CS-GY 9223 – MINING MASSIVE DATASET
FINAL REPORT

HETEROGENEOUS RECOMMENDATION SYSTEM

Under the Supervision of Prof. Gustavo Sandoval

By

Name	**Net ID**
Gayathri Prerepa	gp2254
Rohan Sardana	rs7445

NEW YORK UNIVERSITY

Spring 2022

Contents

1. Problem Statement and Hypothesis

1.1 Problem Statement

In this project, we need to build a recommendation system to suggest movies and books based on the user's reading preferences. People like watching movies that are similar to the books they read. For example, if someone liked the Harry Potter books; they could be recommended to watch Percy Jackson or Narnia.

The end goal of the project would be to recommend the user with five books and five movies that they can watch according to the book name that they enter.

1.2 Hypothesis

This heterogeneous recommendation system aims to utilize users' reading preferences to recommend related books and movies. The proposed framework uses collaborative filtering to recommend similar books and both content and collaborative based filtering to recommend pertinent movies based on Jaccard similarity on tags extracted from summaries and descriptions of the books and movies.

The initial hypothesis is that the user enters the book that is present in the books dataset, and we have enough ratings for that book to apply collaborative filtering. Also, we have movies that are of the same genre as that of the book. Our recommendation algorithms like ALS and NLP techniques like Cosine similarity and Jaccard similarity algorithms should be able to successfully generate recommendations on the basis of the input book. We expect to see the best result while using jaccard similarity algorithm, as we have two descriptions (one for book and one for movie); repetition of a word does not reduce their similarity.

2. Detailed Description of the Dataset

2.1 Detailed Description of the Dataset and how it was obtained

The following datasets are required: (a) Movies and (b) Books

Movies Dataset

The dataset [D1] we used for movies is famous MovieLens dataset which contains 25000095 ratings and 1093360 tags across 62423 movies. This data was created by 162541 users between January 09, 1995, and November 21, 2019 and generated on November 21, 2019.

Users were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided.

The data is contained in the files movies.csv, ratings.csv, and tags.csv.

We downloaded this dataset from GroupLens [D1].

Books Dataset

Dataset 1

The dataset [D2] we used for books genre and description is GoodReads dataset. The dataset contains 25 variables and 52478 records corresponding to books on the GoodReads Best Books Ever list (the largest list on the site). We downloaded the dataset from Zenodo [D2].

Dataset 2

The dataset [D3] we used for user ratings of books is another GoodReads dataset which contains information about approximately 10M books scraped from Goodreads API using the Goodreads Python library. It contains book name, id (to represent a user who gave rating), and rating. We downloaded the dataset from Kaggle [D3].

2.2 Basic Statistics

Before Data Preprocessing:

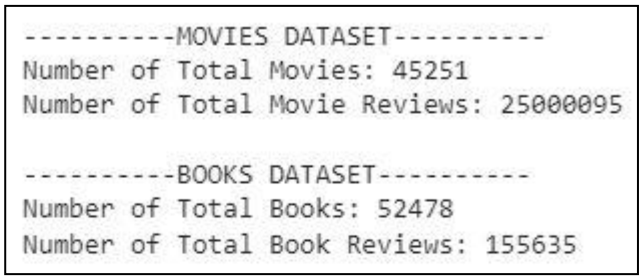
- Movies Dataset:
    - Number of total movies: 62423
    - Number of total reviews: 25000095

- Books Dataset:
    - Number of total books: 52478
    - Number of total book reviews: 155635

After Data Preprocessing:

- Movies Dataset:
    - Number of total movies: 45251
    - Number of total reviews: 25000095

- Books Dataset:
    - Number of total books: 52478
    - Number of total book reviews: 155635



```
----------MOVIES DATASET----------
Number of Total Movies: 45251
Number of Total Movie Reviews: 25000095

----------BOOKS DATASET----------
Number of Total Books: 52478
Number of Total Book Reviews: 155635
```

Fig 1: Preprocessed Dataset Basic Statistics

2.3 Initial Exploratory Data Analysis (EDA) phase

On our first exposure, we realized that the datasets we have are very vast which is a good things since it will give us a better overall result but it would also require a lot of cleaning as it is in its raw form. There are several stop words, repetitions, and gibberish which would require a lot of preprocessing to clear and get meaning out of which in turn would give us the similarity score that we are looking to calculate. The cleaning step is mentioned in the pre-processing step 2.5.

2.4 Features to use in the Analysis Phase

For the Books dataset we require:

- Book Name: For searching the input book
- Genre: For finding same genre movies
- Description: For finding tags and applying NLP techniques to match it to a movie
- User ratings: For applying user-based collaborative filtering

For the Movies dataset we require:

- Movie Name: For recommending the movie
- Genre: For matching the books to same genre movies
- Description: For finding tags and applying NLP techniques to match it to a book

2.5 Description of pre-processing steps

Preprocessing is a must, to clean the data for optimal output. Most of the cleaning is done using common cleaning techniques. Our datasets have a lot of features that we don't need. Also, the description of books and movies contain a lot of stop words, and special characters that if not preprocessed properly will result in an inefficient model.

To achieve the same, we have also resorted to preprocessing steps which are as follows:

- Convert genres and description in lowercase
- Remove punctuations, special characters, numbers, and stop words from description and convert to tags
- Normalize the user ratings and given them a numeric value from 1 to 5 instead of string ratings
- All line breaks were removed

  Cleaned dataset are saved in a new folder named: final_data

3. Detailed Description of the Model

3.1 Description of statistical methods and machine learning algorithms

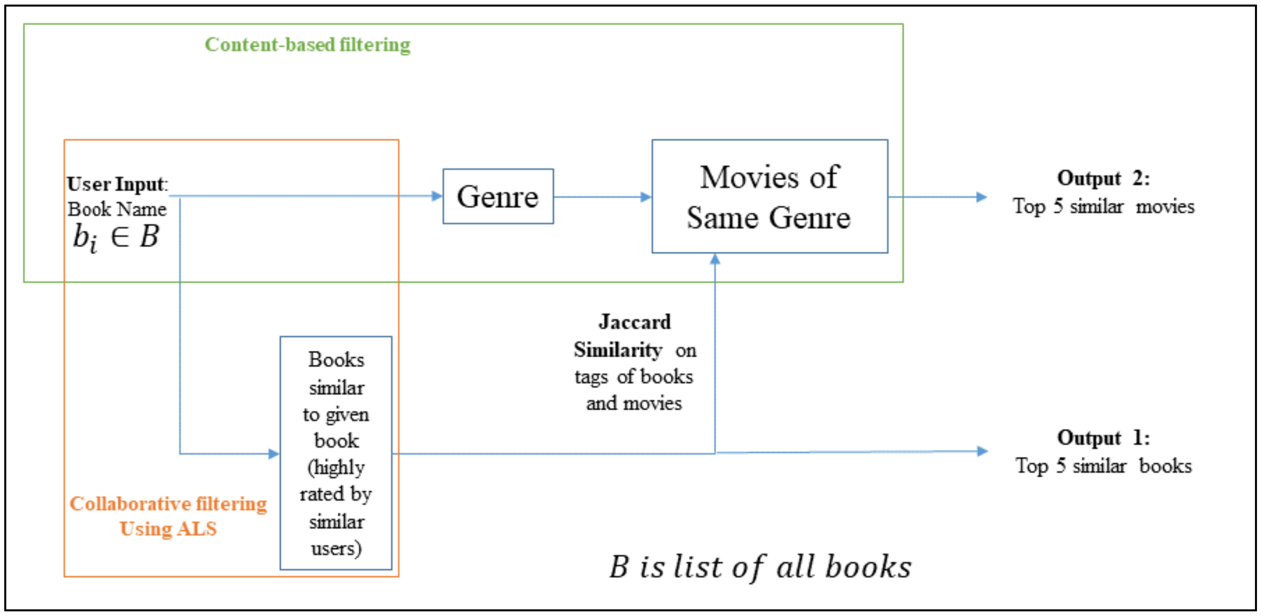

Content-based filtering
User Input: Book Name $b_i \in B$
Genre
Movies of Same Genre
Output 2: Top 5 similar movies
Jaccard Similarity on tags of books and movies
Books similar to given book (highly rated by similar users)
Output 1: Top 5 similar books
Collaborative filtering Using ALS
B is list of all books

Fig 2: Model Architecture for Heterogeneous Recommendation System

The most common types of recommendation systems are *content-based* and *collaborative filtering* recommender systems.

Content-based systems use metadata such as genre, producer, actor, author to recommend items, say movies or books. Such a recommendation would be for instance recommending *Infinity War* that featured *Vin Diesel* because someone watched and liked *The Fate of the Furious*. Similarly, you can get book recommendations from certain authors because you liked their writings. Content-based systems are based on the idea that if you liked a certain item you are most likely to like something that is similar to it.

In collaborative filtering, the behavior of a group of users is used to make recommendations to other users. The recommendation is based on the preference of other users. A simple example would be recommending a movie to a user based on the fact that their friend liked a certain movie. There are two types of collaborative models: *Memory-based* methods and *Model-based* methods. The advantage of memory-based techniques is that they are simple to implement and the resulting recommendations are often easy to explain. Memory based collaborative filtering is further divided into two types:

- **User-based collaborative filtering**: In this model, products are recommended to a user based on the fact that the products have been liked by users similar to the user. For example, if Derrick and Dennis like the same movies and a new movie comes out that Derick likes, then we can recommend that movie to Dennis because Derrick and Dennis seem to like the same movies.
- **Item-based collaborative filtering**: These systems identify similar items based on users' previous ratings. For example, if users A, B, and C gave a 5-star rating to books X and Y then when a user D buys book Y they also get a recommendation to purchase book X because the system identifies book X and Y as similar based on the ratings of users A, B, and C.

We will be using both Content-based and Collaborative based filtering systems to recommend movies and books. Figure 1 illustrates the proposed methodology of our framework.

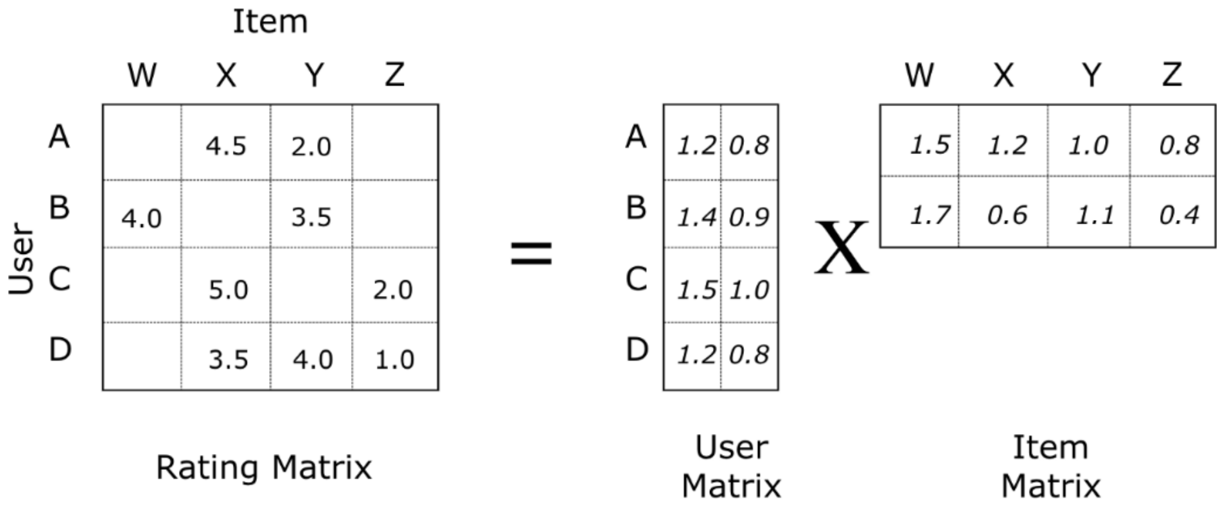
1. **Content-based filtering:** The genre of the book entered by the user will be utilized by the system to filter movies of the same genre.
2. **User-based collaborative filtering:** If the user inputs book b_i, rated highly by users [u_1, u_2, u_3…u_n], we will generate list of books [b_1, b_2, b_3…b_n], rated highly by the same users,

and then find the cosine similarity (based on tags) of these books with the movies generated using content based filtering.

This recommendation model is based on the combination of 2 methods, collaborative filtering and content based filtering. The first part is finding similar books based on the input by using the ALS algorithm. The next step is comparing these recommended books with the tags of the movies using Cosine similarity method and Jaccard similarity method. The method which performs better will be used for the final recommendation system.

ALS Method

The alternating least squares (ALS) algorithm divides a matrix R into two elements, U and V, resulting in R=UTV. Latent factors are the unknown row dimensions that are given to the algorithm as a parameter. The matrices U and V might be dubbed user and item matrix, respectively, because matrix factorization can be employed in the context of recommendation. The user matrix's ith column is designated by u_i, and the item matrix's ith column is v_i. The R matrix is also known as the ratings matrix.



$$\begin{array}{c|cccc} & W & X & Y & Z \\ \hline A & & 4.5 & 2.0 & \\ B & 4.0 & & 3.5 & \\ C & & 5.0 & & 2.0 \\ D & & 3.5 & 4.0 & 1.0 \end{array} = \begin{array}{c|cc} A & 1.2 & 0.8 \\ B & 1.4 & 0.9 \\ C & 1.5 & 1.0 \\ D & 1.2 & 0.8 \end{array} \times \begin{array}{cccc} W & X & Y & Z \\ \hline 1.5 & 1.2 & 1.0 & 0.8 \\ 1.7 & 0.6 & 1.1 & 0.4 \end{array}$$

The main idea is to build a matrix users X items rating values and try to factorize it, to recommend main products rated by other users. The method followed to do this is called **matrix factorization.** It is a class of *collaborative filtering* algorithms used in recommender systems. Matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices. In order to obtain such a representation where movies and users can be compared and matched, they both need to be cast as vectors of identical dimension.

Cosine Similarity

If we compute the similarity of 2 similar strings by counting the repeating words, they would be almost similar. But, there is an inherent flaw by simply counting words, especially when using larger files. Since words may be repeated more frequently in a large string, the comparison using this method becomes less accurate.

So, we have used the *cosine similarity method* using both CountVectorizer and TfidfVectorizer, to compute the similarity of such large strings. This algorithm, as the name suggests, is based on the cosine of vectors. The math behind this machine learning algorithm is:

$$\boldsymbol{A\,.\,B\ =\ ||A||\ ||B||\ cos\ \theta}$$

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

Example of cosine similarity:

Document 1 = “the best data science course”
Document 2 = “data science is popular”

	the	**best**	**data**	**science**	**course**	**is**	**popular**
D1	1	1	1	1	1	0	0
D2	0	0	1	1	0	1	1

D1 = [1, 1, 1, 1, 1, 0, 0]
D2 = [0, 0, 1, 1, 0, 1, 1]

$$\boldsymbol{Similarity(D1, D2)} = \frac{D1\,.\,D2}{||D1||\ ||D2||}\frac{2}{\sqrt{5}\sqrt{4}} = \frac{2}{20} = 0.44721$$

In simpler words, the vectors we are comparing here are the *tags of the movie* to the *tags of the book* entered by the user. The tags are converted to vectors (dictionaries in Python, with each number representing the number of times the word is repeated). This captures the angle between the tag vectors, lesser the angle, more the similarity. It is like computing the deviation of one vector w.r.t. the base vector. Thus, they have a *high* cosine similarity value, if the vectors are pointing in the *same direction*. We can now recommend the *top 5* or *top 10* items to the user by descending order of the similarity scores.

Jaccard Similarity

The Jaccard similarity compares items of two sets to see which of them are common and which are distinct. It's a measure of similarity for the two sets of data, with a range from 0 to 1. The higher the value, the more similar the two populations.

To calculate the Jaccard Similarity between them, we first find the total number of common items in both sets, then divide by the total number of items in either set:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Example for jaccard similarity:

Consider 2 sets C & D:

C = [0, 1, 2, 3, 4, 5]

D = [6, 7, 8, 9, 10]

- Number of observations in C & D: **{} = 0**
- Number of observations in either: **{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10} = 11**
- Jaccard similarity: **0/11 = 0**

3.2 Model Evaluation

The two techniques commonly used for applying NLP techniques are Jaccard Similarity and Cosine Similarity. Implementing the same, we have found following inferences:

- Cosine similarity can be evaluated using two techniques - TfidfVectorizer and CountVectorizer. Upon validation we found that, TFidfVectorizer is more powerful than

CountVectorizer because TF-IDF penalizes word with most occurrence in the document and gives less weight to those words.

- In comparison, Jaccard similarity only takes unique set of words for each sentence/document while cosine similarity takes total length of the vectors.
- If we repeat the word “friendship” in description 1 several times, cosine similarity changes but Jaccard similarity does not, which means Cosine similarity would hamper our recommendation model based on descriptions.
- Jaccard similarity is good for cases where duplication does not matter, cosine similarity is good for cases where duplication matters while analyzing text similarity.

Based on our problem’s need to have two descriptions (one for book and one for movie), it will be better to use Jaccard similarity as repetition of a word does not reduce their similarity.

4. Business Applications

- **Improved Customer Retention and Increase in Sales:** When it comes to companies such as Amazon - that are based on monthly subscriptions of their services like amazon prime, there has to be a reason why the customer should pay for their service every month. For instance, a customer purchases books for his kindle on *amazon.com*, the system can recommend the customer to watch movies that are available on amazon prime video related to the books purchased.
- **Recommendation of academic readings and videos based on research papers:** Websites like *scholar.google.com* can use this recommendation system based on user’s *youtube.com* watching history to recommend articles or vice-versa to recommend academic videos on *youtube.com* based on user’s article reading history on *scholar.google.com*.

5. Important Links

5.1 Github

- https://github.com/rohan-sardana/heterogeneous-recommendation-system

5.2 Colab Notebook

- https://colab.research.google.com/drive/1r3f70BuffNcaSikHd0MxWtcTKcYfMd0K?usp=sharing

5.3 Datasets

[D1] https://grouplens.org/datasets/movielens/25m/
[D2] https://zenodo.org/record/4265096#.YnxjHOjMKUl
[D3] https://www.kaggle.com/datasets/bahramjannesarr/goodreads-book-datasets-10m

6. References

[1] https://spark.apache.org/docs/latest/ml-collaborative-filtering.html
[2] https://studymachinelearning.com/cosine-similarity-text-similarity-metric/
[3] https://studymachinelearning.com/jaccard-similarity-text-similarity-metric-in-nlp/
[4] https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50
[5] https://pypi.org/project/neattext/
[6] https://towardsdatascience.com/understanding-cosine-similarity-and-its-application-fd42f585296a
[7] https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1
[8] https://en.wikipedia.org/wiki/Matrix_factorization_(recommender_systems)
[9] https://www.statology.org/jaccard-similarity/