	0.0	0
5	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	...	0.0	0	0	0	0.0	0	0.0	0.0	0.0	0
...
606	2.5	0.0	0.0	0.0	0.0	0.0	2.5	0	0.0	0.0	...	0.0	0	0	0	0.0	0	0.0	0.0	0.0	0
607	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	...	0.0	0	0	0	0.0	0	0.0	0.0	0.0	0
608	2.5	2.0	2.0	0.0	0.0	0.0	0.0	0	0.0	4.0	...	0.0	0	0	0	0.0	0	0.0	0.0	0.0	0
609	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	4.0	...	0.0	0	0	0	0.0	0	0.0	0.0	0.0	0
610	5.0	0.0	0.0	0.0	0.0	5.0	0.0	0	0.0	0.0	...	0.0	0	0	0	0.0	0	0.0	0.0	0.0	0

Figure 8

Since the recommender systems that were tested used unrary data, I replaced each rating that was greater than or equal to a rating of 3.5 with the value of 1. If the rating was less than 3.5 then the value was replaced with 0. The movies that were not rated by a user kept the value 0.

7.2 Evaluation Metrics

Many recommender systems use numerical ratings from users and those recommender systems are used to predict what a user numerical rating a user would rate a movie. To evaluate these systems we can use root-mean-squared-error or absolute-mean-squared-error. Since my application will only give recommendations to movies based on what a user liked, I used precision, recall, and F1-score to evaluate each algorithm.

To understand precision, recall, and F1-score, we first need to know the differences between true positive, false positive, true negative, and false negative. For the recommender systems I tested, true positive is when the model predicts a movie a user liked. False positive is when the model predicts a movie that the user will like, but the user did not give any information on liking that movie. True negative is when a user did not give any information on liking a movie and the model does not recommend the movie to them. False negative is when a model does not recommend a movie, but the user did like the movie.

The equations for precision, recall, and F1-score are given by equations 1, 2, and 3 respectively. Precision is a metric that is the ratio of the recommended items a user likes with the total number of items recommended to a user. It is also the probability that a recommended item is consistent with what the user prefers (Schroder, Thiele, Lehner).

Recall is the ratio of the recommended items that the user liked with the total number of items that the user likes. It is the probability that an item a user likes is recommended (Schroder, Thiele, Lehner).

F1-Score is a metric that combines precision and recall into one equation. It does this by calculating the harmonic mean of precision and recall (Schroder, Thiele, Lehner).

$$(1) \textit{precision} = \frac{TP}{TP + FP}$$

$$(2) \textit{Recall} = \frac{TP}{TP + FN}$$

$$(3) F_1 \textit{ Score} = \frac{2}{\frac{1}{\textit{precision}} + \frac{1}{\textit{recall}}} = \frac{2 \cdot \textit{precision} \cdot \textit{recall}}{\textit{precision} + \textit{recall}}$$

7.3 Group Bayesian Personalized Ranking

7.3.1 Overview

One of the recommender systems I tested came from a paper titled “Group Bayesian personalized ranking with rich interactions for one-class collaborative filtering”. For this recommender system, the goal is to find parameters U_u , V_i , and b_i . U_u is the feature vector for user u and V_i is the feature vector for item i . Both of these vectors have the same number of elements. The parameter b_i is the bias for item i . The prediction for how much user u prefers item i is given by equation 4 (Pan, chen).

$$(4) \hat{r}_{ui} = U_u V_i^T + b_i$$

7.3.2 Algorithm

The objective function is given by equation 5. In this equation, \mathcal{G} is a user group and S is an item group. The α_v , α_u , and β_v are constants. The value of $\hat{r}_{Gui;uj}$ is given by equation 8. Equation 8 uses equation 7 which uses equations 4 and 6. The ρ in equation 7 is a constant (Pan, chen).

$$(5) f(\mathcal{G}, u, i, S) = \frac{1}{|S|} \sum_{j \in S} \ln[1 + \exp(-\hat{r}_{Gui;uj})] + \frac{\alpha_u}{2} \sum_{w \in \mathcal{G}} \|U_w\|^2 + \frac{\alpha_v}{2} \|V_i\|^2 + \frac{\alpha_v}{2} \sum_{j \in S} \|V_j\|^2 \\ + \frac{\beta_v}{2} \|b_i\|^2 + \frac{\beta_v}{2} \sum_{j \in S} \|b_j\|^2$$

$$(6) \hat{r}_{Gi} = \frac{1}{|\mathcal{G}|} \sum_{w \in \mathcal{G}} \hat{r}_{wi}$$

$$(7) \hat{r}_{Gui} = \rho \hat{r}_{Gi} + (1 - \rho) \hat{r}_{ui}$$

$$(8) \hat{r}_{Gui;uj} = \hat{r}_{Gui} - \hat{r}_{uj}$$

To find the parameters U_u , V_i , and b_i for each user u and item i , we need to get the gradient of equation 5 with respect to each of the parameters. Equation 9 is the derivative of the objective function with respect to $\hat{r}_{Gui;uj}$ and this equation will be part of every gradient.

The gradients with respect to each parameter is given by equations 10 to 15 and the parameters are updated using equation 16 (Pan, chen).

$$(9) \frac{\delta f(\mathcal{G}, u, i, S)}{\delta \hat{r}_{\mathcal{G}ui;uj}} = - \frac{1}{|S|} \frac{\exp(-\hat{r}_{\mathcal{G}ui;uj})}{1 + \exp(-\hat{r}_{\mathcal{G}ui;uj})}$$

$$(10) \quad \nabla U_w = \sum_{j \in S} \frac{\delta f(\mathcal{G}, u, i, S)}{\delta \hat{r}_{\mathcal{G}ui;uj}} [(1 - \rho)V_i + \rho \frac{V_i}{|\mathcal{G}|} - V_j] + \alpha_u U_w, \quad \text{if } u=w$$

$$(11) \quad \nabla U_w = \sum_{j \in S} \frac{\delta f(\mathcal{G}, u, i, S)}{\delta \hat{r}_{\mathcal{G}ui;uj}} \rho \frac{V_i}{|\mathcal{G}|} + \alpha_u \frac{U_w}{|\mathcal{G}|}, \quad \text{if } u \neq w$$

$$(12) \quad \nabla V_i = \sum_{j \in S} \frac{\delta f(\mathcal{G}, u, i, S)}{\delta \hat{r}_{\mathcal{G}ui;uj}} [(1 - \rho)U_u + \rho \bar{U}_{\mathcal{G}}] + \alpha_v V_i$$

$$(13) \quad \nabla V_j = \frac{\delta f(\mathcal{G}, u, i, S)}{\delta \hat{r}_{\mathcal{G}ui;uj}} (-U_u) + \alpha_v V_j, \quad j \in S$$

$$(14) \quad \nabla b_i = \sum_{j \in S} \frac{\delta f(\mathcal{G}, u, i, S)}{\delta \hat{r}_{\mathcal{G}ui;uj}} 1 + \beta_v b_i$$

$$(15) \quad \nabla b_j = \sum_{j \in S} \frac{\delta f(\mathcal{G}, u, i, S)}{\delta \hat{r}_{\mathcal{G}ui;uj}} (-1) + \beta_v b_j, \quad j \in S$$

$$(16) \quad \theta = \theta - \gamma \frac{\delta f(\mathcal{G}, u, i, S)}{\delta \theta}, \quad \theta \text{ can be } U_w, V_i, V_j, b_i, \text{ or } b_j \text{ and the learning rate is } \gamma > 0.$$

The algorithm for Group Bayesian Personalized Ranking takes training data as input. It needs the constants ρ , α_v , α_u , and β_v as inputs and the number of latent features, this is the size of the vectors V_i and U_u . We also need to input the number of iterations and the learning rate. The final inputs needed is the size of item set S and the size of user group \mathcal{G} . The output will be V_i , U_u and b_i for each user u in the training data and item i in the training data. The complete algorithm is shown below (Pan, chen).

- Loop t1 from 1 to number of iterations
 - Loop t2 from 1 to number of users
 1. Select a random user u from the training set
 2. Select a random item i from the items the user liked
 3. Select a random item set S from the items in the training data that user u did not like.

4. Form the user group \mathcal{G} by selecting $|\mathcal{G}| - 1$ random users from the set of users that liked item i , not including user u .
5. Find $\frac{\delta f(\mathcal{G}, u, i, S)}{\delta \bar{r}_{gui;uj}}$ and $\bar{U}_{\mathcal{G}}$
6. Update U_w for $w \in \mathcal{G}$
7. Update V_i
8. Update V_j for $j \in S$
9. Update b_i
10. Update b_j for $j \in S$

Algorithm 1

7.3.3 Implementation

I implemented this algorithm by creating a Python function. One of the inputs was a ratings matrix that was a two-dimensional numpy array. The rows of this array were users and the columns were movies. Each value in this array was 1 if the user liked the movie and 0 if the user did not like the movie or if the user did not rate the movie. The function also contained parameters for item group size and user group size. Another parameter is rho, which refers to ρ in equations 10, 11, and 12. There is also a tuple with three numbers that corresponds to α_v , α_u , and β_v respectively. Finally there are parameters for learning rate and the number of iterations.

The first thing the function did was it randomly initialized two two-dimensional arrays and one one-dimensional array. The two-dimensional arrays are V and U . Each column in V represents vector V_i for item i . Each column in U represents vector U_u for user u . The number of columns in V is the number of movies in the ratings dataset and for U it is the number of users. Both of them have the same number of rows, which is the number of latent features. The one-dimensional array is b . Each element in the array represents b_i for item i . The length of this array is the number of movies.

After that, I got the movies each user liked and placed the movie's index in a list. Once the list contained every movie the user liked, then that list was appended to another list. In the second list, the index u contains the indices of movie user u liked. The same thing is done for

movies each user did not like. This was done for faster retrieval of movies a user liked or did not like.

On step 4 of this algorithm, we have to find users that also liked a movie. For faster retrieval of these users, I created a list in which index i of the list contains the indices of users that liked movie i .

Rather than having two loops, I ran steps 1 to 10 of the algorithm by the multiple of the number of iterations and the number of users. To get a random user, a random number from 0 to the number of users minus 1 is selected. After that, a random item the user liked is selected.. Steps 3 to 10 are done only if the number of people who liked the item is at least the same as the item group size and if the number of movies the random user did not like is at least equal to the item set size. If this criteria is met, then the group set and item set are formed. I then found \bar{U}_g , which the average U_u for each user in the user group. Then I calculate equation 7 and use this value to find $\frac{\delta f(G, u, i, S)}{\delta \hat{r}_{ui;uj}}$ for each item in the item set. After this the gradients from equations 10 to 15 are found and then each user vector from U and item vectors from V are updated. Each bias from the item set is also updated.

The function returns the arrays U , V , and b . To get a matrix that shows the prediction of the users' preference, U was multiplied by the transpose of V and b was added to each of the rows. If user u liked movie m from the training set, then the value in the matrix at row u and column m is replaced with the lowest value in the matrix minus one. This is done to get new movie recommendations.

7.3.4 Testing and Results

To test this algorithm, I first took the original rating matrix and replaced 30 percent of the likes with the value of 0. I used this as the training set. I tested different values for ρ and found that $\rho=1$ performed the best at various iterations and number of latent features. I also tried different iterations and found that 75 performed about as good as 100 and it did better than 50 and 150, so I fixed iterations at 75 for all other experiments. I test for α_v , α_u , and β_v with 0.1, 0.01, 0.001, and 0.0001 and found that 0.001 performed the best for each constant. Each user was recommended the top 10 movies with the highest preference scores.

The first test I did was to try different values for the number of latent features. I test the number of latent features at 2, 3, 5, 7, 10, 15, 20, and 25. The group size and item set size was fixed at 3. Figure 9 shows the results for this test based on precision score. I also got the scores for recall and F1-score, but the graphs for those metrics looked the same except the values on the y-axis were smaller. From the graph, we can see that the best performance happened at a latent feature size of 2. The precision at this point was 0.1393. Between 3 and 15 the precision was almost the same. It ranged from 0.1282 to 0.1325. Finally, the smallest two precision scores happened at the end.

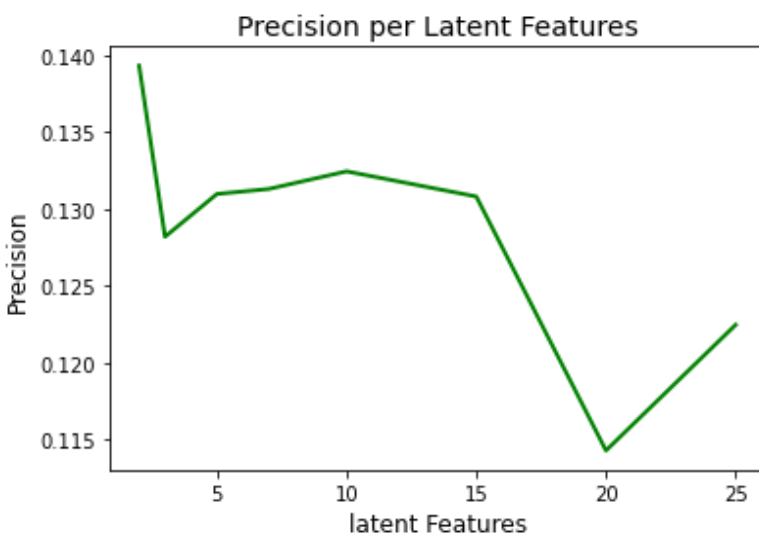


Figure 9

The next test I did was testing different values for group size and item set size. I kept the number of latent features fixed at 2 and tested the set sizes at 2, 3, 4, 6, 8, 10, 12, 14, and 16. At each test I kept the group size and item set size equal to each other. Figure 10 shows the precision score at each test. Based on the graph, there seems to be no correlation between how well the algorithm performed with the set size.

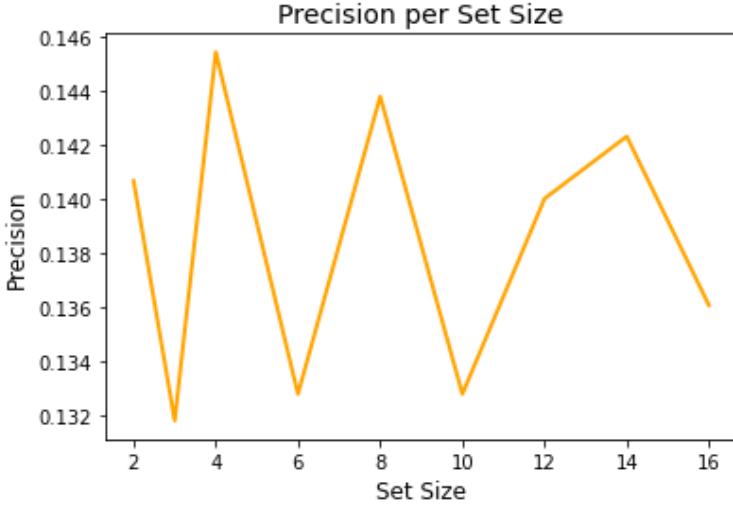


Figure 10

7.4 LREC

7.4.1 Overview

For the LREC algorithm, we need a rating matrix $R \in \mathbb{R}^{n \times m}$, where n is the number of users and m is the number of items. Each element in R is either 0 or 1. We first find $X = R^T$ and $Y = 2R - 1$. Once we find these values, we can find $W \in \mathbb{R}^{m \times m}$ by using the objective function in equation 17. In this equation, U is the set of users and I is the set of items. Y_{ui} is the element in Y at row u and column i . X_i is column i on X and W_u is row u on W . $L(Y_{ui}, X_i W_u)$ is a loss function. Possible choices for this function are logistic loss and squared loss. Once W is found, we can get the recommendation matrix using equation 18 (Sedhain, Menon, Sanner, Brazian).

$$(17) \min_W \sum_{u \in U} \sum_{i \in I} L(Y_{ui}, X_i W_u) + \frac{\lambda}{2} \|W\|_F^2$$

$$(18) \hat{R} = W^T R$$

7.4.2 Implementation

To implement LREC, I created a Python function that takes in the parameters for the ratings matrix, inverse regularization term, and a list of users to evaluate. The default inverse regularization term is set to 0.01 and the default for the list of users is none. If a list of users is not given then, the list is set to all users in the data.

After getting the user list, X and Y are calculated. The authors of this paper had the code on Github, I did not use their code, but I did look at it. One thing I noticed was that to use logistic loss for equation 17, they used logistic regression from Sklearn. To get the row of u of W , they trained logistic regression on X and column u of Y and then used the coefficients of the trained model for the row of W . The logistic regression class from Sklearn has a parameter C , which is the inverse regularization term. The term C was set to the parameter for the inverse regularization term of my function.

After finding W , equation 18 is used to calculate the recommendation matrix. After that, the minimum value of this matrix is found. If a user likes item i from the rating matrix, then the element at row i and column u in the recommendation matrix is replaced with the minimum value minus 1. After updating the recommendation matrix, the LREC function returns it and it can be used to make new recommendations for users.

I also created a function that uses the squared loss function equation 17. This function is similar to the one that used logistic loss. The difference is that the ridge class from Sklearn was used instead of logistic regression. Ridge has a parameter called alpha which is the strength of regularization. Because of this, a new parameter for the regularization constant is added and the inverse regularization term is removed.

7.4.3 Testing and Results

I first tested this recommender system using logistic loss for equation 17. I used 30 percent of the data for testing and the rest for training. I tested the values for inverse regularization terms at 0.1, 0.075, 0.05, 0.025, 0.01, 0.0075, 0.005, 0.0025, and 0.0001. The top 15 values for each user from the recommendation matrix were the movies recommended to them. Figure 11 shows the precision at each term. From the graph we can see that the best performance occurred at 0.025. At this point the value of recall and precision was also the highest. The precision was 0.2255, recall was 0.1114, and the F1-score was 0.1491.

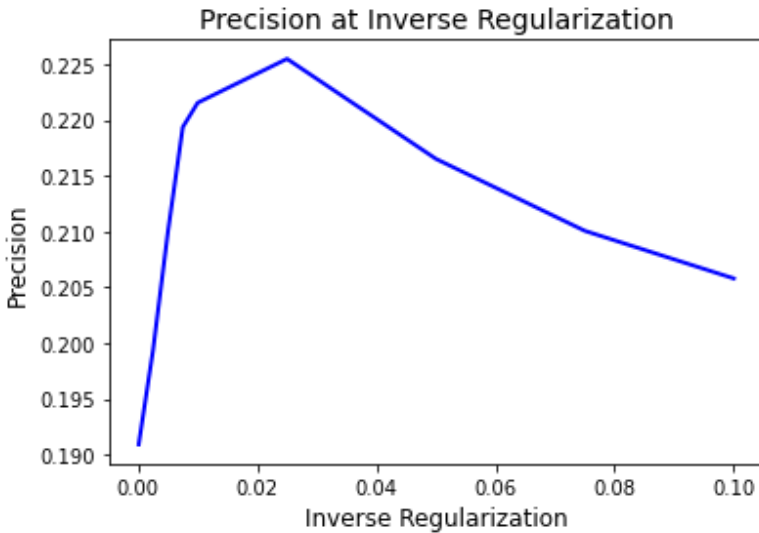


Figure 11

I also tested this recommender system using squared loss. The test was done using the same training and testing set. I test it with regularization constants of 50, 100, 150, 200, 250, 300, 350, 400, 450, and 500. The best performance for precision, recall, and F1-score was with a regularization constant of 250. The precision, recall, and F1-score was 0.2502, 0.1236, and 0.1654 respectively.

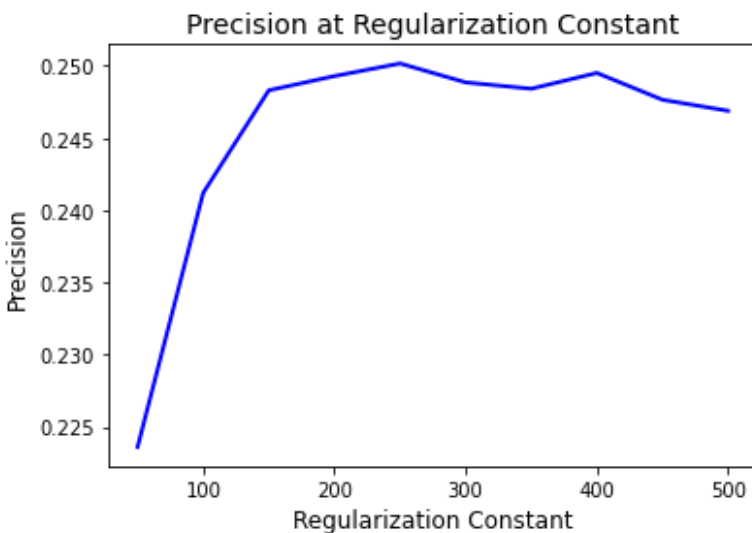


Figure 12

7.5 TSUISIMCF

7.5.1 Overview

Another algorithm I tested is called two-steps combined user and item similarities in graph-based collaborative filtering (TSUISIMCF). This algorithm uses the rating matrix to find similarities between each pair of users and each pair of items and uses these similarities to recommend items to users. There are four similarity metrics given in this paper that could be applied to users and items. The recommender system uses similarity scores for users and similarity scores for items (Putra, Mahendra, Budi, Munajat).

7.5.2 Similarity Metrics

The similarity metrics used in this paper are common neighbors, Jaccard coefficient, Adamic-Adar, and preferential attachment. Each similarity metric uses the notation $\Gamma(x)$. If x is an item, then the term $\Gamma(x)$ is the set of users that liked that item. If x is a user, then $\Gamma(x)$ is the set of items that the user likes (Putra, Mahendra, Budi, Munajat).

Common neighbors are the number of overlapping elements in two sets. It is given by equation 19 (Putra, Mahendra, Budi, Munajat).

$$(19) \text{CN}(x, y) = |\Gamma(x) \cap \Gamma(y)|$$

Jaccard coefficient is the probability that two sets have overlapping elements. It gets this value by getting the number of overlapping elements and dividing it by the total number of unique elements in both sets. It is given by equation 20 (Putra, Mahendra, Budi, Munajat).

$$(20) \text{JC}(x, y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|} = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x)| + |\Gamma(y)| - |\Gamma(x) \cap \Gamma(y)|}$$

Adam-Adar is an improvement to common neighbors because it weighs less popular items more and more popular items are weighed less. The idea is that it reduces the bias from popular elements. It is given by equation 21 (Putra, Mahendra, Budi, Munajat).

$$(21) \text{AA}(x, y) = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{|\Gamma(z)|}$$

Preferential Attachment is the size of the size of each set multiplied by each other. The equation for it is given by equation 22 (Putra, Mahendra, Budi, Munajat).

$$(22) PA(x,y) = |\Gamma(x)| \cdot |\Gamma(y)|$$

7.5.3 Algorithm

The input for TSUISIMCF uses is the rating matrix, similarity measures between users, similarity measures between items, n number of items, and k number of users. It returns n item recommendations to a user. The algorithm for recommending items to a user u from the rating matrix is given below (Putra, Mahendra, Budi, Munajat).

1. Find Items user u already liked.
 - For each item i that user u liked:
 - Place item i in a list of liked items.
2. Find the k most similar users to user u that does not include u and place them in a list of similar users.
3. Find possible item recommendations.
 - For each user s in similar users:
 - For each item user s liked:
 - If item not in liked items:
 - Place item in a list of possible recommendations
4. Set a score for each possible item recommendations
 - For each item p in possible recommendations:
 - Set the score of p to $\max \text{score}(p, r)$, where r is an item in the list of liked items.
5. Find the top n scores and recommend these items to user u .

Algorithm 2

7.5.4 Testing and Results

I test this algorithm by removing 30 percent of the ratings from the original dataset and using it as a test set. I was not able to implement Adamic-Adar with a reasonable runtime, so I did not test it. I was able to implement the other measurements and I applied them with users and movies. For every test, the k number of users was set to 5 and n number of items was set to 15. Figure 13 shows the results based on precision score from different similarity measurements for users and movies. In this graph x-axis is the similarity measure applied to movies and the color of the bars represent the similarity measure used on users. For example, with Jaccard coefficient at the x-axis, the green bar represents the precision of TSUISIMCF using Jaccard coefficient for movie similarity and preferential attachment for users.

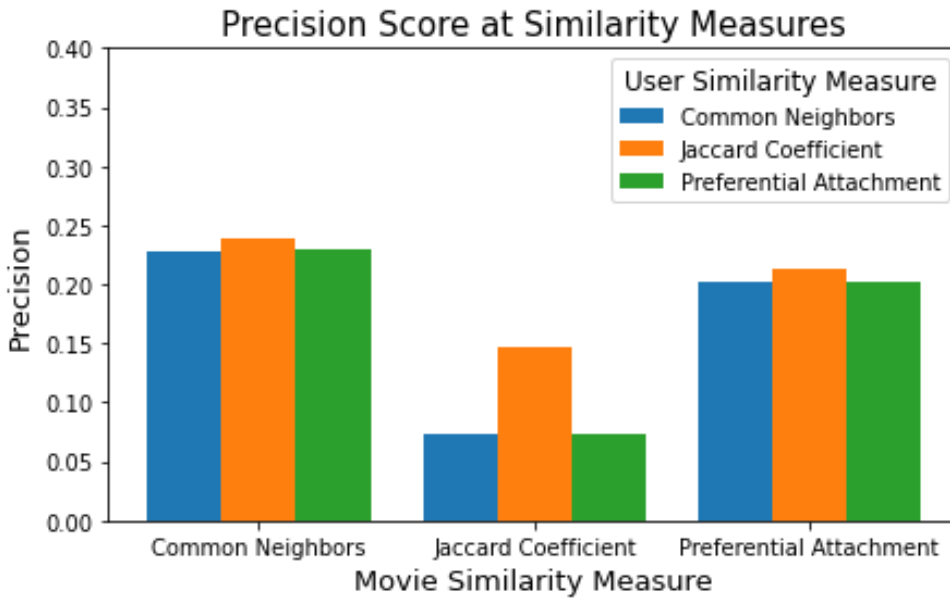


Figure 13

From the graph we can see that the best performance came from using common neighbors on movies with each user similarity. We can also see that Jaccard coefficient applied to users performed the best at each movie similarity. The best performance came from using common neighbors with movies and Jaccard coefficient with users. The precision, recall, and F1-score was 0.239016, 0.039373, and 0.067610 respectively.

7.6 Genre Correlation

7.6.1 Overview

Genre Correlation is an algorithm that I learned from a paper titled “Content-Based Movie Recommendation System Using Genre Correlation.” This algorithm is different from the other algorithms I tested because it is a content-based recommender system and the other ones were collaborative filtering approaches. This algorithm uses the genres of each movie and It uses like and dislike ratings for movies (Putra, Mahendra, Budi, Munajat). This is different from the algorithms I am trying to test because I am only using movies users like. I made changes to the original algorithm for it to work on a rating matrix with only user likes.

7.6.2 Algorithm

The original algorithm uses a rating matrix in which the columns are movies and the rows are users. The values of the rating matrix is 1 if the user liked the movie, -1 if they did not like it, and 0 if they did not rate this movie. It also requires a list of genres for each movie. The algorithm is given below (Putra, Mahendra, Budi, Munajat).

1. Create a genre matrix in which rows are movies and columns are genres. If a movie can be classified as genre the value of that element is 1 and 0 if it cannot.
2. Get the results matrix by multiplying the rating matrix with the genre matrix.
3. Change negative values from the results matrix to 0 and positive values to 1.
4. Calculate the Euclidean distance between each row of the results matrix and column of the rating matrix.
5. Movies with the smallest distances are recommended to the user.

Algorithm 3

To use this algorithm with data that contains only the movies users like, I had to change step 4. The issue is that in step 3 we are multiplying two matrices and both matrices have only nonnegative numbers, so the resulting matrix will only contain nonnegative numbers. I changed step 4 by making it so that if an element in the results matrix is below a certain value then the value is changed to 0. If it is not then the value is 1.

7.6.3 Testing and Results

To test this algorithm, I tested different thresholds for step 4 of algorithm 3. The values I tested were 1 to 35 in increments of 1. I recommended each user the lowest 15 distances. The result based on the precision score is shown in figure 14. The best results came from a threshold value of 5. The precision score was 0.008415, recall was 0.004159, and the F1-score was 0.005567. These values were a lot lower than the collaborative filtering approaches. For example the best precision score from TSUISIMCF was 0.239016. This is over 28 times higher than the best performance from this algorithm.

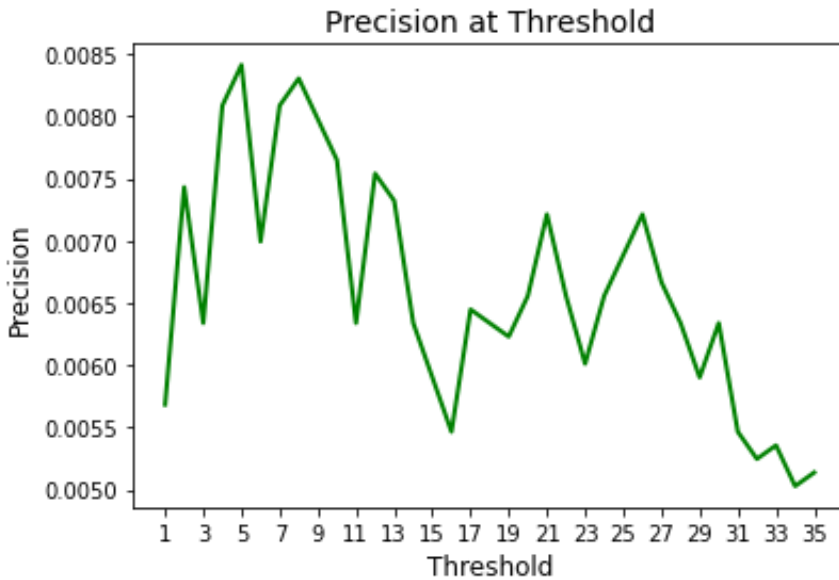


Figure 14

7.7 Term Frequency-Inverse Document Frequency

7.7.1 Overview

Term Frequency-Inverse Document Frequency (TF-IDF) is a measurement used to weigh different terms in a document. To find the TF-IDF of term i in document k , we first have to find the frequency of term i in document k . This is given by equation 23. In this equation, $f_{i,k}$ is the number of times term i appears in document k (Adomavicius, Tuzhilin).

After finding term frequency, we need to find the inverse document frequency of term i . The equation for this is given by equation 24. In this equation, D is the set of all documents and D_i is the set of documents that contain term i . The weight of term i in document k is given by equation 25 (Adomavicius, Tuzhilin).

To use TF-IDF to compare the similarity between documents, each document is given a vector and each element in this vector is the weight for a specific term. To find the similarity between documents we use cosine similarity, which is given by equation 26 (Adomavicius, Tuzhilin). In this equation, v_r is the vector of weights for document r and v_s is the vector for document s .

$$(23) \quad TF_{i,k} = \frac{f_{i,k}}{\sum_{j \in d_j} f_{j,k}}$$

$$(24) \quad IDF_i = \log\left(\frac{|D|}{|D_i|}\right)$$

$$(25) \quad w_{i,k} = TF_{i,k} \times IDF_i$$

$$(26) \quad \cos(v_r, v_s) = \frac{v_r \cdot v_s}{\|v_r\| \times \|v_s\|}$$

A way to apply TF-IDF to movies is by using movies as documents and genres as terms. Each movie will have a vector in which each element of the vector is the TF-IDF weight of a genre. Equation 26 can be used to find how similar two movies are.

7.7.2 Algorithm

To recommend movies to users, I decided to score each movie for a user by using the algorithm below. For this algorithm, we need the rating matrix and the vectors of TF-IDF weights for each movie. Weights came from using genres as terms.

1. Get similarity matrix in where the element of row i and column j is the cosine similarity between movie i and movie j
2. Get the list of movies the user likes

3. From the similarity matrix get columns of the movies the user likes.
4. Get the mean of these columns by summing each of them together and then dividing each element by the number of movies the user liked. This will be the score vector.
5. Update the score vector by replacing each movie the user already likes with 0.
6. Recommend the top k highest scoring movies to the user.

Algorithm 4

7.7.3 Testing and Results

To test this algorithm I randomly removed 30 percent of the ratings and used it as a test set and used the remaining as a training set. I recommend the top 15 highest scoring movies to each user. The results for precision, recall, and F1-score were 0.010710, 0.005293, and 0.007085, respectively. Each of these scores were higher than the scores of genre correlation, but they were still much lower than all of the collaborative filtering approaches.

7.8 TSUISIMCF and TF-IDF

7.8.1 Overview

After testing different collaborative filtering and content-based recommender systems, I decided to try hybrid recommender systems. There are many classes of hybrid recommender systems and one of them is weighted. In this class, we get the sum of weighted scores from different recommender systems (Thorat, Goudar, Barve). I tested this with TSUISIMCF and TF-IDF. For TSUISIMCF, the similarity measure I used for movies was common neighbors and for users I used Jaccard coefficient. To get the weighted sum of these recommender systems, I kept the weight for TF-IDF as 1 and I tested different weights for TSUISIMCF.

7.8.2 Testing and Results

I tested this recommender system by using 0.0025, 0.005, 0.0075, 0.01, 0.025, 0.05, 0.075, and 0.1 for weights for TSUISIMCF. Each user was recommended the top 15 highest scoring movies for them. The best performance came from a weight of 0.01. The scores for precision, recall, and F1-score, were 0.26951, 0.04440, and 0.07623 respectively. These scores

were higher than scores of TSUISIMCF without adding TF-IDF. Figure 15 shows the precision at each weight.

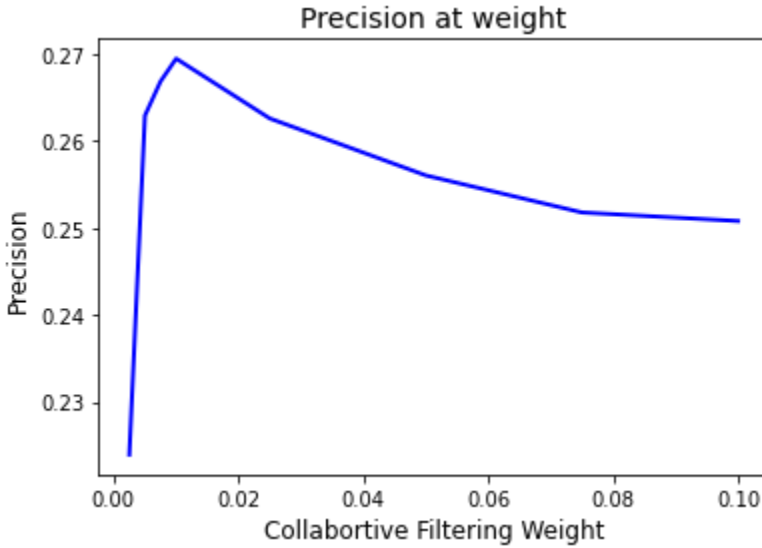


Figure 15

7.9 LREC and TF-IDF

7.9.1 Overview

Another hybrid recommender system I tested was getting the weighted sum of LREC and TF-IDF. For LREC, I used squared loss for the loss function and used the best performing regularization constant, which was 250. I kept the weight of TF-IDF at 1 and I tested different weights for LREC.

7.9.2 Testing and Results

I tested this recommender system with the weights of the LREC scores ranging from 1 to 10 in increments of 1. Each user was recommended the top 15 scoring movies. Figure 16 shows the precision at each weight. Overall, the best performance came from a weight of 6. The precision, recall, and F1-Scores were 0.25290, 0.12498, and 0.16729, respectively.

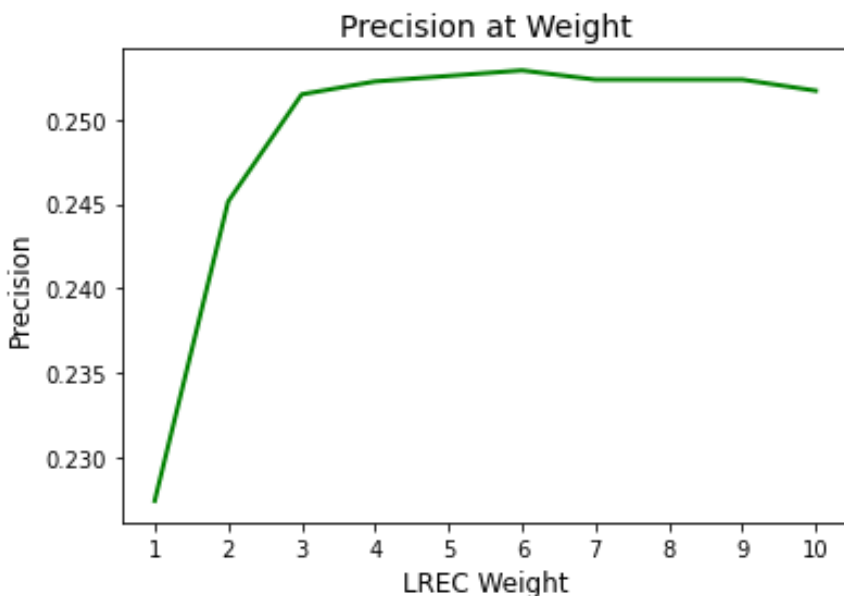


Figure 16

7.10 Overall Results

In this section I will compare the results of each recommender system based on precision, recall, and F1-score. Since, both of the content-based recommender systems performed poorly, they will not be included in any comparisons. The best parameters with the best performance within each model were used.

The first metric I will look at is precision. The results based on precision from each model can be seen in figure 17. The best precision score came from the hybrid model that uses TSUISIMCF and TF-IDF. The precision for this model was 0.26951. From the graph, the two models that use LREC look roughly the same, but LREC and TF-IDF scored slightly more. LREC and TF-IDF had a precision of 0.25290 and LREC had a precision of 0.25016.

Figure 18 shows the performance of each model based on recall. From the graph we can see that the models that used LREC performed much better than the other recommender systems. LREC and TF-IDF performed slightly better with a recall score of 0.12498 and LREC had a score of 0.12363.

Figure 19 shows the performance of each model based on its F1-score. This graph is similar to the graph for recall scores based on how each model compares to each other. The

models that used LREC had a much higher F1-score compared to the other models and LREC and TF-IDF had a slightly higher F1-score than LREC.

The recommender system I ended up implementing to the application was LREC. Although LREC and TF-IDF performed better, by the time I started testing it, I had already implemented LREC to the application. Since the performance with LREC and TF-IDF was only slightly better I did not want to spend the time implementing a new model for the application.

Even Though TSUISIMCF and TF-IDF performed better on precision compared LREC, I still choose LREC because it performed better on the other two metrics. LREC also ran much faster than TSUISIMCF and TF-IDF. I timed how long each model would take with one user and TSUISIMCF and TF-IDF would take nearly 4 seconds and LREC took slightly more than 0.1 seconds to run.

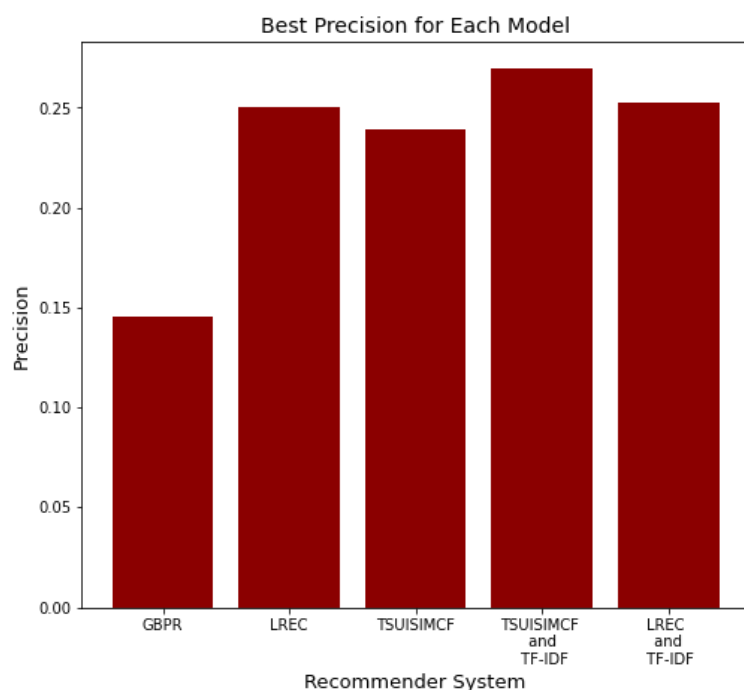


Figure 17

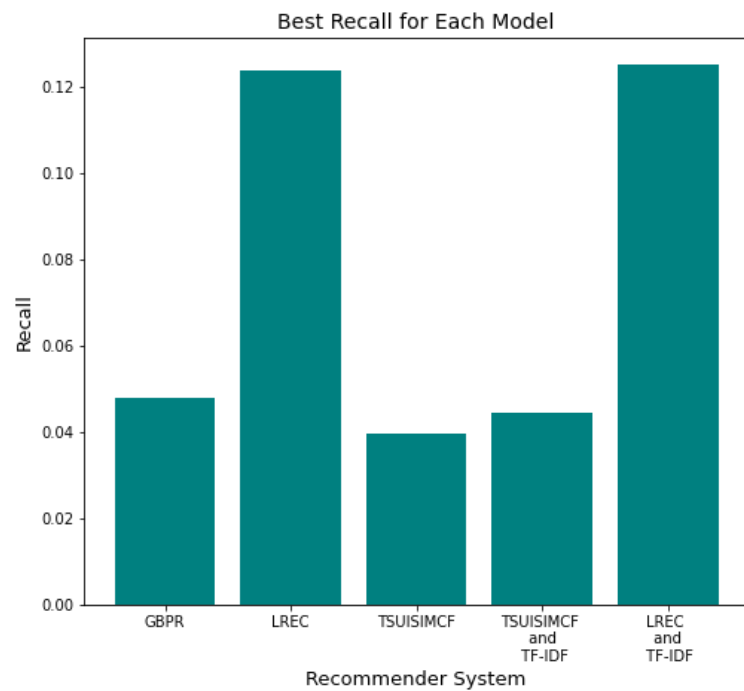


Figure 18

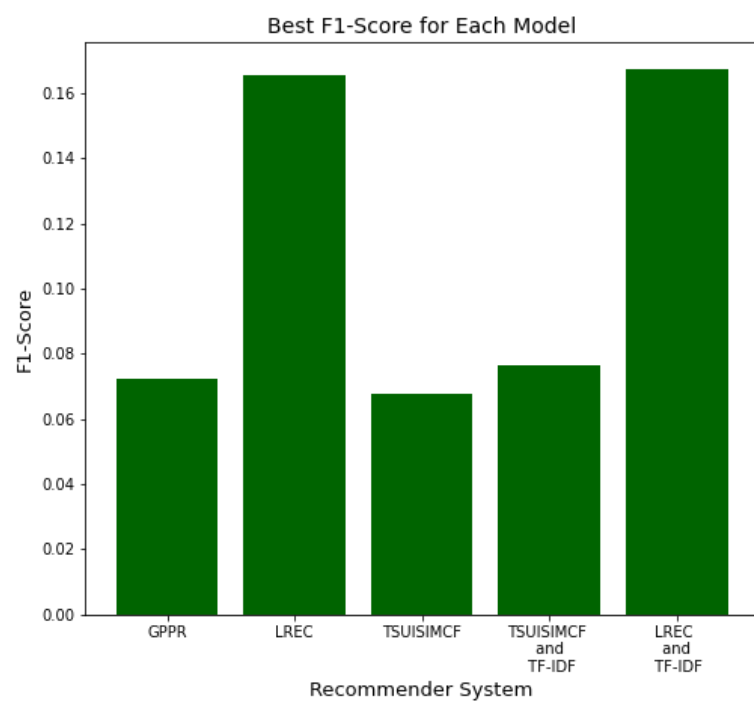


Figure 19

8 User Interface

When a user enters the web application, they will see a screen that looks like figure 20. In this screen they can start using the application.

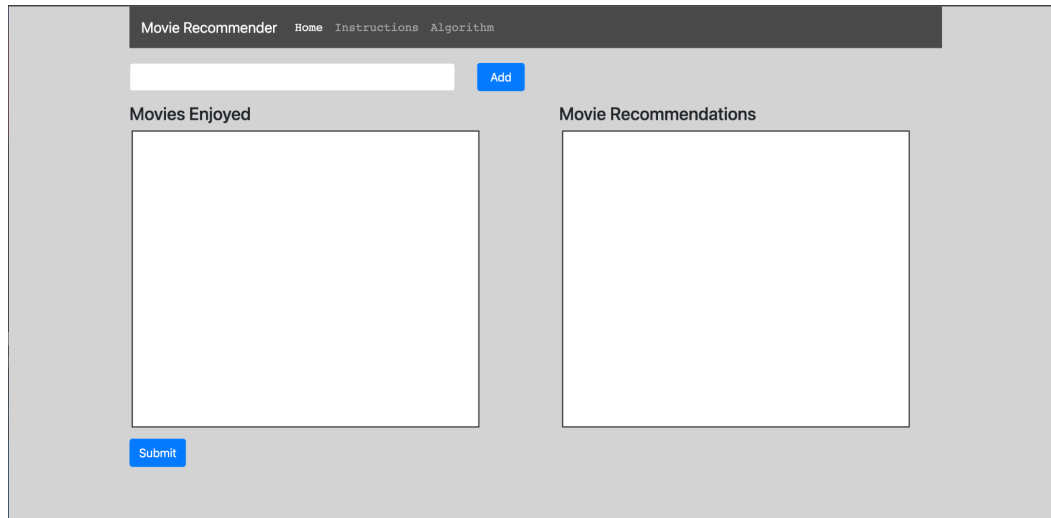


Figure 20

Users can start searching for movies to add to the list of movies they enjoyed by using the text bar towards the top left. As a user types there will be a drop-down menu with suggestions. These suggestions will contain the text entered in the text bar. Figure 21 shows how it will look with the term “spider.”

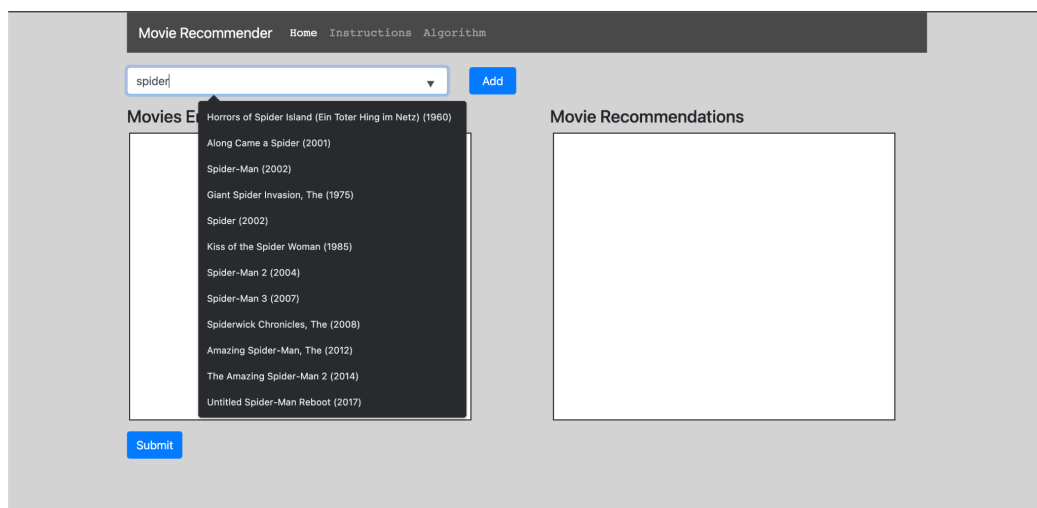


Figure 21

When the user finishes adding movies to the list of movies they enjoyed, they can press submit and receive 25 movie recommendations. Figure 22 shows how the screen will look when the user receives movie recommendations.

In figure 22, we can see that the second to last movie on movies enjoyed has some of its characters replaced with three dots. This was done because I wanted every title to fit in a single line and because of this I shortened titles with more than 35 characters. I did the same thing with movie recommendations except they were shortened only when there were more than 45 characters in the title.

The red button next to each title in the list of enjoyed movies allows the user to remove it from the list. If a user clicks on a title from the movie recommendation list, it will open up a new tab to the movie's IMDB page.

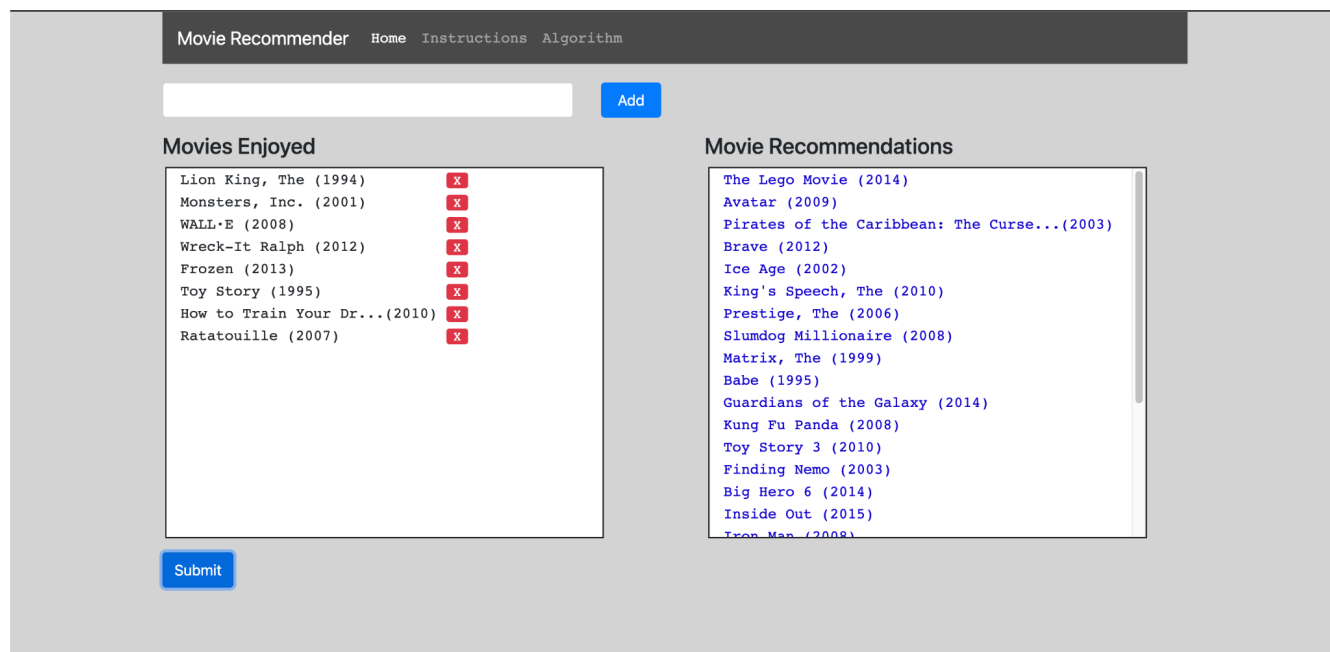


Figure 22

Figure 23 shows the instructions page of the application. To go on this page, the user has to click on "Instructions" on the top of the screen. In this page, the user can learn how to use the application.

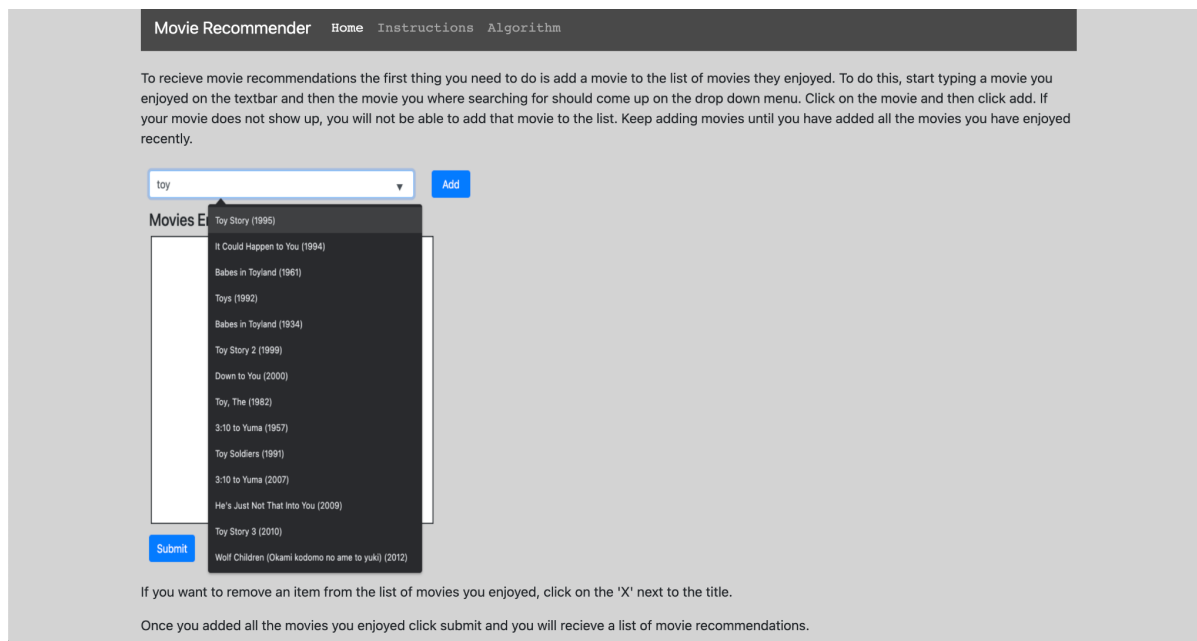


Figure 23

If a user clicks on “Algorithm,” they will be sent to a page that looks like figure 24. In this page a user can learn about the dataset used for the recommender system and they can learn about how the recommender system works.

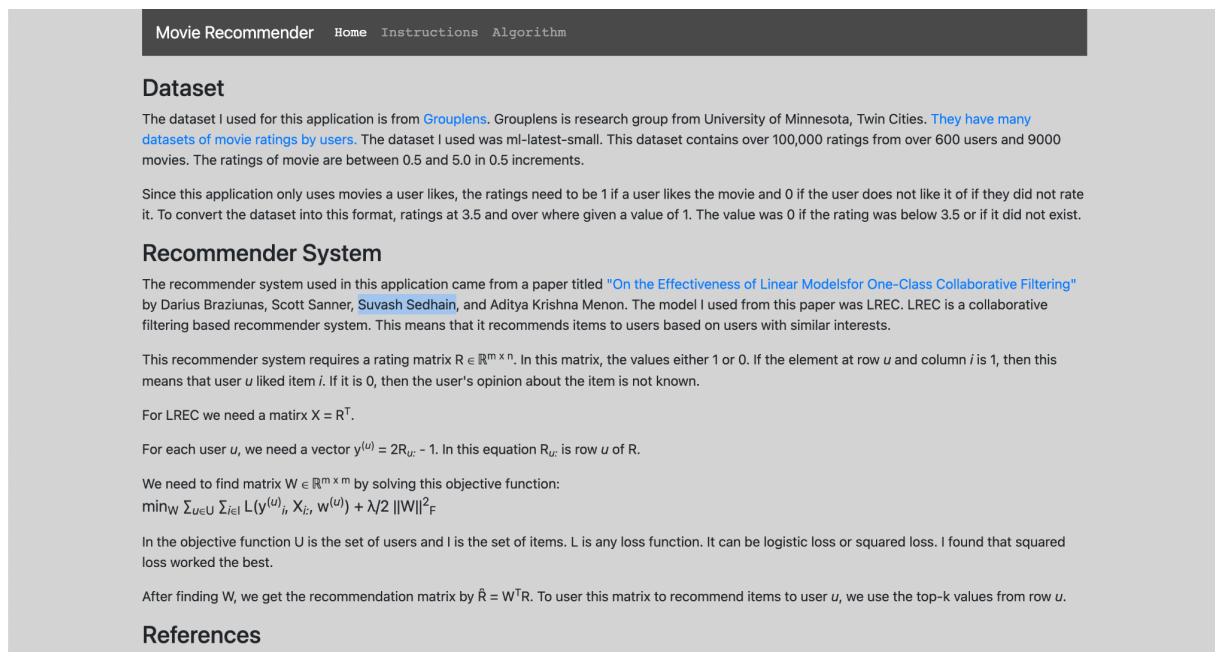


Figure 24

9 Possible Improvements

One possible improvement would be to use a different dataset for the recommender system. One issue of this dataset is that there is not a lot of data in it. There are almost 10,000 movies, but there are only 100,000 ratings. This means that the average number of ratings for each movie is 10. After converting the dataset to 1 and 0, some movies end up with no ratings. This happens because many movies have a small number of ratings and some of those movies do not have any ratings above 3.5. Another issue with this dataset is that it was released in 2018. This means that movies that were released after 2018 are not in the dataset.

Another improvement can be giving user's the ability to create an account and then they can use the account to save all the movies they like. This would allow them to update the list of movies they enjoyed and constantly receive a new list of movie recommendations as they update their list. This would also let them add far more movies, because they can keep adding movies for as long as they wanted.

Having an account could also allow for different rating formats. This includes likes and dislikes and numerical ratings. Without an account, users do not have a lot of time to rate movies. Different rating formats would give more information about the user and this could lead to better movie recommendations.

Another way to improve this application would be to allow users to filter movies by different genres. This could be useful because people may have different desires at different moments. For example, a person may want to watch a comedy movie over a movie that may have stressful moments.

Filtering movies by streaming service may also be an improvement. Many people have different streaming services such as Netflix and Hulu. Those people may want to use those services to watch a movie and not spend extra money. This would make it more convenient for users because they will not have to spend time by going through all of their streaming services to see if a movie is on it.

A possible improvement would be to have different recommender systems based on the movies given by the user. If the user gives a small list of movies that are similar in some way, then this may mean that the user wants to watch other similar movies. For example if the user gave a list of animated movies, then we could assume that they are interested in only watching animated movies. In cases like these we would use a content-based recommender system. If the

user gave a long list of movies that are not similar, then this could mean that the user is looking for a wide variety of recommendations. A collaborative filtering approach may be best in this case, because it would recommend movies based on other users that had similar interests as the original user. Since this approach does not look at the movie's content, it would recommend a large variety of movies.

Finally, I believe that the best metric to test a recommender system would be user feedback. I feel that the main goal of a recommender system should be how useful it is for a user and to know this we would need user feedback. One way to receive feedback would be to survey users on their experience using the recommender system. We could ask them questions like how they felt about the recommendations they received and how many movies did they watch because of the recommender system.

References

Burke, Robin. (2002). Hybrid Recommender Systems: Survey and Experiments. User Modeling and User-Adapted Interaction. 12. 10.1023/A:1021240730564.

G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," in IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 6, pp. 734-749, June 2005, doi: 10.1109/TKDE.2005.99.

B. Sarwar, G. Karypis, J. Konstan and J. Riedl, "Item-based Collaborative Filtering Recommendation Algorithms", Proc. 10th International Conference on the World Wide Web, pp. 285--295, 2001.

Schröder, G., Thiele, M., and Lehner, W. (2011). "Setting Goals and Choosing Metrics for Recommender System Evaluations," in UCERSTI2 workshop at the 5th ACM conference on recommender systems (Chicago, USA) Vol. 23, 53.

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19. <https://doi.org/10.1145/2827872>

Schroder G, Thiele M, Lehner W (2011) Setting goals and choosing metrics for recommender system evaluations. In: Proceedings of UCERSTI2 workshop at the 5th ACM conference on recommender systems, vol 23, pp 78–85

W. Pan and L. Chen. Gbpr: Group preference based bayesian personalized ranking for one class collaborative filtering, In Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, pp.2691-2697, 2013

S. Sedhain, A. K. Menon, S. Sanner, D. Braziunas, "On the effectiveness of linear models for one-class collaborative filtering", in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, AAAI Press, 2016, pp. 229–235.

A. A. Putra, R. Mahendra, I. Budi, and Q. Munajat, "Two-steps graph-based collaborative filtering using user and item similarities: Case Study of e-commerce recommender systems," in 2017 International Conference on Data and Software Engineering (ICoDSE), 2017, pp.1–6.

S. Reddy, S. Nalluri, S. Kuniseti, S. Ashok, and B. Venkatesh, "Content-based movie recommendation system using genre correlation," in Smart Intelligent Computing and Applications, ed: Springer,(2019), pp. 391-397.

G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," in IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 6, pp. 734-749, June 2005, doi: 10.1109/TKDE.2005.99.

P. B. Thorat, R. M. Goudar and S. Barve, "Survey on collaborative filtering, content-based filtering and hybrid recommendation system," International Journal of Computer Applications, vol. 110, no. 4, pp. 31–36, 2015.