Final Report

**Project Title:** Exploring the Use of Recommender Systems

**Problem Statement**

Recommender systems frequently give users personalized suggestions based on their interests and activities and are widely used in different industries such as finance, health, and entertainment. Many challenges with data sparsity, cold start, scalability, and diversity affect the results of the recommender system and are common among many available approaches.

Using the MovieLens dataset, which contains movie ratings and metadata of movies, I aim to explore different methods for building a recommender system and create a hybrid recommender system using the combination of those methods.

In this project, my focus is on comparing different techniques individually and then combining them to have a hybrid(ensemble) approach to improve the overall results of a recommender system. The methods that I will explore are collaborative filtering, content-based, and matrix factorization to increase the accuracy of movie recommendations.

My recommender system's effectiveness will be assessed using measures such as root mean square error (RMSE) and hit rate. Although the hybrid approach is not unique, I think this will produce unique results and offer valuable information to movie fans.

**Data Source**

GroupLens Research has made several datasets available under the name MovieLens. These datasets include reviews and tag applications for movies that have been gathered from the MovieLens website (http://movielens.org) over a variety of time periods. The datasets are available in various sizes, enabling researchers to select the dataset that best suits their needs by offering various information on user behavior and preferences.

The MovieLens dataset used in this project contains 100,004 ratings and 1,296 tag applications across 9125 movies. These data were created by 671 users between January 09, 1995, and October 16, 2016. This dataset was generated on October 17, 2016.

MovieLens Small dataset contains several files, as follows (Harper, 2015):

links.csv: This file contains identifiers that can be used to link the MovieLens data with data from other sources. Each row represents one movie and contains the following columns: movieId, imdbId, and tmdbId.

movies.csv: This file contains information about the movies in the dataset. Each row represents one movie and contains the following columns: movieId, title, and genres.

ratings.csv: This file contains the ratings given by users to movies. Each row represents one rating and contains the following columns: userId, movieId, rating, and timestamp.

tags.csv: This file contains the tags applied by users to movies. Each row represents one tag application and contains the following columns: userId, movieId, tag, and timestamp.

Below is a bar chart showing the mean rating by genre in the MovieLens Small dataset.

Chart, bar chart

Description automatically generated

As we can see the average rating for each genre are close to each other. The following pie chart shows the percentage of movies in each genre in the dataset:

Chart, pie chart

Description automatically generated

The genre distribution in this dataset can affect a recommender system in several ways. For example, it can indicate the availability and diversity of movies in each genre, and thus affect the coverage and serendipity of the recommendations. It can also reveal the popularity and trends of different genres, therefore, analyzing and understanding the genre distribution in this dataset is important for building and evaluating a recommender system based on this dataset.

**Methodology**

This project aims to explore and compare different methods and finally create a hybrid recommender system. Each method in this project is described below.

**Content-based filtering:** This approach recommends items to users based on the characteristics or features of the items, such as genres, actors, directors, or plot keywords. This project uses a content-based approach to recommend movies to users based on their preferred genres.

|  |  |
| --- | --- |
| Movie | Genres |
| Schindler's List (1993) | Drama|War |
| Shawshank Redemption, The (1994) | Crime|Drama |
| Lord of the Rings: The Return of the King, The (2003) | Action|Adventure|Drama|Fantasy |

The similarity between the movies can be calculated by converting genres to dimensions and then computing the cosine similarity based on these dimensions:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Movies | Drama | War | Crime | Action | Adventure | Fantasy |
| Schindler's List | 1 | 1 | 0 | 0 | 0 | 0 |
| Shawshank Redemption | 1 | 0 | 1 | 0 | 0 | 0 |
| Lord of the Rings | 1 | 0 | 0 | 1 | 1 | 1 |

To calculate the multi-dimensional cosines, the following formula (Cosine similarity, n.d.) will be used:

Drama

War



The rating for a popular movie can be calculated by first, computing the similarity score between that movie and every movie that a user has rated, and then, using KNN approach, K movies with the closest similarity score to the movie that is being evaluated for the user will be selected. Finally, the weighted average of the similarity score of the K movies will be computed and weighed by the rating the user gave them. (Kane, 2020)

**Collaborative filtering:** Collaborative filtering approach can be done for User-User or Item-Item. User-user collaborative filtering calculates the similarity between users and recommends items based on the behavior of a user’s nearest neighbors. In other words, it finds users who are similar to a given user and recommends items that similar users have liked. Item-item collaborative filtering, on the other hand, calculates the similarity between items and recommends items that are similar to those previously liked by the user. (Bart Baesens, 2020)

In this project, the focus is on the movie-movie (item-item) approach as items are simpler and belong to a smaller set(genres). Furthermore, movie similarity is more meaningful as they tend to not change over time. (University, Collaborative Filtering, 2016)

To compute the rating for unknown values, first, we define the similarity of movie i and j as . For this project, centered cosine similarity which is a variation of cosine similarity where the attributes vectors are normalized by subtracting the vector mean will be utilized. Then k-nearest neighbors will be selected and rating (how much user u likes movie i) can be estimated as follow:

Where N(i, u) is a set of movies similar to movie i that were rated by user u. (Kane, 2020)

**Latent factor models:** Suppose R is the rating matrix with rows are movies, columns are users, and values are rating scores, the main idea of this method is to use matrix factorization and SVD (), to reduce the dimensionality of the user-movie matrix(R) and find the latent factors that can explain users ratings for different movies with the focus of reducing the discrepancy between the predicted value and true value.

This method solves the recommendation problem via optimization by using SVD and stochastic gradient descent (SGD). Assuming that matrix R can be written as (factorized R into two matrices) and by comparing it with SVD ():

Compare with SVD:

SVD returns minimum reconstruction error (SSE), and SSE and RMSE are monotonically related, therefore SVD is minimizing RMSE, but since the R matrix has missing values, SVD is not defined when entries are missing. To find P and Q the following optimization needs to be solved:

In summary in the above optimization problem, the goal is to find P and Q such that by summing over all the known ratings the square value of the rating minus our predicted rating are as small as possible. To solve the above optimization problem, gradient descent can be used (University, Latent Factor Recommender System, 2016).

To avoid overfitting the model to the training data, a regularization parameter can be added to the above minimization problem so we will have:

Where is error and is the length. is the regularization parameter.

To find the values for P and Q using gradient decent we will follow the following procedure:

* Initialize P and Q (missing rates are 0)
* Gradient decent:

Where is gradient/derivative of matrix Q as follows:

and

Furthermore, to model bias and user interactions, we can define u as the overall mean rating, bias of user u, and bias of move i:

The gradient descent will be with respect to P, Q, and . (University, Latent Factor Recommender System, 2016)

**Hybrid System (ensemble approach):** A hybrid recommender system can be built using different methods and, in this project, a weighted hybrid method will be explored. In the weighted hybrid approach, different weights are assigned to the predictions of each method, and they are combined linearly to produce a final prediction. The weights can be based on different criteria. For this project, accuracy, hit rate, and manually tuned weights (trying different weights and reporting the best result) will be tested to tune the model. (Mr.Avadhut D.Wagavkar, 2017)

**Implementation:**

In this project, python surprise package was used to build and evaluate recommender systems. Surprise is a scikit-like library that provides various tools for collaborative filtering, such as algorithms, datasets, and evaluation metrics. Surprise was chosen because it is easy to use, flexible, and well-documented. Different algorithms were applied from surprise packages, such as SVD and KNN, to the small Movie Lens dataset and their performance were compared, which will be discussed in the next section. (Hug, n.d.)

The code for this project can be found in the following GitHub repository:

<https://github.com/shshakib/Recommender-System>

In this project, object-oriented programming was used to implement a recommender system for movies. The project consisted of the following classes:

* Movie: This class reads data from CSV files and returns movie names and genres based on movie ID.
* DataSlicer: This class splits the data into train and test sets using a random state. It also generates leave-one-out and anti-test datasets.
* Methods: This class evaluated a dictionary list of methods. It is also returned the top\_n\_recommendation movies for a given user.
* Metrics: This class calculated the evaluation metrics for each method, such as MAE, RMSE, and hit rate.
* ContentBased: This class was inherited from the AlgoBase class of the surprise package and implemented a content-based filtering method using cosine similarity between movie genres.
* Hybrid: This class was inherited from the AlgoBase class of the surprise package and implemented a hybrid filtering method using a weighted average of the ratings from collaborative filtering, SVD, and content-based filtering.
* RunMe: This file created objects of the above classes and ran the main program. We can pass a Boolean compute argument to load predictions from previously saved results without computing them again since computing the similarity matrix might take a while.

**Evaluation and Final Results**

There are various methods to evaluate a recommender system such as MAE, RMSE, Hit-Rate, ARHR, and cHR, …. In this project, the focus is on RMSE and hit rate.

A lower RMSE means a more accurate prediction and this method is commonly used for offline evaluation of recommender systems, where historical data is used to test the performance of a model before deploying it to real users. However, RMSE does not capture other aspects of recommendation quality, such as diversity, novelty, or serendipity. The RMSE for our user/movie rating matrix R can be calculated as follows:

Where  is the predicted rating and  is the true rating of user u on movie i.

To avoid a narrow focus on accuracy and the fact that RMSE might penalize a method that does well for high ratings, in this project Hit rate and leave one out cross-validation was considered. To calculate hit rate, top-end recommendations for all the users in test set were generated, if one of the recommendations in a user’s top-end recommendation is something they rated, it will be considered a hit. (Users found something interesting enough to watch on their own). By adding all the hits in our top-end recommender system and dividing it by number of users we have hit rate. (Kane, 2020)

The following table summarizes the result for all 4 different methods and a random prediction.

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | RMSE | MAE | Hit-Rate |
| Content-based | 1.0547 | 0.8471 | 0.0089 |
| Collaborative filtering | 0.9904 | 0.7690 | 0.0015 |
| Latent factor | 0.8905 | 0.6873 | 0.0328 |
| Hybrid | 0.9142 | 0.7078 | 0.0224 |
| Random | 1.4384 | 1.1501 | 0.0045 |

To further interpret the results of recommender systems, it is necessary to remember how they work. Both Content-based and Collaborative filtering are using neighborhood-based methods, with Collaborative filtering KNN finds K similar items that were rated by the same user, and Content-based KNN utilizes similarity scores between everything that a user has rated and the movie that we want to predict a rating for. SVD, on the other hand, is a matrix factorization method that models user-item interactions using latent factors. Hybrid is a combination of these techniques, involving both similarity between items and latent factors.

Based on the above metrics, we can see that latent factor (SVD) has the best performance in terms of accuracy and hit rate, as it has the lowest RMSE and MAE values and the highest Hit-Rate value. As expected, Random has the worst performance in terms of accuracy and relevance, since it has the highest RMSE and MAE values and a very low Hit-Rate value since it just randomly recommends movies. Content-based and collaborative filtering have similar performance in terms of accuracy, but content-based has a slightly higher Hit-Rate value than collaborative filtering, which means it can recommend slightly more relevant items than collaborative filtering. Hybrid has a slightly worse performance than latent factor in terms of accuracy and relevance, but better than content-based and collaborative filtering. In the Hybrid approach, the weights are equal for each method.

As for the list of predicted movies, we can identify some well-known movies that were recommended to our test subject user. Among the top 20 recommended movies for our test subject user, we can find a couple of famous movies such as Usual Suspects, Taxi Driver, Fargo, Godfather, Godfather: Part II, L.A. Confidential, Seven Samurai, Matrix, Fight Club, Gladiator, and Harry Potter and the Half-Blood Prince. (Please see appendix for a list of top 10 for each method)

In conclusion, latent factor (SVD) is the best method among the algorithms that we test in this project for our small movie dataset, as it can predict ratings more accurately and recommend more relevant items to the users, but in this project, other evaluation metrics such as coverage, diversity, and novelty were not evaluated. Content-based and collaborative filtering are intermediate methods that can predict ratings moderately accurately and recommend some relevant items to the users. Considering other similarity features such as release years, IMDB score, or other properties of films such as average shot length, color variance, lights, and number of shots, … might improve these methods in a recommender system (Cremonesi, 2017). Hybrid recommender is a close second to latent factor, as it can predict ratings fairly accurately and recommend fairly relevant items to the users. Ultimately a recommender system should be tested online using A/B test so that we can measure how customers are reacting to the recommender system.

# References

Bart Baesens, S. v. (2020). *Item-Item Collaborative Filtering: a Refresher*. Retrieved from datamining apps: https://www.dataminingapps.com/2020/01/item-item-collaborative-filtering-a-refresher/

*Cosine similarity*. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Cosine\_similarity#:~:text=Cosine%20similarity%20is%20the%20cosine,the%20product%20of%20their%20lengths.

Cremonesi, Y. D. (2017). Using Mise-En-Sc{\`e}ne Visual Features based on MPEG-7 and Deep Learning for Movie Recommendation. *ArXiv*.

Harper, F. M. (2015). The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.*, 19.

Hug, N. (n.d.). *Matrix Factorization-based algorithms*. Retrieved from Surprise’ documentation!: https://surprise.readthedocs.io/en/stable/matrix\_factorization.html#surprise.prediction\_algorithms.matrix\_factorization.SVD

Kane, F. (2020). *Building Recommender Systems with Machine Learning and AI*. Retrieved from Linkedin Learning: https://www.linkedin.com/learning/building-recommender-systems-with-machine-learning-and-ai/install-anaconda-review-course-materials-and-create-movie-recommendations

Mr.Avadhut D.Wagavkar, P. M. (2017). Weighted Hybrid Approach in Recommendation Method. *International Journal of Computer Science Trends and Technology (IJCST)*, 5.

University, S. (2016). *Collaborative Filtering*. Retrieved from Youtube: https://www.youtube.com/watch?v=h9gpufJFF-0

University, S. (2016). *Latent Factor Recommender System*. Retrieved from Youtube: https://www.youtube.com/watch?v=E8aMcwmqsTg

Appendix:

List of recommended movies for a subject test user (id=85):

|  |  |  |  |
| --- | --- | --- | --- |
| Content-based | Collaborative filtering | Latent factor (SVD) | Hybrid |
| Dangerous Minds | Vive L'Amour (Ai qing wan sui) | Usual Suspects | Dangerous Minds |
| Dumbo | Vagabond (Sans toit ni loi) | Taxi Driver | Dumbo |
| Sleepers | Under Suspicion | Three Colors: Red (Trois couleurs: Rouge) | Sleepers |
| Escape from New York | Ariel | Fargo | Escape from New York |
| Cinema Paradiso (Nuovo cinema Paradiso) | Ali: Fear Eats the Soul (Angst essen Seele auf) | Wallace & Gromit: A Close Shave | Cinema Paradiso (Nuovo cinema Paradiso) |
| Deer Hunter | Distant (Uzak) | Godfather | Deer Hunter |
| Ben-Hur | Kwaidan (Kaidan) | Citizen Kane | Ben-Hur |
| Gandhi | Maborosi (Maboroshi no hikari) | Godfather: Part II | Gandhi |
| Dracula (Bram Stoker's Dracula) | Genghis Blues | Graduate | Dracula (Bram Stoker's Dracula) |
| Cape Fear | Virgin Spring, The (Jungfrukällan) | Donnie Brasco | Cape Fear |