Book Recommendation System based on query logs

Saima Sharleen Islam
University of Trento
Trento, Italy
saimasharleen.islam@studenti.unitn.it

Letizia Fabbri
University of Trento
Trento, Italy
letizia.fabbri@studenti.unitn.it

ABSTRACT

The book recommendation system anticipates the interests of purchasers and suggests books accordingly. This has enormous financial ramifications, as the likelihood of a book being purchased improves significantly if it is suggested by a powerful recommender. In this research, we examine the named things proposing a method that can shed light on the reader's potential options for reading related books. To do this, we offer an approach to automate and anticipate recommendations for a specific query. Finally we provide a possible method to compute the utility of a query.

KEYWORDS

Data Mining, Recommendation System, Utility Matrix, Query, Content Based, Collaborative Filtering, Hybrid SOlution, Query utility.

ACKNOWLEDGMENTS

We would like to thank Professor Dr. Yannis Velegrakis for clarifying every aspect of this project to us.

1 INTRODUCTION

In a world where there may be an overwhelming number of options, recommendation systems aid users in discovering and evaluating items of interest. Recommendation systems have evolved into useful tools in several fields (just to name a few, healthcare[4], marketing[12], manufacturing[14] or finance[18]) for identifying relevant patterns in data and predicting outcomes based on known patterns. These systems attempt to anticipate a user's rating, namely a preference, for an item and they are required to assist users in discovering more of what they like, through the provision of relevant alternatives.

As already declared our focus will be on books recommendation systems. These ones are widely utilized to offer things to customers based on their individual interests. In a highly competitive market, a book recommendation system is one of the most effective methods for generating earnings by retaining more consumers. Websites for reading and selling books online, such as Kindle[1] and Goodreads[2], compete with one another in numerous ways. Their book suggestion system is one of these significant features. Generally speaking, the purpose of a book recommendation system is to suggest books of interest to the buyer. However customers might have disparate preferences composed of different characteristics (for instance one might like horror books but only ebook ones). Consequently, books with more than one feature are often searched and according to the users, all of these characteristics must be satisfied altogether. To be more precise, every time users is searching something, in this case a book, is sending implicitly a query to the database. The problem is that the queries asked are not always a unique object, as they may be built of several criteria as the previous example shown. Moreover, users can give implicitly different levels of priority to each requirement based on their own tastes. In other words, this collection of requirements may not all have the same weight; some of them may be more significant than others. Another challenge regards the type of users we are considering: some could have read and rated many books, thus predicting their scores will be easier and more accurate. Differently, for those with no or few ratings before, setting a good algorithm to predict their ratings will be more arduous. Our solution will try to address all these specificities.

2 PROBLEM INTERPRETATION

As stated before, our research will concentrate on a specific sort of recommendation system, namely the query based recommendation system in a particular context (books). As its name implies, queries are encouraged under this section; through the development of an hybrid solution, our approach attempts to take into account the peculiarities of this type of recommendation system. Our algorithm will be implemented with a random dataset and then contrasted and analyzed using a dataset scraped in the web that is composed of:

(1) goodreads_books.json.gz which contains details about each individual book.
(2) goodreads_reviews_dedup.json.gz which contains information about each user's rating of a book.

After these steps, a general method on how to compute the utility of a query is provided, assigning more importance to the scores directly judged by the users and less to the ones estimated by our algorithm.

Given this context, a more thorough explanation of the issue is necessary. We set out:

(1) A relational table that contains tuples. The first row is composed of the names of the main book attributes ('bookID', 'title', 'authors', 'average_rating', 'isbn', 'isbn13', 'language_code', 'num_pages', 'ratings_count', 'text_reviews_count', 'publication_date', 'publisher' in our case), while the following ones enclose a tuple that contain specific realizations of each attribute.
(2) A query set as a union of tuples. In this dataset the first column contains the query identifier (a number in our case). Instead each following column contain the tuples that compose the query. Here, a generic query, q_j could be as follows: $q_{189} =< language = English, title = The\_Alchemist >$. As can be seen, each condition consists of an attribute and the attribute's manifestation. Notice that in this paper, we are going to consider queries with one to four criteria.
(3) A set of users $u_1, ...u_n$ as an identifier vectors.

(4) A rating ranging from 1 to 100, where 1 indicates that the user dislikes the condition fully and 100 indicates that the user likes the query completely. In other words, this indicates the amount of pleasure each user has with the query's results.
(5) The utility matrix as a matrix with users as rows and queries as columns. The entries consist of ratings from users. In our datasets, items may be null, indicating that the user did not perform the specified query. For instance:

	q_1	q_2	q_3
u_1	100		10
u_2	50	20	90
u_3	20	80	

Table 1: Example of utility matrix

In table 1, u_1 did not like the results of q_3, while u_2 appreciated them; finally, the same query was not searched by u_3.
(6) A certain measure of similarity between pair of users or items.
(7) A certain threshold n representing the top n similar items or users related to a generic item or user; we are going to consider them in order to estimate the missing ratings.
(8) A certain threshold k above which we are going to recommend the query to the user.
(9) Two different weights, one for the queries directly evaluated by users and one for estimation of these scores. They will be used in the computation of the utility of a query.

We are tasked with developing a system that recommends books to users depending on their preferences. Using the provided information, we may conclude:

(1) Filling up the ratings R in the utility matrix, exploiting the similarity measure and the threshold n previously defined.
(2) Recommending to each user all the queries that have a score higher than the threshold k.

Then, we have to evaluate the accuracy of our output adopting the two mentioned datasets. Finally using all the ratings and the weights established before, we have to propose a method to compute the utility of the query for all users. Formally speaking, the utility for the generic query (U_j) must be:

$$U_j|U_{u1j}, ...U_{unj} = f(R, W) \tag{1}$$

where:

(1) U_{u1j} represents the utility for the q_j expressed by u_1.
(2) W represents the two weights previously defined.
(3) R represents the ratings of the utility matrix

Our goal will be specifying the nature of the function *f*.

3 RELATED WORK

In our case, to construct a system capable of recommending a book to a user, generally the two main family to address this issue are content based approach or collaborative filtering approach. The former case aims to find similar items through the exploitation of their features[17]. The first step is building an item profile, in which one has to specify the features on which the item is composed. Then one is required to construct a user profile, where each feature importance is valued based on the ratings of the utility matrix[17]. Finally this system is going to propose new items that are similar, i.e. they have similar features, to the ones already highly rated[17]. However, through the application of this method, one can face the problem of finding the appropriate features in advance[15]. Moreover since this system is not able to recommend items outside the user profile[15], collaborative filtering approach, more specifically user x user, can be applied. By contrast, this procedure is based on the searching of the most similar users in terms of ratings[10]. This method is going to propose items that were rated highly to the correspondent similar users[10]. This mechanism is able to recommend items that are outside of the user day-to-day search, suggesting him or her something different[11]. Yet, this method requires data from different users (while content based not) and at the same time if an item has not been rated previously, the algorithm is not able to recommend it[11].

To evaluate the hybrid solution, cross validation was applied. According to Payam, Lei and Huan[16], this procedure requires a dataset division in two main parts: one represents the training part and the other the test part. In the training part one will perform the models required in his or her task (in our case we applied it our hybrid solution) and the other one will serve as comparison to understand what is the degree of accuracy of these predictions. To avoid possible bias in considering only a fixed portion of the dataset, this procedure is repeated k times, using the data k-1 times as training set and the remaining one as test set. We chose k=10, however every number from 2 to n-1 is possible, where n is the number of units of the dataset. When k=n-1 this type of cross-validation is named Leave One Out Cross Validation (LOOCV, see for instance Collins, Graham, and Hansen[9]). In this case n-1 units will be used as training set and the remaining one as test set. This procedure is repeated also to ensure that every units of the dataset will be one time in test set. During each step a measure of goodness of the model is computed the final one is the average of the ones of each step. This procedure can be implemented in different contexts: for instance in a classification problem one can employ cross validation providing AUROC (Area Under the Receiver Operating Characteristics) to test the classification ability of the developped solution (see for instance Hanley and McNeil[3]). In our paper we computed RMSE and MAE (this is going to be explained later in the same paragraph). Notwithstanding the higher is k the more this procedure is computational expensive, however it is encouraged as a initial form of verification, because it prevents overfitting problem of the model, namely when the solution developed fits too well on the dataset used, that the risk of poor performance with different datasets is serious[5]. Arguably, as a further measure of assessment, testing the algorithm with external datasets is recommendable, in order to improve the degree of replicability and generalizability of the algorithm and checking definitely for possible overfitting.

As mentioned before, final considerations are needed for MAE. Recalling Yunchuan, Fang, and Wang's work[13], MAE is defined as:

$$MAE = \frac{\sum_{i=1}^{n} |e_i|}{n} \tag{2}$$

Thus MAE is the average of the absolute error committed by the model. According to Chai and Draxler [8], RMSE performs better when the errors distribution is expected to be gaussian. Generally speaking, having large datasets, due to the central limit theorem, one can infer with a quite degree of confidence that model errors will be gaussian. The same authors claimed that MAE would perform better with uniform errors distributions, however there is disagreement on such point[6, 13]. Yet, MAE is more robust to the outliers due to the consideration of the absolute value of the difference in its formula, instead of the square of the difference.

4 SOLUTION

The purpose of this section is to provide the reader with a better understanding of the structure of the dataset by describing its production and providing descriptive statistics. Then, we will discuss the challenge of filling in the utility matrix, and using the obtained data, we will develop a method to compute the utility of a query for all users.

4.1 Dataset preparation

Here is a description of our technique for scraping a **real dataset** from the web. Then, we will discuss the creation of the **synthetic dataset**.

A relational table, a user set, a query set, and a User-Query utility matrix should be required inputs for our approach. Consequently, we constructed our relational table initially. There are tens of thousands of accessible book datasets, including the goodreads dataset. These datasets contain either 100,000 or 1,000,000 values. Due of our limited processing capabilities, only 45,641 rows are used. Also, we would want to emphasize that, instead of utilizing small datasets, we attempted to choose datasets that were slightly larger because of this. The more information a recommendation system knows about user preferences and behaviors, the more data it has to work with. Yet, this can result in more precise and tailored advice. In addition, it typically has a greater diversity of things and users, which might assist a recommendation system in generating various recommendations that are not unduly focused on a narrow subset of items. When a system lacks sufficient knowledge about a new user or object, it is difficult to generate effective recommendations. Large datasets can aid in the resolution of these issues since they enable the algorithm to discover similar users or products from which to derive a prediction. Using their API key, we extracted their data. Here is the pseudocode for our actions on this website:

A user set was then built. The collection of users consists of merely an array of ids. This data is provided in a file format, with one user ID per line. On the basis of our relational table, we then generated a query set, which is a collection of previously supplied queries. Each query has a unique identity and definition (i.e., the set of conditions). The query set is expressed as a CSV file with attribute=value conditions on each line, separated by commas. Then, the User-Query utility matrix contains, for each combination of user and query, a value between 1 and 100 that represents the user's level of satisfaction with the result set for that query. This is also offered as a CSV file, with the first row including a list of query ids separated by commas. Each alternate row begins with a userId, then a sequence of fields separated by commas (as many as the number of query names of the first row). These fields may be left empty or populated with numbers between 1 and 100. Here we provide the pseudocode for creating the utility matrix.

Algorithm 1 Scrape data from Goodreads

1. **Import** the necessary libraries (e.g. scrapy)
2. **Create** a new class called "GoodreadsSpider" that inherits from scrapy.Spider
3. **Define** the name and starting URLs for the spider
4. **Create** a function called "parse" that takes in a "response" parameter
5. **Use** a loop to iterate through each "book" element on the page
6. **Within** the loop, use CSS selectors to extract the "bookID","title","authors","average_rating","isbn","language_code", "num_pages","ratings_count","text_reviews_count","publication_date", "publisher" of the book
7. **Add** the extracted data to a dictionary
8. **Yield** the dictionary to return the scraped data

Algorithm 2 Pseudocode for creating a utility matrix

1. **import** necessary libraries (pandas and random)
2. **read** in a csv file named "user_set.csv" into a DataFrame named "user_set"
3. **create** an empty DataFrame named "utility_matrix" with columns "user_id" and unique values from "query_id" column of "user_set"
4. **create** a list of unique "query_id" from "user_set" and join them into a string
5. **add** the string of "query_id" as the first row of "utility_matrix"
6. **for** each row in "user_set"
   if "user_id" not in "utility_matrix"
      add the "user_id" to the "utility_matrix"
      fill in the ratings for the user-query combination with a random integer value between 1 to 100 with probability of 80%
7. **for** each "user_id" and "query_id" in the "utility_matrix"
   if cell is empty
      fill in the cell with a random value between 1 and 100 with a probability of 80%
8. export the "utility_matrix" to a csv file named "utility_matrix.csv"

Regarding the construction of the **synthetic dataset**, it generates a csv file named "book_dataset.csv" with a random book dataset. It utilizes the "Faker" library to generate fictitious data such as the book's title, author, ISBN number, release date, etc. Notice that we decide to use the same attributes of the real dataset in order to make better and complete comparisons between them.

It begins by opening a new csv file titled "random_books.csv," starting a csv writer, and writing the header row containing the field names. Then, the "Faker" library generates 20,000 rows of fake data and writes each row to the CSV file. The file is then renamed from "random books.csv" to "book dataset.csv." Following this, we constructed the user id set, the query set, and the utility matrix in the

same manner as for our real dataset.

4.2 Datasets Description

This table 2 provides a succinct summary of both datasets:

	Synthetic Dataset	Real Dataset
N	20,000	20,000
P	1,000	5,000
Empty-Values	50%	20%

Table 2: Description for both dataset

We included:

(1) N: Number of users
(2) P: Number of queries
(3) Empty-Values: Percentage of empty cells in the utility matrix.

Note that for both datasets, we have set N > P, but a different percentage of empty values.

4.3 Filling the utility matrix

This section is more focused on providing a theoretical explanation of all the steps of our approach. Evaluation and more practical considerations will be given in the section 5.
We present a method for computing the missing values in the utility matrix values based on the combination of two distinct algorithms. Initially, as stated in the problem statement, our utility matrix also contained NAN-categorized empty values. To enhance the utility matrix, we intend to fill the NAN values with a combination of collaborative recommendation system and content recommendation system. But before we describe our combination recommendation system, we'd like to mention that prior to implementing our proposed method, we filled the NAN values with content and calculated the RMSE value, and then we filled the NAN values with collaborative and calculated the RMSE value for this one as well.

Our solution started with the application of a content based algorithm (we will denote this as solution 1 to fill up our matrix). We already had an item profile, because each query already contains the features upon which is composed. In particular this was contained in the relational tables, whose characterics were listed in the assumption 1 of paragraph 2. Thus a typical problem of content based algorithm of finding items features did not involve us. Subsequently we created a user profile algorithm, associating to each feature a rating. Then cosine similarity was computed across these profile; we chose as a measure of similarity (1-cosine distance), as this measure enable us to take into account the queries ratings. Finally we considered for each queries its top ten similar ones (so our threshold n solution 1 is n=10) and the missing value were computed as an average of the ratings inside them weighted for the corresponding similarity values. As an example, we provide the formula used to compute the estimate utility for the generic q_j:

$$r_{ej} = \frac{\sum_{i=1}^{5} r_{tij} * s_{ij}}{\sum_{i=1}^{5} s_{ij}} \tag{3}$$

Where:

(1) r_{ej}: estimated rating for q_j;
(2) r_{tij}: rating already given for q_j among the top five similar queries to q_j;
(3) s_{ij}: generic measure of similarity;

Then we adopt a user x user collaborative filtering and to perform this, we compute cosine similarity directly across the users. Resorting equation 2, we estimated the missing utility through the computation of the weighted average mean of the already known ratings of the top five similar queries (so the threshold n for solution 2 is n=5) weighted for the corresponding measure of similarity.
To sum up, to date we obtained two different filled up utility matrix, with solution 1 and solution 2. Finally we built our main solution, i.e. a union of the previous ones (see Figure 1 for a general overview). Through this procedure, we were able to propose new queries to users, also going out the range of conditions that users tend to search. Meanwhile we were able to exploit each feature (each condition) of a query, proposing to users queries that tend to be similar to their own tastes. To begin, we computed RMSE for both the algorithms $RMSE_1$ for solution 1 and $RMSE_2$ for solution 2. Then we summed the result, obtaining $RMSE_{tot} = RMSE_1 + RMSE_2$. Then we divide each RMSE for the $RMSE_{tot}$, in that way we can scale both of measure to 1. The two metrics have been used as weights that have been multiplied to the corresponding utility matrix. However with such procedure we were assigning more importance to the solution that was having an higher RMSE. Since we wanted to attain the opposite result, i.e. penalizing the solution with higher RMSE, we performed a subtraction between one and each RMSE previously computed. In that way we were able to penalize the solutions with higher RMSE and simultaneously keeping the scale to 1. Note that maintaining the scale to 1 is fundamental, because it allows us not to change the magnitude order of the measurements. Mathematically expressed we did:

$$Ut_{hybrid} = W_1 * U_1 + W_2 * U_2 \tag{4}$$

where:

(1) Ut_{hybrid}: utility obtained with the hybrid method;
(2) U_1: utility matrix according to solution 1;
(3) U_2: utility matrix according to solution 2;
(4) W_1: weight for the matrix 1. It was found as: $1 - \frac{RMSE_1}{RMSE_{tot}}$;
(5) W_2: weight for the matrix 2. It was found as: $1 - \frac{RMSE_2}{RMSE_{tot}}$;

Here we provide the pseudocode to compute our hybrid solution (see Algorithm 3).
Hence, the weighted sum of the two matrix has returned a new utility matrix precisely Ut_{hybrid}. From it, we proposed to each user only those queries that have an estimated ratings above than 70. Thus, our threshold declared in the problem statement part will be 70, because we believe that a score close to 50 expresses neutrality; moreover with a score close to 60, the user likes not so strongly those queries, (our score goes from 0 to 100), thus we believe that a score of 60 or slight above will be not sufficient in proving that the user will like those queries. In the end, through cross validation, we compute the RMSE also for this last solution. Yet, since we did not want to rely our conclusions only on a measure already exploited, MAE was also calculated.

Algorithm 3 Pseudocode for creating our hybrid solution

```
import necessary libraries (pandas, time and numpy)
start a timer
estimate missing ratings
  for each row in the DataFrame
    for each column in the DataFrame
      if cell is missing
        fill the cell using the column average of the item
compute RMSE for solution 1 and solution 2
calculate the weights to be used in the hybrid utility matrix
create a hybrid utility matrix using the weights
save the hybrid utility matrix to a CSV file
stop the timer
print the time taken
```

As a final consideration, we are aware of the existences of hybrid

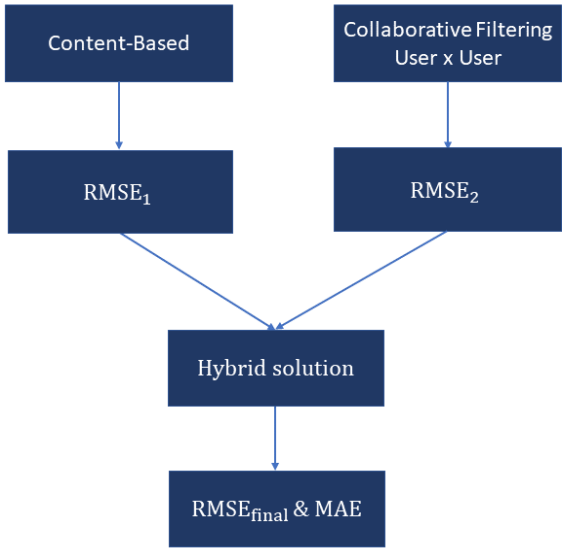

Figure 1: Development of the hybrid solution

solution that tries to find a balance between two different approach (see for instance Burker's paper [7] for applications of traditional hybrid solutions, while see Çano and Morisio's review [19] for a more complete framework of the traditional hybrid approach). However investigating the literature, we saw that too little emphasis was given to the concept of RMSE. On the contrary we believe it is fundamental measure to be considered and included more in the analyses.

4.4 Utility of a query

Additionally, we propose a way to compute the utility of a query for all the users. As a premise, since we deem that the utility of a query is expressed directly by a score, we will use these terms interchangeably.
Mathematically speaking, the utility of a query can be decomposed in two main elements: the importance of each attribute (i.e the conditions that form the query) and external factors. We believe that these variables might have changed the total rating assigned to the query. Possible examples of them should include the speed of internet connection or the time necessary to the database to provide answers to this query. Indeed if the user is forced to wait, probably he will be bothered by the waiting and he will rate lower this query also if the result is consistent with his or her expectations. Note that one can estimate the coefficients of this model, discovering that some conditions are more relevant than others in determining the final scores.Therefore, this score is the interaction of these two components. One can also express this relation through a linear regression:

$$r_{tij} = b_{ij1} * A_{j1} + b_{ij2} * A_{j2} + ... + b_{ijn} * A_{jn} + e_i * Z_i + \epsilon_i \quad (5)$$

The variables in the equations are the following:

(1) r_{tij}: Utility of the query j for user i, so the score assigned by user i to the query j
(2) b_{ij1}: Importance of the attribute 1 of the query j for the user i;
(3) A_{j1}: Attribute 1 of the query j;
(4) e_i: User i's importance of the external variables in determining the total rating for the query j;
(5) Z_i: External variables for the user i that might have affected the rating for the query j;
(6) ϵ_i: Error of the model for the user i;

We suppose that each expressed rating is already the results of this model. Since we have already the generic rating r_{tij}, the generic utility U_j associated to the query q_j could be computed through the following process:

$$U_j = W_{tj} * R_{tj} + W_{ej} * R_{ej} \quad (6)$$

$$R_{ej} = \frac{\sum_{h=1}^{K} r_{ehj}}{K} \quad (7)$$

$$R_{tj} = \frac{\sum_{i=1}^{L} r_{tij}}{L} \quad (8)$$

$$W_{tj} = \frac{L}{L+K} \quad (9)$$

$$W_{ej} = \frac{K}{L+K} \quad (10)$$

The variables in the equations are the following:

(1) U_j: Utility of the generic query j;
(2) W_{tj}: Weight related to the "true" score, (with true score we define all the individuals that effectively searched for this query and later they rated it);
(3) W_{ej}: Weight related to the estimated score (i.e. the one estimated through the algorithm);
(4) R_{ej}: Average estimated utility for the query j obtained through the application of an algorithm;
(5) R_{tj}: Average utility for the query j obtained by users who search effectively for this query;
(6) r_{ehj}: Estimated utility of the query j for user h;
(7) r_{tij}: Utility of the query j for user i;

(8) K: is the total number of ratings estimated by the algorithm for the query j;
(9) L: total number of "true" scores for query j;

The utility of a generic query was seen as the weighted average of the true ratings assigned by the users and the ones estimated by the algorithm (1). We believe that the true scores must have an higher influence than the ones estimated, due to the presence of possible estimate errors. To compute W_{tj} for each query, we propose to count the number of scores expressed by the users and divided it by the total number users (4). Similarly one has to count the number of scores estimated by the algorithm and then divided it by the same denominator (5). In this way one attains two different numbers, whose sum is 1. In case L<K, since the estimated scores are more than the real ones, we advice to subtract from 1 W_{tj} and subtract from 1 W_{ej}. Basically the same trick was applied to find W_1 and W_2. In that way we come up with attributing less importance to the estimated ratings, whatever number they are.
After the computation of the weights, it is necessary to find the R_{ej} and R_{tj}. For the former we encourage an average of all the estimated ratings for the generic query (7). Instead for the former we suggest the computation of the mean of all the already given scores for the generic query (8).

Final considerations are needed:

(1) In our opinion it would be superfluous to add an extra-coefficient that expresses the consistency between the users expected results after they asked the queries and the ones provided by the database. Indeed, the ratings are already an expression of this, making this extra-term redundant.
(2) We expected that R_{ej} could be 0 or R_{tj} (and so their related weights), however we this method is still possible to compute the utility of this query. Of course we recognize that in this case the weights would be unnecessary.

5 EXPERIMENTAL EVALUATION

In this section, which is more practical, we are going to evaluate at first the filling of the utility matrix (paragraph 5.1) and then the computation of queries utility (paragraph 5.2).

5.1 Experimental Evaluation for filling the utility matrix

As stated previously, the NAN values were filled using a content-based recommendation method (see Algorithm 4). After reading a csv file that contains a utility matrix, where each row represents a user, and each column is a query, and the entries are the ratings given by the users to the questions. The code substitutes NaN values with approximated values in the matrix. The cosine similarity between users and searches is calculated using the sklearn cosine_similarity function. The cosine similarity measures the cosine of the angle between two non-zero vectors of an inner product space to determine their similarity. The function iterates over each row and column of the utility matrix to determine whether the current cell value is NaN. If the value is NaN, the code estimates what it should be. The function begins by determining the average rating of the current user and the current item. Based on the cosine similarity scores, the code then identifies the k nearest neighbors (k = 10) of the current user and the current item. The algorithm then utilizes these k nearest neighbors to estimate the missing value by taking a weighted average of their ratings, where the weights are the cosine_similarity scores between the current user and the k closest users, and the current item and the k closest objects. Then filled utility matrix is then saved to a csv file.
Then, we implement collaborative filtering by utilizing Pearson correlation to fill in missing utility matrix values (see Algorithm 5). The function begins by reading a CSV file containing a utility matrix, and then fills any missing values in the matrix with the mean of each column. The Pearson correlation coefficient between each pair of users is then calculated and kept in the variable "user corr."

Algorithm 4 Pseudocode for creating a content utility matrix

import necessary libraries (pandas and numpy)
read the utility matrix
calculation done between users and queries using the cosine_similarity
  for each row & column in the utility matrix
    if current cell value = NAN
      estimates the value based on the cosine similarity scores
      identifies the k nearest neighbors (k = 10) of the current user and the current item.
      estimate the missing value by taking a weighted average of their ratings, and the current item and the k closest objects
save the content utility matrix to a CSV file

Algorithm 5 Pseudocode for creating a collaborative utility matrix

import necessary libraries (pandas and numpy)
read the utility matrix
fill any missing values with the mean of each column
compute the Pearson correlation coefficient between each pair of users
  for each row & column in the utility matrix
    if current cell value = NAN
      estimates the value based on the k nearest neighbors of the user and user, as indicated by the pearson correlation coefficient
      identifies the k nearest neighbors (k = 5) of the current user and the current item.
      estimated value for the missing cell is the weighted average of the evaluations of the current item by these k similar users, with the weights being the absolute values of the Pearson correlation coefficients
save the collaborative utility matrix to a CSV file

The subsequent step entails looping through each matrix cell and determining if the value is absent (NaN). If so, the code computes the estimated value for that cell based on the k nearest neighbors of the user and item in question, as indicated by the Pearson correlation coefficients.

To achieve this, the method first collects the current user's and item's ratings, ignoring any missing values. The algorithm then

selects the k users who have the highest Pearson correlation coefficients with the current user. The estimated value for the missing cell is the weighted average of the evaluations of the current item by these k similar users, with the weights being the absolute values of the Pearson correlation coefficients. The completed matrix is then saved to a new CSV file.

For our hybrid solution, at first, we calculated the RMSE for both the content and collaborative solutions. We then determined the entire RMSE value. We then determined the weighted values for each solution. Then we recommended queries whose ratings was above 70 (see Algorithm 6).

Algorithm 6 Pseudocode for recommending queries to the users

import necessary libraries (pandas and numpy)
read in the utility matrix
threshold to consider a query as recommended; here threshold = 70
create an empty dataframe to store the recommended queries
iterate over each user and find the queries with an estimated rating above the threshold
  for each user in the DataFrame
    recommend queries > threshold
    recommendations = append this to list(user & query)
save the recommended queries to a CSV file

Regarding the filling of the utility matrix for the evaluations that will follow, we have considered the average rating of the query as a baseline, namely the mean per column of the matrix. Thus, we supposed that the true score of the missing values in the utility matrix is going to be the query average. Precising this, We computed:

(1) RMSE and MAE for the synthetic and real dataset (see Figure 2 and Figure 3). Recalling the paragraph 3, we decided to use MAE as additional measure of evaluation as it is more robust to the outliers. Moreover, since we already resort RMSE in the development of the hybrid solution, we agreed to add an extra measure of assessment.

```
RMSE of hybrid solution: 35.69238866381714
MAE of hybrid solution: 25.23552574054206
```

Figure 2: RMSE & MAE of Synthetic Dataset

```
RMSE of hybrid solution: 22.586449557449995
MAE of hybrid solution: 10.10272272154499
```

Figure 3: RMSE & MAE of Real Dataset

(2) Time required by the algorithm for the synthetic and real dataset (see Figure 4 and Figure 5).
(3) Memory required by the algorithm for the synthetic and real dataset MISSING (see Figure 6 and Figure 7).

We compared the obtained results between the two datasets observing that:

```
Time taken:  432.7803328037262
```

Figure 4: Time (seconds) required by the algorithm to suggest queries based on the Synthetic Dataset

```
Time taken:  1452.3658142089844
```

Figure 5: Time (seconds) required by the algorithm to suggest queries based on the Real Dataset

```
Memory usage of the data: 76.29 MB
```

Figure 6: Memory required by the algorithm to suggest queries based on the Synthetic Dataset

```
Memory usage of the data: 762.94 MB
```

Figure 7: Memory required by the algorithm to suggest queries based on the Real Dataset

(1) Our hybrid solution performs better with the real dataset than random dataset. We hypothesize that this happened due to the larger dimension of the real dataset and the lower presence of empty cells (recall Table 2). In other words, we can conclude that both percentage of empty values and both the total cells affected the performance of our algorithm.
(2) The time required for the algorithm is higher in the real dataset due to the presence of more data, even if the presence of a minor percentage of empty cells.
(3) As expected, the memory usage for the real dataset is higher than the synthetic one.

In addition we checked that for both of the datasets, the number of queries suggested for each users was above 200 for the synthetic dataset and 1,000 for the real dataset, respectively at least the 20% of the total queries. While for the the real dataset the result was positive, for the synthetic dataset some users were receiving suggestions under the aforementioned threshold (as can be noticed from 9). However this number was limited and not much below the threshold. Further, as an example, we provide a representation of the number of the queries suggested for 5 random users, considering the real dataset (see Figure 8) and the synthetic dataset (see Figure 9).

Notice that the complexity of the part A is $O(n^2)$ where n is the number of items in the utility matrix (df). This is because of the two nested for loops in the code, which go through every item in the matrix. The similarity calculation and the root mean squared error calculation are both $O((n^2))$ as well, but these are minor in comparison to the nested for loops.
Regarding part A, a final observation is needed: so far we performed our analyses on dataset where the number of users (N) was greater

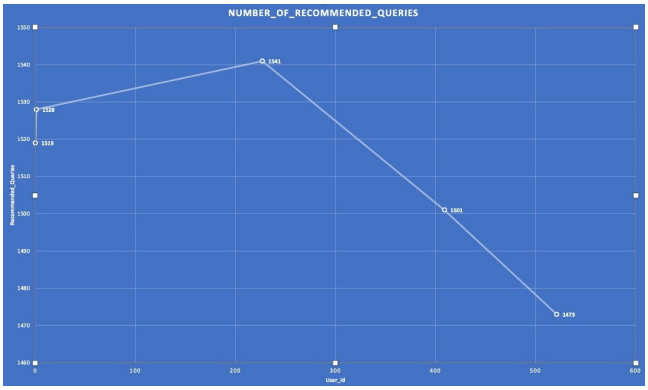

Figure 8: Recommended queries for 5 random users for the real dataset

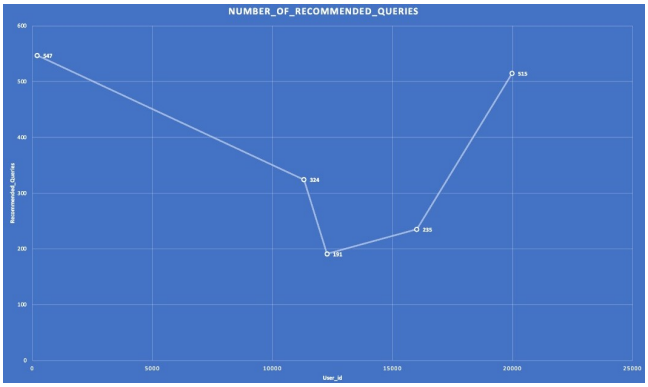

Figure 9: Recommended queries for 5 random users for random dataset

```
RMSE of hybrid solution: 35.708518158987815
MAE of hybrid solution: 25.250244338301215
```

Figure 10: RMSE & MAE of Synthetic Dataset when N<P

than the number of queries (P). Thus, we substituted in the synthetic dataset N and P, respectively in N=1,000 P=10,000, leaving the percentage of empty cells to be 50%. These changes were applied because we deem that in the reality one can face not only in a situation where the number of users are greater than the variables but also in the opposite case. Hence, we wanted to test the performance of the algorithm when N<P and compared it when N>P, always considering in the synthetic dataset. In this case we observed that:

(1) RMSE & MAE did not change significantly when N<P compared to N>P. We might conclude that the N and P do not affect these measure (see Figure 2 and Figure 10).
(2) The time decreased significantly when N<P, suggesting that the algorithm performed better when N<P (see Figure 4 and Figure 11).
(3) In spite of this change, memory usage does not vary (see Figure 6 and 12).

```
Time taken:  291.5119957923889
```

Figure 11: Time (seconds) required by the algorithm to suggest queries based on the Synthetic Dataset when N<P

```
Memory usage of the data: 76.29 MB
```

Figure 12: Memory used by the algorithm on the Synthetic Dataset when N<P

5.2 Experimental evaluation for the computation of the utility of a query

In this section we provide an overall evaluation regarding the computation of queries utility. At first, as we declared in the paragraph 4, we computed this utility as a weighted average between the estimated ratings and the already existing ratings, paying attention of penalizing the estimated ones. Here the pseudocode is provided (see Algorithm 7).

Algorithm 7 Pseudocode for computing the utility of a query

import necessary libraries (pandas, numpy and csv)
read in the utility matrix hybrid
initialize a list to store the utility values
loop through the queries
  for j in range of number of queries:
    initialize lists to store the ratings for the target users and external users
      textbfloop through the ratings for each user
        store the rating for a target user
        store the rating for a external user
    compute the total number of ratings estimated by the algorithm for the query
    compute the total number of "true" scores for the query
    compute the utility for the current query
print the utility values for each query
define header for the CSV file
convert the utility_values list to a list of lists
write the results to a CSV file

Then, we evaluated the performance of our solution, considering again the synthetic and real datasets and computing for them the time required and the memory usage. We observed that:

(1) Regarding the time required, the synthetic dataset performed significantly better than the real one (see Figure 15 and Figure 16). Recalling table 2, we hypothesize that this happened due to the less number of queries of the synthetic dataset than the real one.
(2) Consenquently, for the memory usage, anew the synthetic dataset performs better than the real one (see Figure 13 and Figure 14).

Then we made anew the comparison between the synthetic dataset where N>P (recall table 2 where was N=20,000 and P=1,000)

Figure 13: Memory usage for the computation of the queries utility for the Synthetic dataset

Figure 14: Memory usage for the computation of the queries utility for the Real dataset

Figure 15: Time (seconds) required for the computation of the queries utility for the Synthetic dataset

Figure 16: Time (seconds) required for the computation of the queries utility for the Real Dataset

and synthetic dataset N<P (recall that N=1,000 and P=10,000). In this case we noticed that:

(1) Regarding the time we observed that when N<P our solution performs slightly better (see Figure 15 and Figure 17). We considered this an unexpected result, especially because of the great number of queries (P=10,000), whose utility is to be estimated.
(2) For the memory usage we caught that when N<P our solution performs significantly better (see Figure 15 and Figure 18). As the previous point, we considered this as a startling result due to the great number of queries (P=10,000), whose utility is to be estimated.

6 CONCLUSION

Nowadays recommendation system are fundamental in order to suggest results more consistent with users' tastes, increasing their satisfactions and thus, augmenting companies profits. In this paper we focused on books recommendation system, proposing a new type of hybrid solution. This last one was a combination of content based and collaborative filtering user x user, exploiting RMSE measure. We assessed our results through cross validation procedure with the computation of RMSE and MAE. Moreover we compared the results with using a real dataset scraped from good_reads. We would like to conclude with general recommendations to create a recommendation system. These were the bullets points that we tried to follow in all step of this paper:

(1) We promote the use of large dataset in order to train better the algorithm;
(2) We encourage the adaptation of a combination of methods in order to balance each possible advantages and disadvantages. In our case we were aware of the problematics of each approaches and one can try to limit them with the union of different ideas.
(3) We suggeste the use of cross-validation method as an initial step to prevent overfitting problem.
(4) We advised to use external datasets when it is possible in order to assess better the results.
(5) Try different dataset combinations. we are aware of the fact that in the reality, especially, for for the content based approach, one can have more variables than units; by contrast it could happen having more units than variables.
(6) Do not focus on a particular measure of evaluation, but try to assess the goodness of the solution in different ways.

In the end hybrid solutions already exist, however our shared hope is that, with this approach, we have shed new light on the developing of recommendation system.

Figure 17: Time (seconds) required by the algorithm to suggest queries based on the Synthetic Dataset when N<P

Figure 18: Memory used by the algorithm to compute the queries utility on the Synthetic Dataset when N<P

REFERENCES

[1] [n. d.]. Get daily kindle book deals, reader giveaways, and author resources. https://www.thekindlebookreview.net/

[2] [n. d.]. Meet your next favorite book. https://www.goodreads.com/

[3] Hanley James A. and McNeil J. Barbara. 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology* 143.1 (1982), 29–36.

[4] Ihnaini Baha, M. A. Khan, Tahir Abbas Khan, Sagheer Abbas, Mohammad Sh Daoud, Munir Ahmad, and Muhammad Adnan Khan. 2021. A smart healthcare recommendation system for multidisciplinary diabetes patients with data fusion based on deep ensemble learning. *Computational Intelligence and Neuroscience* (2021).

[5] D. Bashir, G. D. Montañez, S. Sehra, P. S. Segura, and J Lauw. 2020. An information-theoretic perspective on overfitting and underfitting, AI:Advances in Artificial Intelligence (Ed.). 33rd Australasian Joint Conference, AI 2020, Canberra, ACT, Australia, Springer International Publishing, 347–358.

[6] Brassington and Gary. 2017. Mean absolute error and root mean square error: which is the better metric for assessing model performance? *EGU General Assembly Conference Abstracts* (2017).

[7] R. Burke. 2007. Hybrid web recommender systems. *The adaptive web: methods and strategies of web personalization* (2007).

[8] Tianfeng Chai and Roland R. Draxler. 2014. Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature. *Geoscientific model development* 7.3 (2014), 1247–1250.

[9] L. M. Collins, J. W. Graham, J. D. Long, and W. B Hansen. 2017. Mean absolute error and root mean square error: which is the better metric for assessing model performance? *Multivariate behavioral research* 29.2 (2017).

[10] M. D. Ekstrand, J. T. Riedl, and J. A. Konstan. 2011. Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction* 4.2 (2011).

[11] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. 2004. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)* 22.1 (2004).

[12] Lun-ping Hung. 2005. A personalized recommendation system based on product taxonomy for one-to-one marketing online. *Expert systems with applications* 29, 2 (2005), 383–392.

[13] Willmott Cort J. and Kenji Matsuura. 2005. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate research* 30.1 (2005), 79–82.

[14] Wang Jiaxing, Sibin Gao, Zhejun Tang, Dapeng Tan, Bin Cao, and Jing Fan. 2021. A context-aware recommendation system for improving manufacturing process modeling. *Journal of Intelligent Manufacturing* (2021), 1–22.

[15] P. Lops, M. De Gemmis, and G Semeraro. 2011. *Content-based recommender systems: State of the art and trends.* Recommender systems handbook.

[16] Refaeilzadeh Payam, Tang Lei, and LIU Huan. 2009. *Cross-validation.* Vol. 5. Encyclopedia of database systems. 532–538 pages.

[17] M. J. Pazzani and D. Billsus. 2007. Content-based recommendation systems. *The adaptive web: methods and strategies of web personalization* (2007).

[18] Sun Yunchuan, Mengting Fang, and Xinyu Wang. 2018. A novel stock recommendation system using Guba sentiment analysis. *Personal and Ubiquitous Computing* 22, 3 (2018), 575–587.

[19] E. Çano and M. Morisio. 2017. The adaptive web: methods and strategies of web personalization: A systematic literature review. *Intelligent Data Analysis* 21.6 (2017).