Evaluating Dataset portions based on query logs

Leonardo Vicentini
232221
leonardo.vicentini@studenti.unitn.it

Luca Vian
238744
luca.vian@studenti.unitn.it

1 INTRODUCTION

We live in a world in which the amount of digital information that surrounds us is so high that we need to use some recommendation systems everyday in order to filter the ones that could be interesting for us. We need to use these technologies so frequently that we almost take for granted their presence in all the devices and platforms that we use during our days.[6]

A **Recommendation System** is a tool that is able to filter and prioritize the relevant information for users, based on their personalized basis. This process allows to reduce the information load that the user will receive after having learnt, in an implicit and explicit way, their preferences.[10] The functionalities provided by a well-developed recommendation system could be helpful both for the platform that host it, because it will increase in traffic, and for the users, that will be more satisfied.[7]

Many businesses' revenues are directly related to the quality of their recommendation system. For example, search engines like Google are interested in proposing web pages that could be found meaningful by the users. Other services instead, like Amazon and AirBnB, are interested in proposing products and apartments based on the users' preferences. Finally, other services, such as Netflix and Spotify, propose a well-developed recommendation system in order to have a competitive platform useful for growing a large user base.

1.1 Netflix

Netflix, indeed, represent one of the most successful examples of recommendation systems: its movies are proposed by considering some **attributes** that they have (like information about the titles, such as their genre, categories, actors, release year, etc) together with the **habits that a certain user has** (the time of the day when a user accesses the service, the average length of his/her stay in the platform, and so on). The preferences of an user are obtained both in a directed way, for example by understanding what the users like using their ratings or requesting some feedback, either in a non-directed way, analyzing the users' habits.[9]

The data that Netflix's recommendation system uses are very different, and as a result, they must be treated and processed differently: there are data that can change or stay the same over time, that can be discrete or continuous, and that can be known or unknown. Typically, the unknown and unfixed information pertains to the users.

1.2 Challenges behind a recommendation system

As illustrated, the development of a recommendation system could be very challenging for many reasons independently by its scope and scenario. The key problems are related to the lack of data collection and to the fact that both the information fields that could be considered relevant, as well as their values, may change over time.[8] Furthermore, how to use those data at disposition is quite complex and not intuitive, so only a few companies can provide a very high level of user satisfaction with their recommendation systems.

To those difficulties another one is added, which is related to dealing with those information in an **efficient** and feasible manner, not obtainable with naïve approaches.

It should be noted that there are no issues with cold starts in the scenario under consideration: cases where new items or users are added are not within the scope of the problem addressed by this work.

1.3 Approaches to tackle this task

Throughout the years, research has focused on the complications associated with the recommendation task in order to find the best approaches and techniques for dealing with it. The main ones can be divided into three major categories:

- Content-based systems
- Collaborative filtering systems
- Hybrid recommendations approaches

Each category is characterized by their own main advantages such that, depending on the specific task that must be performed, we can prefer a specific approach with respect to another one.

Content-based systems focus on **properties** of items. Similarity of items is determined by measuring the similarity in their properties.[4] In general, collaborative filtering systems focus instead on the **relationship** between users and items. For instance, the similarity of two items could be determined by the similarity of the ratings of those items by the users who have rated both of them.[4]

Usually some preliminary steps are required to avoid either slow run-time computation and results that are far from the actual preferences of the users. Similarity, for instance, could be used to **cluster** users and/or items into small groups with high similarity.

Most recommendation systems now employ a **hybrid approach** that combines collaborative filtering, content-based filtering, and other techniques. Hybrid approaches can be implemented in several ways, including separately making content-based and collaborative-filtering predictions and then combining them; adding content-based capabilities to a collaborative filtering approach (and vice versa); and unifying the approaches into a single model. [12]

1.4 Our query recommendation system

The context on which we decided to focus on is similar to the Netflix one, even if we have decided to put more emphasis in recommending some queries related to a set of movies instead of recommending the movies themselves. In other words, we wanted to compute how much a set of queries identifying one or more movies characterized by **discrete** and **continuous** attributes could satisfy a user's preferences. The most effective approach we were able to obtain was composed of two different Collaborative Filtering components weighted according to **query result cardinality**.

Our tool's ultimate goal is to be able to complete the tasks outlined in section 2 regardless of the domain being analyzed. In other words, if "MOVIES" is the current domain, "SUPERMARKET PRODUCTS" could be an alternative one. To be able to perform the later formalized tasks correctly regardless of the domain chosen, the tool or set of algorithms must be elastic and make as few assumptions as possible.

2 PROBLEM STATEMENT

We can define the previously mentioned problem more formally, specifically by dividing our task into **two major sub-tasks** or **sub-problems**.

For the first sub-task, the inputs given are:

- A **relational table** RT: where each item i is represented by a row containing some values that characterize specific attributes of that item. Therefore, each row represent a tuple of the shape: $\langle attr1, attr2, attr3, ..., attrN\rangle$. All the fields of all the tuples have a value and there are no NULL values. Formally:
$$\mathbf{RT} = \left(v_{1,i}, v_{2,i}, \ldots, v_{N,i}\right) \mid v_{j,i} \in \mathbf{V}$$
- A **user set** US: where each user u is represented by one row that contains only the id of the user u. Formally:
$$\mathbf{US} = u_i \mid u_i \in \mathbf{U_{ID}}$$
- A **query set** QS: where each query q previously posed is represented by a row. Each row starts with an identifier of the query and continues with its definition composed by conjunction of *"attribute=value"* conditions. For the same query, there couldn't be more than a condition regarding the same attribute. Formally:
$$\mathbf{QS} = \left(q_i,\ \left(a_j, v_j\right)_{j=1}^{n}\right) \mid q_i \in \mathbf{Q_{ID}}, \left(a_j, v_j\right) \in \mathbf{AV}, n \geq 1$$
- A **utility matrix** U: a matrix in which each row corresponds to a user u, each column to a query q and for some user-query pairs (u, q), a satisfaction rating between 0 and 100 is provided. So, $U_{u,q}$ represents the satisfaction rating for user u and query q and the condition $0 \leq U_{u,q} \leq 100$ holds. Formally:
$$N = |\mathbf{US}|$$
$$M = |\mathbf{QS}|$$
$$\mathbf{R} = \{r \in \mathbb{R} \mid 0 \leq r \leq 100\}$$
$$\mathbf{U} = \left\{\mathbf{U}_{u,q} \in \mathbf{R} \cup \{NULL\} \mid 0 \leq u < N, 0 \leq q < M\right\}$$

The goal of the first task is to **fill** the utility matrix's user-query pairs that still don't have a value in a coherent way with the interest of each user. In this manner we could, given that completed matrix C and a user u, return the top-k queries that might be of interest to a specific user u. More formally, the outputs of the first task are:

- A **completed utility matrix** C, where $C_{u,q}$ represents the satisfaction rating for user u and query q, including the previously missing values. Formally:
$$\mathbf{C} = \left\{\mathbf{C}_{u,q} \in \mathbf{R} \mid 0 \leq u < N, 0 \leq q < M\right\}$$
with R, N and M defined like in the case of U.
- A **list of top-k queries** L that might be of interest to a specific user u. Formally:
$$\mathbf{L} = (u, q_i, r_i) \mid u \in \mathbf{US}, q_i \in \mathbf{QS}, r_i \in \mathbf{R}, 1 \leq i \leq k,$$
$$\forall x, y \in \{1, 2, 3, ..., k\}, x < y, 100 \geq r_x \geq r_y \geq \mathbf{BR} \geq 0$$
where r_i represents the satisfaction rating for the fixed user u and query q_i, R defined as before. We define the set of good ratings $GR = \{r_1, r_2, ..., r_k\}$ and bad ratings BR such that $\mathbf{GR} \cap \mathbf{BR} = \varnothing$ holds.

After completing the first sub-task, we want to **broaden** the problem in a more general sense: given the elements produced by the previous task as input, we want to find a way to compute a utility matrix for a query in general for all the users.

More formally, the output of the second sub-task is:

- An **array** QU of satisfaction ratings which represents the utility of a query in general, (i.e. **an unseen query**) for all the users in the user set US. Formally:
$$\mathbf{QU} = \left(q, u_i, r_i\right) \mid q \in \mathbf{NQS}, u_i \in \mathbf{US}, r_i \in \mathbf{R}, 0 \leq i < N$$
where NQS is a set of unseen queries (i.e.: $\mathbf{NQS} \cap \mathbf{QS} = \varnothing$), r_i represents the satisfaction rating for user u_i and the fixed query q, R and N are defined as before.

2.1 Our scenario

The inputs and the outputs managed by the tool are organized in **csv** files.

The context that we exploited in order to create a tool for accomplishing the previously formalized task is related to movies.

In the context of this specific chosen domain, the tuples composing the relation table RT are of the shape: $\langle name, genre, runtime, year, country, score\rangle$. Note that *rating* could represent a generic rating given by a review aggregator like Rotten Tomatoes or IMDb.

As a result, the relational table contains some discrete fields (such as genre and nationality) as well as some continuous ones (such as length, publication year, and rating) that are also used in the defined queries. We also assumed that users could have an unspoken preference for the genre, nationality, length, and publication year when generating the partial utility matrix, while there are fields that have no significance (the title) and others that influence the preferences of all users in the same way (the rating).

3 RELATED WORK

3.1 Utility Matrix and Complete Utility Matrix

The concept of **Utility Matrix** is fundamental in order to understand the problem statement and also the solution proposed. An Utility Matrix is a matrix composed of u rows, with u the number of **users** considered, and i columns, with i the number of **items** belonging to the Relational Table taken into account. In the interception between a certain row n and a certain column m there could be a value, to represent the preference score given by the user n for the item m, or not, in the case the user n still have not expressed explicitly a rating regarding the item m.[4] The main goal of a recommendation system regards filling the missing values of this Utility Matrix in order to obtain a complete one, in which the **blank ratings** are deduced in a coherent way by the ones explicitly expressed in the partial one.

3.2 Collaborative filtering systems

3.2.1 Item-Item Collaborative filtering systems. The **collaborative filtering systems**, and in particular the **item-item** collaborative filtering ones, represent the first approach tried in developing the proposed solution and that ended up being an important part of it. Indeed, it was demonstrated that the item-item collaborative filtering approaches represent one of the most valid methods because they work referring to the items' attributes that are much simpler in comparison to the ones that regards the users, that instead can be characterized by multiple tastes.[4] The main idea of this approach, that will be better developed during the **Chapter 4**, is related in finding, for each item, a certain set of a chosen cardinality composed by its most similar ones. In this way, it's possible to assume that an **unrated cell** belonging to the utility matrix will have a rating obtained by combining the known ratings of other similar items in a weighted way [4]. Also the theory regarding the advantages and disadvantages of a Collaborative Filtering System suggested why it would work well in a context like ours. Indeed:

- The main advantages regards its ability to be used **independently** by the nature and the attributes' shape. This aspect is critical in developing a solution that will work well regardless of the dataset proposed.
- The common disadvantages of a collaborative filtering system, for example the Cold Start (the inability of identifying the taste of some users due to the lack of their ratings) and the one related, similarly, to the fact that a newly introduced item will not have any valuation, will not regard our scenario. Those problems indeed often characterize **online contexts**, in which new data are presented over time to the system.

3.2.2 Why a bigger dataset can be helpful in solving our task. The fact that the ratings computation consists in finding the items that are similar to one chosen has a consequence that more will be the data in a dataset, better will be the results found. This works because:

- more are the **items** considered, more probable will be to have, in the dataset, a set of items very similar to any one proposed.
- more are the **users** considered, more will be the number of ratings given by them to the items. This situation is also advantageous in order to find groups of similar items.
- also a **less sparse** utility matrix will be helpful in order to fill it in a better way. Indeed, also in this case, a greater number of known ratings will be helpful in order to understand better the items features, useful to find precisely the more similar items to one chosen.

3.2.3 Approaches to reduce the time complexity. Even if having more data will be helpful in computing a more coherent complete utility matrix, this will also have the **drawback** of analyzing and filling a greater matrix at the expenses of the performance in time. For this reason, some approaches were developed in order to fill an utility matrix with a lower complexity, having as consequence a slightly worse ratings predictions. The main approach used regards the **clustering**, with which is reduced the number of users, of items or of both by considering together the users or the items that are similar between them. This clustering procedure is performed by establishing a distance measure between the users and the items based on the ratings that regards them and, if their distance is under a threshold, it is possible to cluster them and consider them as elements having the same behavior. After having filled the clustered utility matrix, it is possible to expand it by assigning, to each user or item belonging to the same cluster, the rating regarding the cluster itself.

3.3 Hybrid recommendation approaches

A recommendation system that combines the predictions made by different ones is known as a **hybrid recommendation system**. Those approaches are quite popular and used due to the fact that usually it is hard to find a single recommender model that will fit well a given dataset. The solution proposed (Chapter 4) use one of the most popular methods for developing a hybrid recommendation system, which is combining, in a **weighted way**, the prediction performed by two or more different recommendation systems.

3.4 Distances

When creating a recommendation system, different distance measures of similarity can be used in a variety of situations.

3.4.1 Cosine similarity.

$$S_c(A,B) := cos(\theta) = \frac{A \cdot B}{\|A\| \, \|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Cosine similarity is a measure of similarity between two sequences of numbers (vectors).

3.4.2 Centered cosine similarity. Referring to cosine similarity, if the attribute vectors are normalized by subtracting the vector means (e.g., $A - \bar{A}$), the measure is called the Centered cosine similarity and is equivalent to the **Pearson correlation coefficient**.

3.4.3 Jaccard similarity. The Jaccard index, also known as the **Jaccard similarity** coefficient, is a statistic used for gauging the similarity and diversity of sample sets. [2]

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

3.5 Programming language and libraries

The entire solution was developed in **Python**. Mainly, the Pandas, NumPy and Scikit-learn.metrics libraries were leveraged.

4 SOLUTION

The path leading to the final solution was guided by analysis of related topics, reasoning deemed logical, and empirical results.

The proposed solution is a **hybrid recommendation system** leveraging a linear combination of **Expanded-Item-Item Collaborative Filtering** and **Compact Item-Item Collaborative Filtering** based on **query result cardinality**.

Therefore, the solution consists of 3 main components:

- Compact (Standard) Item-Item Collaborative Filtering component
- Expanded-Item-Item Collaborative Filtering component
- Hybridization component

The first 2 components are used to obtain 2 *Complete Utility Matrices*, which are **combined** according to a precise logic (i.e., using the cardinality of the result set of each query).

In the current configuration, the following logic is enforced: if a query's cardinality is high, a greater weight is given to the rating provided by the utility matrix of the compact method; instead, if the cardinality is low, a greater weight is given to the rating provided by the utility matrix of the expanded method. The basic reasoning behind this was the fact that if a query is composed only by few items as result, it is possible to get the queries' ratings from the ones of their results. When, instead, the rating computation regards a query having an higher number of results, the prediction couldn't be computed correctly by starting from the one of its items. This because, summing up together a great number of predictions, each one characterized by a small inaccuracy, will have end up in a result in which all those errors are accumulated.

In the **Compact Item-Item** Collaborative Filtering component, the queries were treated **directly** as items, whereas in the **Expanded Item-Item** Collaborative Filtering component, items are relational table tuples.

Since the solution devised an offline setting an important assumption were made: **heavy pre-computation task** are tolerated. This assumption was fundamental in order to keep our solution valid because of the fact that the intermediate user-item utility matrix, that will be computed and filled in the expanded collaborative filtering approach, has a numbers of cells much higher in comparison to the utility matrix representing the main input of the problem statement. The intermediate results computed, i.e., compact complete utility matrix and expanded complete utility matrix are saved into ".csv" format which are then exploited to execute the two tasks.

4.1 Compact Item-Item Collaborative Filtering component

The Compact Item-Item Collaborative Filtering component was the first one to be developed since it would have been used as starting point for the Expanded version. Moreover, this component is used, as is, as part of the final solution. The core part of this component is an implementation of a **standard** Collaborative Filtering algorithm. In this Compact version, items are the queries previously posed.

The **task** to be performed by this component is the generation of a *Complete Utility Matrix* starting from the *Utility Matrix* as input and filling it with newly calculated ratings where ratings were previously unknown. As a validation proof, the implementation was tested during the development on a small dummy dataset.

To break down the implementation of the previously described algorithm, four major phases can be identified:

- Pre-processing
- Computing similarities between items
- Calculating the ratings
- Generating the *Complete Utility Matrix*

4.1.1 Pre-processing. The **input** that this component leverages is the only *Utility Matrix* which is stored in a `.csv` file. Each row u represents a user, each column q represents a query and each cell (u,q) represents a rating r given by the user u for the query q. If the user u has not expressed his rating for the query q, the corresponding cell is empty.

Once parsed from file, the *Utility Matrix* is transposed and the missing values are replaced with NaNs. We should point out the fact that the transposition was made just to be able to verify that the actual implementation was correct, the operations that will be later described are symmetrically equivalent with respect to rows and columns.

A *Centered Matrix* is generated from the Transposed Utility Matrix as follows: if the cell contains a valid rating (not NaN), the Centered Matrix cell value is the rating minus row mean (which represent the mean rating of the relative query).

Instead, the cells of the *Centered Matrix* containing *NaN* values are filled with the mean of the query ratings (the mean of the row in this case). Also in this case, it was preferred to follow the philosophy of using item characteristics (more synthesizable) versus user characteristics (more complex).

4.1.2 Computing similarities between items. The *Centered Matrix* described above is needed in order to compute the Cosine Similarity between rows, which is actually a **Centered Cosine Similarity**, also known as **Pearson Correlation**.

The similarities between rows are computed once for the entire Centered Matrix, stored in a matrix of similarities:
$similarities\_matrix = cosine\_similarity(centered\_matrix, centered\_matrix)$

This matrix is then needed for the following step of the algorithm: calculating the rating of a cell $(query, user)$ of the *Complete Utility Matrix*.

4.1.3 Calculating the ratings. Exploiting the matrix of similarities, the similarities of the specific query against the others are selected.

Those similarities are sorted in descending order and the top N similar queries are selected (i.e. the ones with the highest values), excluding the query itself. The chosen *TOP_N* value is 2. This step acts as a "neighbor selector".

The formula used to calculate a **rating** is:

$$r_{xi} = \frac{\sum_{j \in N(i)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i)} s_{ij}}$$

where:

- r_{xi} is the rating of user x on query i
- s_{ij} is the similarity of query i and j
- r_{xj} is the rating of user x on query j
- $N(i)$ is the set of queries similar to i

To be as general as possible while making few assumptions, some **edge case handling** is needed, since the density and composition of the *Utility Matrix* U, given as input, is not fixed.

In particular, if a rating r_{xj}, corresponding to a query within the *TOP_N* highest similarities, is missing for the user x in the *Utility Matrix* U, the value considered in the computation is the mean of the correspondent query.

Another edge case that should be handled is the division by zero, that is, when the sum of the top similarities at the denominator of the above formula is zero. Even after the pre-processing, this could happen if the matrix is very small or sparse. The final rating assigned in this case is the mean of the query.

The following step is to round the obtained rating to an integer and ensure that it falls within the range $[0 - 100]$.

4.1.4 Generating the Complete Utility Matrix. The final phase consists of cycling the *Utility Matrix* U and calculating the cell $(query, user)$ rating for each NaN value using the elements described above. Finally, the matrix can be transposed, to have users as rows and queries as columns and saved into a `.csv` file.

4.2 Expanded-Item-Item Collaborative Filtering component

The **Expanded Item-Item Collaborative Filtering** component was the second approach combined, in a weighted way, in the final solution proposed. The main idea of this approach consists in **expanding** the User-Query Utility Matrix given as input, in order to obtain a *User-RelationalItem Utility Matrix*, sparse as well. Once that this last Utility Matrix will be filled with the Item-Item Collaborative Filtering approach implemented as in the subchapter 4.1, it will be possible to reduce it back to a Complete User-Query Utility Matrix, the one requested as output by the problem statement. The implementation of this algorithm can be broken down in three major phases:

- Expansion users-queries to users-items utility matrix
- Item-Item collaborative filtering execution on the users-items utility matrix
- Compression users-items to users-queries utility matrix

4.2.1 Expansion users-queries to users-items utility matrix. First of all, another *Utility matrix* is computed in order to provide better predictions to the ratings that a user will give to the queries that output, as results, a relatively small number of items in comparison to the entire cardinality of the relational table *RT* taken into account. The first step to be done, in order to produce this second Utility Matrix, consists in trying to predict the ratings that every user considered will give to each item belonging to the relational table, based on the **known preferences** expressed by them in the *Utility Matrix U* provided as input of the problem.

In order to do that, at first a query pre-computation is performed in order to increase the performance in time and the modularity of the approach. This pre-computation consists in building a new matrix called *preprocessed_queries* in which each row q represents a query, each column i represents an item of the relational table and, at each cell (q,i), is added a placeholder in order to take track of the case in which an item i is part of the result of the query q. *preprocessed_queries*, so, will work as a list of **BitSet** of the items present in the correspondent query result set.

The *preprocessed_queries* matrix is computed in the way illustrated in Algorithm 1.

Algorithm 1: Preprocessed_queries computation

```
Data: items,queries
Result: preprocessed_queries
preprocessed_queries[][]
for item in items do
    for query in queries do
        is_item_in_query_results=True
        for attribute in query do
            if query[attribute] != item[attribute] then
                is_item_in_query_results=False
                break
            end
        end
        if is_item_in_query_results==True then
            preprocessed_queries[query][item]=True
        end
    end
end
```

Once that the *preprocessed_queries* matrix is computed, a non-complete *users-item_utility_matrix* is predicted; in which each row u represents a user, each column i represents an item and each cell (u,i) represents a rating prediction r of the user u for the item i if the user u has expressed his rating in the utility matrix U for at least a query having the item i as result, or is empty otherwise.

In the case a cell (u,i) contains a prediction and not an empty value, this rating r is computed in a **weighted way** in order to keep it closer to the known ratings that the user u gave to the queries that returned a smaller number of results. This because, when the cardinality of the results of a query is small, the preference of a user for a single relational item will influence more the final rating of the query itself and so, as consequence, those query ratings will be very similar to the ones assigned to the singular items that are composing their results. Each rating r is computed with the following expression:

$$r = \frac{\sum_{q \in iQS} \mathrm{U}_{u,q} \cdot TW \div |preprocessed\_queries[q][:]|}{\sum_{q \in iQS} TW \div |preprocessed\_queries[q][:]|}$$

where:

- iQS is a subset of the query set QS, containing the queries rated by the user u having the considered item i in their results
- $U_{u,q}$ is the rating given by the user u to the query q
- $|preprocessed\_queries[q][:]|$ return the number of results of the query q
- TW is the sum between all the $|preprocessed\_queries[q][:]|$ faced during the computation of the preference score of a user u for a certain item i. It can be computed with the formula:

$$\sum_{q \in iQS} |preprocessed\_queries[q][:]|$$

In pseudo-code, the *users-item_utility_matrix* is computed as illustrated in Algorithm 2.

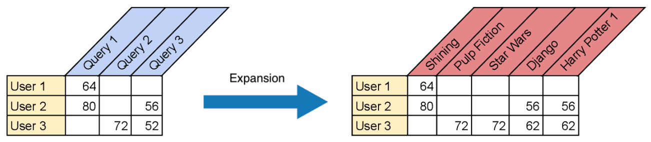
```
Algorithm 2: users-item_utility_matrix computation
Data: users,items,queries,utility_matrix,preprocessed_queries
Result: users-item_utility_matrix
n_results[]
users-item_utility_matrix[][]
for query in queries do
    n_results[query] = preprocessed_queries[query].count()
end
for user in users do
    for item in items do
        for query in queries do
            if utility_matrix[user][query] &&
              preprocessed_queries[query][item] then
                total_weight+=n_results[query]
            end
        end
        for query in queries do
            if utility_matrix[user][query] &&
              preprocessed_queries[query][item] then
                partial_score+=utility_matrix[user][query] *
                  total_weight / n_results[query]
                denominator+=total_weight /
                  n_results[query]
            end
        end
        if denominator!=0 then
            users-item_utility_matrix[user][item]=
              partial_score / denominator
        end
    end
end
```

As it is possible to notice, the *users-item_utility_matrix* returned is **sparse** due to the fact that:

- A relational item could **not appear** as result in any query.
- A user could have **not rated** any query having, as result, a certain item.

In both the cases in which the *denominator* variable used in the Algorithm 2 will be 0, it will have, as consequence, the introduction of an empty cell due to the fact the last *if* condition won't be satisfied. In Figure 1 it is shown an example of the result of the expansion described.

4.2.2 Item-Item collaborative filtering execution on the users- items utility matrix. In this step, the main objective regards filling the *users-item_utility_matrix* previously computed in order to obtain a complete matrix in which each row u represents a user, each column i represents an item and each cell (u,i) represents a rating prediction of the user u for the relational item i.

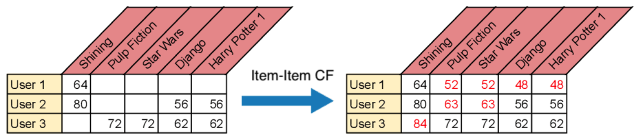


	Query 1	Query 2	Query 3
User 1	64		
User 2	80		56
User 3		72	52

	Shining	Pulp Fiction	Star Wars	Django	Harry Potter 1
User 1	64				
User 2	80			56	56
User 3		72	72	62	62

Figure 1: Expansion Users-Queries Utility Matrix into Partial Users-Items Utility Matrix

To fill the *users-item_utility_matrix* it was used the same **Item-Item Collaborative Filtering** approach described in the subchapter 4.1, but, instead of using the Utility Matrix U as input, were used the *users-item_utility_matrix* itself. In Figure 2 it is shown an example of the result obtained by doing so.



	Shining	Pulp Fiction	Star Wars	Django	Harry Potter 1
User 1	64				
User 2	80			56	56
User 3		72	72	62	62

	Shining	Pulp Fiction	Star Wars	Django	Harry Potter 1
User 1	64	52	52	48	48
User 2	80	63	63	56	56
User 3	84	72	72	62	62

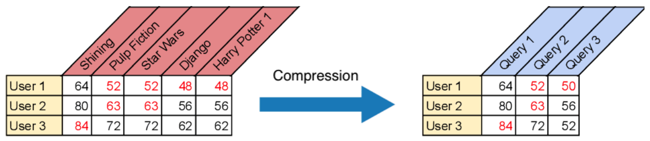
Figure 2: Item-Item Collaborative Filtering applied on the Expanded Utility Matrix

4.2.3 Compression users-items to users-queries utility matrix. Once the *users-item_utility_matrix* was filled, as previously described, it was used to compute the *complete users-queries_utility_matrix*, the requested output of the problem statement. This **compression** phase was performed by taking each user-query combination, by **identifying the items** that will be part of the result set of the query previously considered (referring to the *preprocessed_queries* matrix computed in Algorithm 1) and by computing the **mean** of the ratings of those items for the specific user in consideration. Those ratings are actually read from the complete *users-item_utility_matrix* computed in the subsection 4.2.2. In order to improve both the performance in time of this compression, both its correctness, if a user-query combination was already known from the Utility Matrix U given as input of the problem, those satisfaction values were also reported in the complete *users-queries_utility_matrix* without compute any other operation. In pseudo-code, the complete *users-queries_utility_matrix* is computed as described in Algorithm 3.

The *users-queries_utility_matrix* is actually complete because every user has expressed, in the *users-item_utility_matrix*, a rating toward any item. This has as consequence that every item that compose the result of any query will be rated, and so computing the mean of those items ends up in being a trivial operation. This *users-queries_utility_matrix*, for how it was intended during the expansion part described in the subchapter 4.2.1, is **more reliable** in predicting the preferences regarding the queries that involves a relatively **small amount of items** in comparison to the cardinality of the entire Relational Table given as input. In Figure 3 it is shown an example of the result of the compression described.

Algorithm 3: Complete users-queries_utility_matrix computation

```
Data: users,items,queries,utility_matrix,users-
      item_utility_matrix,preprocessed_queries
Result: users-queries_utility_matrix
users-queries_utility_matrix[][]
for user in users do
    for query in queries do
        item_counter=0
        partial_rating=0
        if utility_matrix[user][query]=="" then
            for item in items do
                if preprocessed_queries[query][item]==True
                 then
                    partial_rating+=
                     users-item_utility_matrix[user][item]
                     item_counter+=1
            end
            users-queries_utility_matrix[user][query]=
             partial_rating /item_counter
        else
            users-queries_utility_matrix[user][query]=
             utility_matrix[user][query]
        end
    end
end
```



	Shining	Pulp Fiction	Star Wars	Django	Harry Potter 1
User 1	64	52	52	48	48
User 2	80	63	63	56	56
User 3	84	72	72	62	62

	Query 1	Query 2	Query 3
User 1	64	52	50
User 2	80	63	56
User 3	84	72	52

Figure 3: Compression Complete Users-Items Utility Matrix into Complete Users-Queries Utility Matrix

4.3 Hybridization

The final solution exploits the two previously computed *Complete Utility Matrices* by both the **Compact Item-Item** Collaborative Filtering and **Expanded Item-Item** Collaborative Filtering components with a linear combination of their ratings driven by **query result cardinality**. More precisely, based on the number of tuples belonging to the result set of a specific query, a different weight is assigned to the ratings of the same cell (user, query) of the two *Complete Utility Matrices* to obtain a new rating. Therefore, this rating takes into account both the rating calculated with Compact and Expanded Item-Item CF components, but with a **different weight** based on how many results are returned by the corresponding query.

In order to count the values of the result set of each query, an intermediate utility file, which is already used for the Expanded Item-Item Collaborative Filtering component and described in Section 4.2.1, is exploited. Leveraging the fact that each row, representing a query, acts like a BitSet of the items present in the correspondent query result set (its value is *True* if the item is present), the cardinality is easily computed. This cardinality, compared with **thresholds**, is used as discriminator for the choice of the **weights**, as illustrated in Algorithm 4.

There are **several ways** to assign weights and thresholds. The most accurate method of assigning them may change due to inherent hidden concepts of the particular dataset, the shape and characteristics of the dataset itself (e.g. mean, minimum, maximum of the result set cardinality). Each of these options needs to be carefully considered on a case-by-case basis in order to select the best way to proceed for the context being examined and finally to be able to fine-tune the parameters.

One path that could be chosen, that is the current one used for the evaluation experiments, consists in giving more weight to the rating provided by the *Utility Matrix* of the Compact method if the query result set's **cardinality is high**; instead, if that **cardinality is low**, a greater weight is given to the rating provided by the *Utility Matrix* of the expanded method. Thresholds must be chosen having a clue of the mean, minimum and maximum value of the queries' result sets of the dataset analyzed, in order to achieve a **coherent splitting** of the 3 possible linear combination. For instance, if the higher threshold is set to a value higher than the widest query result set cardinality, then the first linear combination would be never used in the computation. For the current dataset the mean value of the result set cardinality is 221, the minimum value is 0 and the maximum value is 4696. Therefore the 2 threshold chosen, that are consistent with the characteristics described above, are 200 (lower bound) and 1000 (upper bound). The three weights were chosen in such a way that the combinations did not lean too heavily toward a single approach but were **symmetrically balanced** with one another. Therefore the weights chosen and currently used are: 0.75, 0.50, 0.25 and they are used in a symmetric way as show in Algorithm 4.

The *hybrid_utility_matrix* that can be computed with the above mentioned algorithm, is the *Complete Utility Matrix* requested as **output** in the problem statement.

4.4 Leverage the final solution to solve PART A

The first goal met by the proposed solution is to generate a *Complete Utility Matrix*. Directly related to this first objective, one can use the tool to obtain the *Top-K* queries that may be of interest to the user u. The parameters k and u are therefore the input parameters of this component of the solution, referred to as **PART_A**.

In order to perform the above described task, the user row is scanned in the *Complete Utility Matrix* proposed and the *Top-K* queries are given as output. The viewer can choose whether to receive only new ratings or ratings that were also previously present in the initial (sparse) *Utility Matrix* U.

4.5 Leverage the final solution to solve PART B

The **PART_B** of the project asks to develop a more general solution that is able, given some **unseen queries**, to compute a preference score about them for each user belonging to the User Set US in input. Another way to interpret the task is to fill a new *Utility Matrix*, so such was requested in **PART_A**, composed by the matrix U to which are appended some queries that are not

Algorithm 4: Linear combination of Expanded Item-Item Collaborative Filtering and Compact Item-Item Collaborative Filtering

```
Data: preprocessed_queries, THRESHOLD_1,
      THRESHOLD_2, WEIGHT_1, WEIGHT_2,
      WEIGHT_3
Result: hybrid_utility_matrix
for query in queries do
    results_in_query[query] =
    preprocessed_queries[query].value_counts())
end
for query in queries do
    if results_in_query[query] >= THRESHOLD_2 then
        compact_CF_weight = WEIGHT_1
        expanded_CF_weight = 1 - WEIGHT_1
    else if results_in_query[query] < THRESHOLD_2
    && results_in_query[query] > THRESHOLD_1 then
        compact_CF_weight = WEIGHT_2
        expanded_CF_weight = 1 - WEIGHT_2
    else
        compact_CF_weight = WEIGHT_3
        expanded_CF_weight = 1 - WEIGHT_3
    end
    for user in users do
        hybrid_utility_matrix[user][query] =
          compact_CF_weight *
          compact_CF_utility_matrix[user][query] +
          expanded_CF_weight *
          expanded_CF_utility_matrix[user][query]
    end
end
```

rated by any users, and so that will have their corresponding column composed only by *NULL* values. The solution proposed, that will be described in details in the following subchapters, use the results achieved by the PART_A in order to compute a new *users-RelationalItem_utility_matrix*, whose ratings are combined in order to give a prediction regarding new unseen queries.

4.5.1 Expansion user-item. The first operation done in order to achieve the **PART_B** goal, as previously mentioned, is the computation of a new *users-RelationalItem_utility_matrix*, starting from the output computed by the procedure developed in the PART_A. This expansion follows the same underlying idea described in the subchapter 4.2, even if some differences occurs due to the fact that:

- The matrix used as starting point for the expansion is **complete** and so all the users will have a preference score, in the computed *users-RelationalItem_utility_matrix*, about the same items. The score regarding the relational tuples, indeed, is retrieved by shifting the rating of a query to their results and there are not users, in the matrix obtained in PART_A, that have rated some queries that others won't.
- In the expanded *users-RelationalItem_utility_matrix* there could exists, by the way, some items that still not have a rating: it is indeed possible that some queries present in the Utility Matrix U, and so also in the *Complete Utility Matrix* obtained from PART_A, won't have returned some items belonging to the relational table taken into account. It is also not possible to use any collaborative filtering approach because all the users will have the cells belonging to those items **empty**. In order to fill those users-items combinations in a coherent way, it is assigned, in those cases, a score equal to the mean of the other scores expressed by the user considered for the rated relational items.

The algorithm used in order to achieve this aim is very similar to Algorithm 2 and differs from it only for managing the previous observations, by introducing a new behavior in the case the variable *denominator* will be equal to 0. The pseudocode following those steps is illustrated in Algorithm 5, where the parameter *utility_matrix* requested as input represent the *Complete Utility Matrix* computed in 4.3.

4.5.2 Score computation. Once that the *users-RelationalItem_utility _matrix* is computed, it is possible, given some new unseen queries, to compute their preference scores by:

- **checking** whose items of the relational table represent a result for each given query. This procedure is accomplished by executing Algorithm 1 to which the list *queries*, requested as input, represent the new queries introduced.
- **computing**, for each user u, the mean of the scores that he assigned to the relational tuples composing the results of each new query q given. This result will be the one to be assigned to the cell (u,q) of the matrix in output.

The relative *users-queries_utility_matrix*, computed by executing Algorithm 3 with *users-RelationalItem_utility _matrix* as input, will be the output requested by **PART_B**. In order to replicate the entire procedure we will refer to Algorithm 1 with the new queries as input, for obtaining the BitSet *new_preprocessed_queries*, and to Algorithm 6, for getting the *Complete Utility Matrix* requested by PART_B.

The *users-queries_utility_matrix*, obtained by the execution of the previous algorithms, satisfies the PART_B of the project. Indeed, it has at each row u each user presents in the user set US taken as input, at each column q each new queries proposed and, finally, in each cell (u,q) a prediction about the preference score of the user u for the new **unseen** query q based on the **users history**.

4.6 Further improvements

Clustering and dimensionality reduction techniques could improve temporal performance in the *Utility Matrix*'s pre-computing phase of the current solution. However, as stated in the introduction section of Chapter 4, the proposed solution was assumed to be in an offline context. As a result, on-demand requests for PART_A and PART_B tasks use pre-computed data that is saved on disk in the form of ".csv" files.

4.7 Discarded approaches

4.7.1 User-User Collaborative Filtering. The User-User Collaborative Filtering method yielded **lower performance** metrics for both the compact and expanded versions, as it can be seen is Section 5.5.2. This confirmed what was discovered in the analysis of related

Algorithm 5: PART_B users-RelationalItem_utility_matrix computation

```
Data: preprocessed_queries, users, items,
      queries,utility_matrix
Result: users-RelationalItem_utility_matrix
n_results[]
users-RelationalItem_utility_matrix[][]
for query in queries do
    n_results[query] = preprocessed_queries[query].count()
end
for user in users do
    for item in items do
        for query in queries do
            if utility_matrix[user][query] then
                current_weight=n_results[query]
                total_weight+=current_weight
            end
        end
        for query in queries do
            if utility_matrix[user][query] then
                partial_score+=utility_matrix[user][query] *
                  total_weight / n_results[query]
                denominator+=total_weight /
                  n_results[query]
            end
        end
        if denominator!=0 then
            users-
              RelationalItem_utility_matrix[user][item]=partial_score
              / denominator
        else
            users-
              RelationalItem_utility_matrix[user][item]=
              "PLACEHOLDER"
        end
    end
    rating_mean=0
    item_counter=0
    for item in items do
        if users-RelationalItem_utility_matrix[user][item]!=
          "PLACEHOLDER" then
            rating_mean+= users-
              RelationalItem_utility_matrix[user][item]
            item_counter+=1
        end
    end
    rating_mean=rating_mean/item_counter
    for item in items do
        if users-RelationalItem_utility_matrix[user][item]==
          "PLACEHOLDER" then
            users-RelationalItem_utility_matrix[user]
              [item]=rating_mean
        end
    end
end
```

Algorithm 6: PART_B users-queries_utility_matrix computation

```
Data: new_preprocessed_queries, users, items,
      queries,utility_matrix
Result: users-queries_utility_matrix
users-queries_utility_matrix[][]
for user in users do
    for query in queries do
        partial_query_score=0
        results_counter=0
        for item in items do
            if new_preprocessed_queries[query][item]==True
              then
                partial_query_score+=users-
                  RelationalItem_utility_matrix[user][item]
                results_counter+=1
            end
        end
        if results_counter!=0 then
            users-queries_utility_matrix[user][queries]=
              partial_query_score / results_counter
        else
            users-queries_utility_matrix[user][queries]=0
        end
    end
end
```

works in Chapter 3: Item-Item Collaborative Filtering typically outperforms User-User Collaborative Filtering.

4.7.2 Content-based method. A **content-based** method was also implemented. The initial strategy was to elect it as a method to be included in the final hybrid solution, as a weight portion, but **failed** some preliminary performance tests.

Although a fairly large number of options and alternatives were tested, good performance metrics were not obtained.

Best performances were achieved with the use of "dynamic profiles" that change based on what *user-query* comparison is to be done. Each user profile is composed of its *TOP_Q* values for each possible attribute plus the values present in the query attributes if not already present in the user profile.

5 EXPERIMENTAL EVALUATION

5.1 Dataset composition

The dataset taken into account during the experimental evaluation is composed both by **real** and **synthetic** data organized in ".csv" files. In particular the **Relational Table** *RT* is composed by real data that, in the scenario considered, represents 7669 existing movies. The movies taken into account were retrieved from kaggle.com, a popular collection of public datasets used for data science works. The dataset was moreover cleaned and projected in the tuple $\langle name, genre, runtime, year, country, score\rangle$ in order to make it compatible with the other synthetic data used. The **User Set** *US*, the **Query Set** *QS* and the **Utility Matrix** *U* are instead synthetic and were generated by the team. In particular:

- The **User Set** is composed by a list of 2500 users' ids generated through the use of the *uuid* python's module. Each user has an **unspoken preference** regarding each attribute belonging to the items' tuple in order to generate the utility matrix in a **coherent** way.
- The **Query Set** is composed by 100 queries. Each query is represented by at least a key-value constraints, where each one of them corresponds to existing attribute and value present in at least a tuple of the Relational Table *RT*.
- The **Utility Matrix** is composed by a matrix having u rows, with u corresponding to the number of users, and q columns, with q corresponding to the number of queries. **One third** of the user-query combinations of the matrix is filled with a rating computed by taking into account the unspoken user preferences on the results produced by the execution of the queries on the relational table.

The *Complete Utility Matrix*, composed by the Utility Matrix *U* without any hidden value, was also stored in order to use it only during the **performance evaluation** of the proposed solution and of the ones produced by the baselines considered.

5.2 Dataset generation

The synthetic data of the dataset considered were generated in order to accomplish the attributes and values of the real relational table. Another choice taken, in order to work on a dataset having as **few assumptions** as possible and so whose solution would work in the same way also with other ones, was to do **not** establish any correlation through different distances between the values belonging to discrete fields. For instance, in the current scenario, the Thriller genre could have been considered a closer concept to the Horror genre than the Musical genre.

5.2.1 Fields recognition. In order to generate the data in this way, first of all it was necessary to analyze the kind of attributes belonging to the real relational table and to divide them in:

- **Discrete Attribute** (as the genre and the country)
- **Continuous Attributes** (as the runtime, the publication year and the score)
- **Not-relevant attributes** (as the "name")

In order to do that, the distinction was computed by considering at first the attributes where each of their value could be parsed in a numeric format. Those fields were considered as **continuous**. The non-continuous fields, in addition, were divided in **not-relevant** attributes, in the case they were unique in at least the 75% of the times, and **discrete**, otherwise. During the generation of the dataset it is also given the possibility of specifying manually which continuous fields will influence the user ratings in the same way for everyone, for example in the current relational table the field "score" (like a IMDB score) will influence everyone **in the same way** by increasing or decreasing the rating given to an item independently by the users' preferences.

5.2.2 User profiles. Once the fields were divided in this way, they were leveraged in order to build the users' profiles. Each user's profile is characterized by:

- a **user_id**, which represents the only known field about the user used by the actual solution algorithm.
- an **average_score_translation** field, a random value between -10 and 10 that will indicate if a user is **severe or not** in his rating.
- **10 fields for every discrete field** of the relational table, in order to indicate which are the values for those fields that a user will **appreciate** the most (the first 4 ones) or the least (the last 6 ones). The values not present in the user profile will be simply rated as unpleasant, with a contribution of 60 out of 100 each.
- **a field for every continuous attribute** regarding the items (except for the score one), in order to express the users' unspoken preferences toward a certain continuous item field. Those preferences contains a value picked from a **Gaussian distribution** having as mean the mean value regarding a certain continuous attribute and with a standard deviation such that over the 98,5% of the values will be in the range of values found in the relational table for each attribute. It was chosen to use a Gaussian distribution in order to generate those preferences by also managing the existence of very high or very low values appearing in the big minority of the items considered. If it was chosen to pick some random equiprobable values for computing the users' preferences between the real range of values, a big amount of those picks will be strongly similar to only few (usually one) items of the relational table.

Those users profiles built were used both in the creation of the Utility Matrix *U* belonging to the input of the problem, both during the evaluation performed for the PART_B of the solution proposed.

5.2.3 Query definition. It was then built the definition of the attribute-value constraints belonging to 100 queries that will compose the columns of the utility matrix proposed. In order to reduce the number of queries that won't return any result if executed on the Relational Table *RT*, after having picked a attribute-value constraint referring to a considered field present in the relational table, the other ones were added with 25% of probability each. The combination of those constraints will represent our query definition.

5.2.4 Query result computation. As a pre-computation used during the creation of the Utility Matrix, each query was computed on the entire relational table and its results were saved in a file called **query_id*.csv*. This operation was simply done by iterating over all the attribute-value pairs defined for each query and by writing to file the items that were satisfying all of it.

5.2.5 Utility matrix computation. With all those computations performed, it is actually possible to compute both the *Real Utility Matrix*, a complete matrix containing the ratings of all the users for all the queries, both the *Utility Matrix U*, that represents an **input** of our problem statement. In order to compute them, an iteration through all the users and all the query results achieved was performed and it was assigned, using the unspoken preferences of the users, a rating for each query. The rating of a query by a user corresponded to the **mean** of all the ratings that the user gave to its **results**. The rating given to a single item by a user is obtained by the sum of:

- the ratings given to each **discrete attribute** of the item. Those ratings depend by the position of which the effective discrete value of an item appears in the user profile.
- the ratings given to each **continuous attribute** of the item. Those ratings depend on the distance between the value characterizing an item and the one presents in the user preferences. It was chosen to use an arithmetic distance normalized considering the values' spectrum of each attribute taken into account in order to obtain a preference score between 0 and 100. This distance obtained it was furthermore subtracted to 100 in order to have an higher value when the user profile and the item's one were similar.
- a value indicating how the item in account was **generally** appreciated. This value is represented, in our case, by the *score* field that can also influence negatively the rating assigned to an item. This field was normalized in order to give to it the possibility to add a value between -7,5 and 7,5 to the final score assigned to the item.

After having summed up the previous values, the final result was normalized in order to obtain a value between **0 and 100** by dividing it by the number of the considered items' fields taken into account (excluded the *score* one that was managed in a different way). The case in which a rating was still over 100 or under 0 were shifted respectively to 100 and to 0.

The mean of the ratings, belonging to items composing the results of the same query, will **compose the rating** assigned by a user to the query taken into account. In order to provide more variety between the queries ratings assigned by different users, it was furthermore added to it:

- the *average_score_translation* belonging to each user profile, a value between -10 and 10 used to indicate if a user is **severe or not** in his reviews.
- a **random noise** between -5 and 5, in order to get less predictable results.

Also this obtained rating was shifted in order to get a value between 0 and 100. The pseudocode related to this rating-computation procedure is illustrated with Algorithm 7.

The *compute_discrete(user[field],item[field])* function in 7 checks in which position, if it exists, a discrete value of the relational tuple appears in the user profile. In particular, if a value appears in the first 4 positions of the user profile regarding the attribute considered, the rating given is between 100 and 70, according to its exact position.

Algorithm 7: Utility Matrix and Complete Utility Matrix Computation

```
Data: users,query_results (by reading each
      *query_id*.csv),queries,utility_matrix,preprocessed_queries,
      DISC_FIELDS,CONT_FIELDS,SCORE
Result: utility_matrix,complete_utility_matrix
complete_utility_matrix[][]
utility_matrix[][]
for user in users do
    for query in queries do
        partial_query_rating=0
        for item in query_results[query] do
            partial_item_rating=0
            for field in item do
                if field in DISC_FIELDS then
                    partial_query_rating+= compute_discrete(user[field],item[field])
                end
                if field in CONT_FIELDS && field not in SCORE then
                    partial_query_rating+= compute_continuous(user[field],item[field])
                else
                    partial_query_rating+= compute_score(item[field])
                end
            end
        end
        partial_query_rating+= partial_item_rating /
          (|DISCR_FIELD|+| CONT_FIELDS|-|SCORE|)
        partial_query_rating=partial_query_rating /
          query_results[query].len()
        partial_query_rating+=user["average_score_translation"]
        final_query_rating=partial_query_rating+random(-5,5)
        complete_utility_matrix[user][query] =
          final_query_rating
        if random(1,3)==1 then
            utility_matrix[user][query] = final_query_rating
        end
    end
end
```

If the value instead appears in the last 6 positions, the rating given is poor, between 50 and 0. If the rating regarding a value does not appear in the user profile, it has a **standard contribution** of 60. *compute_continuous(user[field],item[field])* is a function that computes the rating contribution regarding a continuous field that characterize the relational table in this way:

$$100 - \frac{(abs(float(item[field]) - float(user[field]))}{((max(field) - min(field))/100))}$$

So, it computes an arithmetic distance between the user preference and the item value regarding the same continuous field and it normalizes it in order to obtain a value between 0 and 100. It is also computed the complementary of this distance in order to obtain a similarity value.

compute_score(item[field]) normalize the *item[field]* in a value between -7.5 and 7.5.

Finally, because the scores (belonging to the "score" field) in our dataset were between 0 and 5, the relative computation was performed in the following way:

$$(float(item[field]) - 2.5) * 3$$

In this way, it is possible to compute all the ratings belonging to each user-query combination that will be part of the real and complete utility matrix used during the evaluation process. By **randomly keeping only the 33%** of those ratings it is possible to obtain, instead, the Utility Matrix U used during the solution's idealization.

5.3 Dataset independence

The solution provided is **independent** from the default input dataset. This priority in building a general solution was considered since the **dataset generation**, whose algorithm adapts itself based on the type of values present in the **relational table** proposed as starting point. With the same approach, were also developed all the strategies that were tried for both the PART_A and the PART_B of the problem statement such that, every input given with the requested ".csv" representation, will be accepted independently by its fields and its cardinality.

5.4 Evaluation metrics

The evaluation metrics taken into account were:

- **Mean Absolute Error** (MAE)
- **Root Mean Squared Error** (RMSE)
- **Mean Absolute Percentage Error** (MAPE)

5.4.1 MAE. In statistics, Mean Absolute Error (MAE) is a measure of errors between paired observations expressing the same phenomenon. [3] In this particular case, Y versus X is the comparison of real value versus predicted. MAE is calculated as the sum of absolute errors divided by the sample size:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

In other words, its result show how close the predictions (x_i) are to the actual model (y_i) on average. Low MAE values indicate that the model is correctly predicting. Larger MAE values indicate that the model is poor at prediction. [1]

5.4.2 RMSE. The Root Mean Squared Error (RMSE) is the square root of the average of squared errors. The effect of each error on RMSE is proportional to the size of the squared error; thus RMSE is sensitive to **outliers**. [5]

$$RMSE = \sqrt{\frac{\sum_{t=1}^{N} (A_t - F_t)^2}{N}}$$

RMSE is used to determine whether there are any large errors or distances that could be caused if the model overestimated the prediction (that is, the model predicted values that were significantly higher than the actual values) or underestimated the predictions (that is, predicted values less than actual values). [1]

5.4.3 MAPE. The Mean Absolute Percentage Error (MAPE) is another measure of prediction accuracy of a forecasting method in statistics. It usually expresses the accuracy as a ratio defined by the formula:

$$MAPE = \frac{100\%}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|$$

where A_t is the actual value and F_t is the forecast value. Their difference is divided by the actual value A_t. The absolute value of this ratio is summed for every forecasted point in time and divided by the number of points n.

According to some studies, a MAPE less than 5% could be considered as an indication that the forecast is acceptably accurate. A MAPE greater than 10% but less than 25% could indicate low, but acceptable accuracy and MAPE greater than 25% very low accuracy, so low that the forecast is not acceptable in terms of its accuracy. [11]

These 3 metrics were used to **compare** the *Real Complete Utility Matrix* (available right after the dataset generation) with the *Complete Utility Matrix* computed by the final solution (Algorithm 4), and by the others approaches developed.

It should be noted that all of the aforementioned evaluation metrics are only ever used to compare two **full matrices**, one calculated by a developed component and the other created during the dataset generation phase.

In order to evaluate the tasks of PART_A and PART_B of the project, i.e., retrieve the *TOP_K* queries that may be of interest to the user u, **Jaccard similarity coefficient** were used. This index is exploited to compare the **sets** of *TOP_K* queries computed from *Real Complete Utility Matrix* and one computed with the developed algorithms.

5.5 Experimental results accomplished by single components

As said in the introduction of Chapter 4, the entire development process were driven not exclusively by reasoning deemed logical but also by empirical results. In this section, the **key experimental results** are reported. Since the final solution relies on two separate components, it is helpful to examine their outcomes first.

5.5.1 Compact Item-Item CF metrics values. In order to determine the values of the three evaluation metrics mentioned in the previous section, the first experiments were carried out on the Compact Item-Item CF, the first component of the final solution. Tables 1 and 2 contain the results, and figures 4 and 5 contain the relative plotted charts. Regarding Figure 4, it is evident that all metrics considered in the chart reach a **plateau** after considering 200 users, even though the values are still slowly declining. Figure 5 shows a slow decline in the metrics values rather than an initial steep section for the lower number of queries as there is for few users.

The data indicates that in order to reach a performance plateau, it is critical to cross a relatively consistent threshold of users, at least for this dataset. This behavior did not occur in the query variations considered.

Compact Item-Item CF, n_queries (100) fixed			
N_USERS	**MAE**	**RMSE**	**MAPE**
10	4.048	7.6871	8.7731
20	3.7915	6.8982	8.2058
50	3.6162	6.7762	8.3140
100	3.3873	6.2586	7.5723
200	3.3025	6.0730	7.2555
300	3.2853	6.0379	7.2441
400	3.2496	5.9567	7.1688
500	3.2677	5.9896	7.2085
1000	3.2384	5.9402	7.0622
1500	3.2211	5.9151	7.0052
2000	3.1691	5.8260	6.8783
2500	**3.1420**	**5.7890**	**6.8218**

Table 1: Compact Item-Item CF, n_queries fixed

Compact Item-Item CF, n_users (2500) fixed			
N_QUERIES	**MAE**	**RMSE**	**MAPE**
10	3.2684	6.0499	7.5426
20	3.1939	5.9215	7.3000
30	3.1099	5.7911	7.0829
40	3.0310	5.7740	7.2400
50	3.2652	6.0235	7.2698
60	3.3483	6.0666	7.1452
70	3.1616	5.8735	7.0560
80	3.2359	5.9161	6.9690
90	3.2323	5.8874	6.8850
100	**3.1420**	**5.7890**	**6.8218**

Table 2: Compact Item-Item CF, n_users fixed

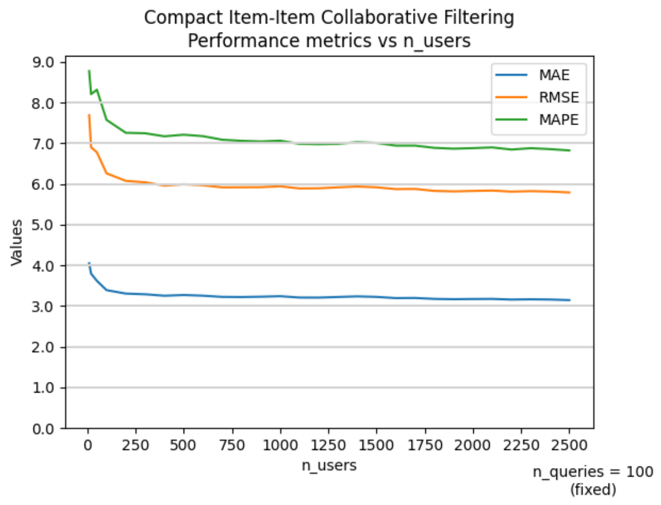


Figure 4: Evaluation metrics of Compact Item-Item CF, varying number of users

5.5.2 *Compact User-User CF metrics values.*

The experimental evaluation of Compact User-User Collaborative Filtering **confirmed** **what the theory says** about this topic, even in this specific setting: Item-Item CF often works better than User-User CF. As shown in Figures 6 and 7, all three metrics considered have significantly higher values than the values described in the previous section.

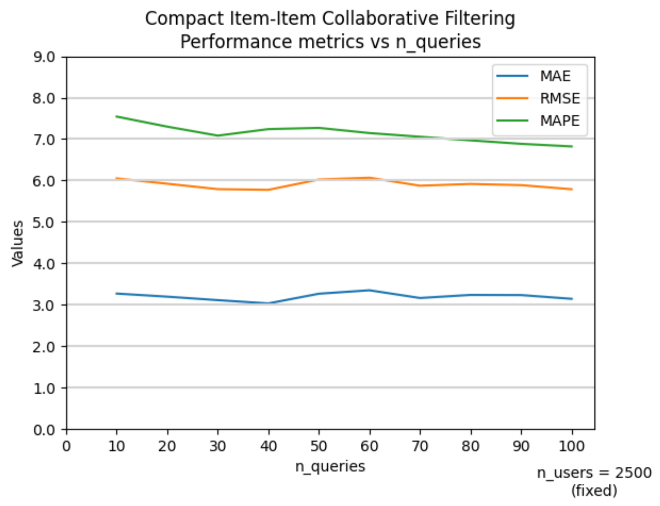


Figure 5: Evaluation metrics of Compact Item-Item CF, varying number of queries

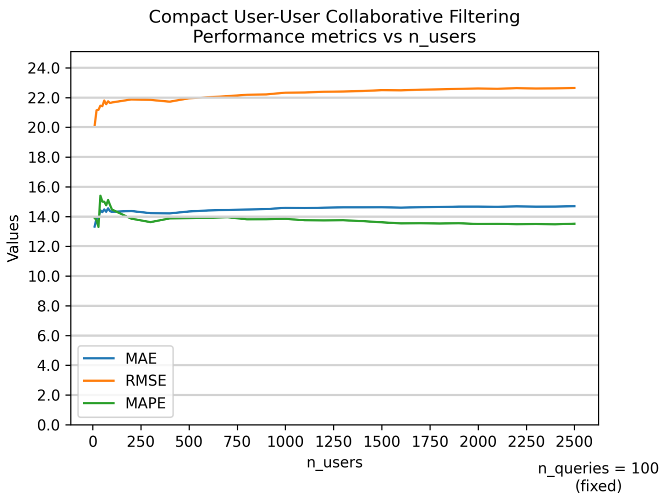


Figure 6: Evaluation metrics of Compact User-User CF, varying number of users

5.5.3 *Expanded Item-Item CF metrics values.*

When tested, this component behaves similarly to the previous ones, but the main difference is that it reaches a performance plateau for all three evaluation metrics at a much **faster** "speed", that is, after only 15 users considered, as shown in Table 3 and in Figure 8 (representing only data considering the first 100 users).

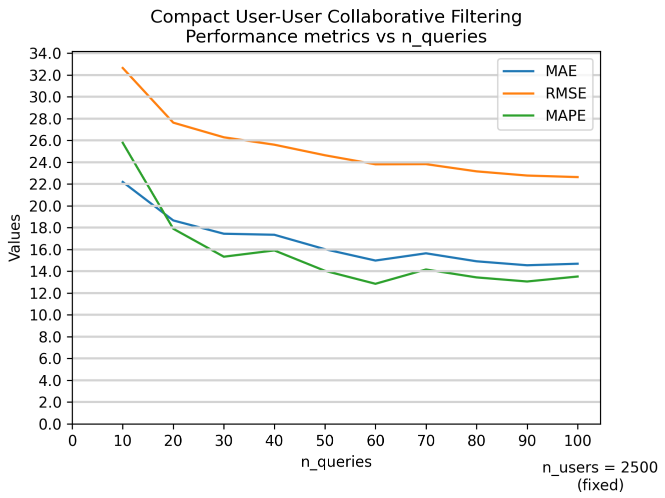


Figure 7: Evaluation metrics of Compact User-User CF, varying number of queries

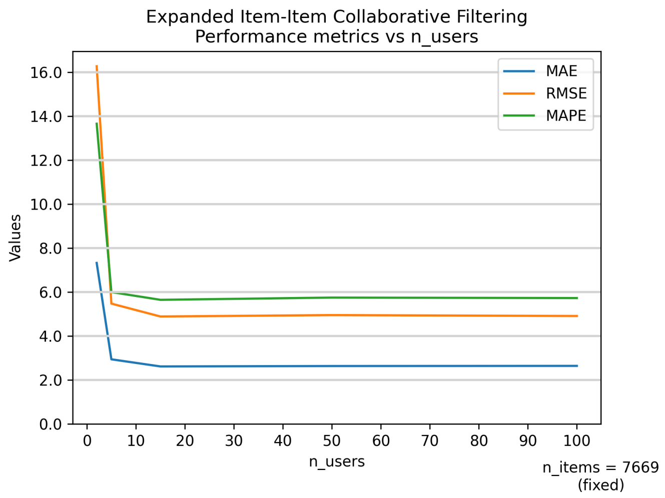


Figure 8: Evaluation metrics of Expanded Item-Item CF, varying number of users

Expanded Item-Item CF, n_items (7669) fixed			
N_USERS	**MAE**	**RMSE**	**MAPE**
2	7.3249	16.2740	13.6591
5	2.94	5.4743	5.9994
15	2.616	4.8884	5.6474
50	2.6342	4.9518	5.7490
100	2.641	4.9109	5.7269
200	2.6540	4.9446	5.7057
1250	2.7587	5.1672	5.9894
2500	**2.7630**	**5.1779**	**5.9939**

Table 3: Expanded Item-Item CF, n_items fixed

5.6 Baseline selection

5.6.1 Part A. Among all the approaches that were implemented during the project development, it was chosen as baseline to be used as comparison meter for the PART_A, the **Compact** (Standard) **Item-Item Collaborative Filtering** component described in the Section 4.1, mainly due to the implementation availability and popularity of that approach in Data Science related works.

5.6.2 Part B. Regarding the baseline chosen in order to allow a comparison with the solution described in the subchapter 5.7.2 for the PART_B, it was decided to expand, as described in the Algorithm 6, the *Complete Utility Matrix* obtained by running the **Expanded Item-Item Collaborative Filtering** on the Utility Matrix U. The obtained Complete *user-RelationalItem_utility_matrix* was used to compute the ratings regarding the new queries.

5.7 Experimental results accomplished by hybrid solution

5.7.1 Part A. To compare the baseline chosen in section 5.6.1 to the final solution, two main strategies were put in place: evaluate their ability in correctly **filling** the Utility Matrix U and, to give the task a more practical meaning, the approaches were also compared in the efficiency in **finding** the *TOP_K* queries belonging to the users of the User Set US considered.

Concerning the ability in filling the Utility Matrix U, it was actually computed, using both the solution and the baseline method, a *Complete Utility Matrix*, which was then compared to the *Real Complete* one, computed as described in the subsection 5.2. Referring to the Table 4, it is possible to see that with the current dataset under consideration (considering N_QUERIES = 100, N_USERS = 2500), the proposed solution **performs slightly better** than the baseline in **all** of the evaluation metrics proposed.

The configuration parameters for the hybrid solution are:

- THRESHOLD_1 = 200.0
- THRESHOLD_2 = 1000.0
- WEIGHT_1 = 0.75
- WEIGHT_2 = 0.5
- WEIGHT_3 = 0.25

PART A - Utility Matrix: Compact Item-Item CF vs. Hybrid			
Method	**MAE**	**RMSE**	**MAPE**
Baseline	3.1420	5.7890	6.8218
Final solution	**2.7446**	**5.1184**	**5.9671**

Table 4: PART A - Utility Matrix: Baseline vs. Final solution

In terms of the second proposed comparison, the *TOP_K* queries from both the *Complete Utility Matrix* computed by the final solution and the baseline are retrieved. After those query identifiers have been stored in **two sets** for each user (baseline, solution), they are compared with the actual *TOP_K* queries set belonging to the *Real Complete Utility Matrix* from the dataset, using the **Jaccard similarity coefficient** as distance. It should be remarked that the higher the value, the more similar the two sets are. This distance is

	Jaccard similarity coefficient, versus real dataset		
TOP_K	**Min**	**Max**	**Mean**
1	C: 0.0	C: 1.0	C: 0.3176
	H: 0.0	H: 1.0	H: 0.3296
2	C: 0.0	C: 1.0	C: 0.2621
	H: 0.0	H: 1.0	H: 0.2814
3	C: 0.0	C: 1.0	C: 0.2504
	H: 0.0	H: 1.0	H: 0.2753
4	C: 0.0	C: 1.0	C: 0.2529
	H: 0.0	H: 1.0	H: 0.2786
5	C: 0.0	C: 1.0	C: 0.2606
	H: 0.0	H: 1.0	H: 0.2862
10	C: 0.0	C: 0.8181	C: 0.2821
	H: 0.0	H: 0.8181	H: 0.3115
15	C: 0.0	C: 0.7647	C: 0.3098
	H: 0.0	H: 0.875	H: 0.3373
20	C: 0.0256	C: 0.7391	C: 0.3436
	H: 0.0526	H: 0.8181	H: 0.3696
30	C: 0.1538	C: 0.8181	C: 0.4309
	H: 0.1320	H: 0.7647	H: 0.4576

Table 5: PART A - TOP-K queries: Baseline (C) vs. Final Solution (H)

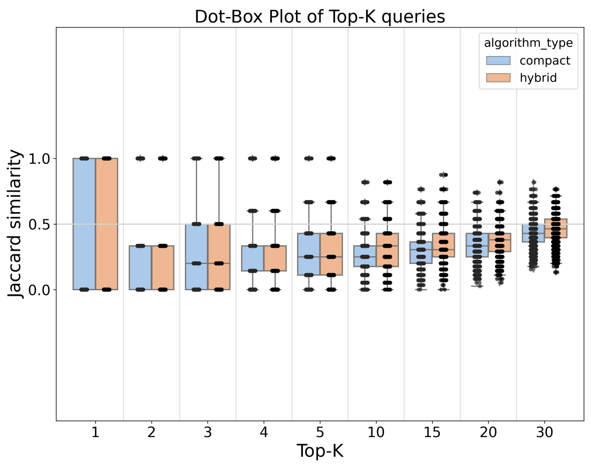


Figure 9: PART A - Jaccard similarity for the Top-K queries sets

calculated considering multiple values of K. The minimum, maximum and mean value of the Jaccard index for each value of K considered are reported in Table 5.

A **Dot-Box plot**, represented in Figure 9, is used to visualize the distribution of the gathered data, showing the median (middle line in the box), 25th quartile (lower edge of the box), 75th quartile (higher edge of the box), and the outliers. The dots, in addition, give a sense of how many data points lie within each group. As we can see in the figure, for the first 5 values of K, the lowest, median and highest values of the Jaccard index are the same for both the baseline and the solution. This suggests that there is basically **no advantage** in using a hybrid approach with respect to a standard one, for low K values, i.e., small sets of highest rated queries. Starting from the value K = 10, median, 25th quartile and 75th quartile values related to the solution are **higher**, as it can be seen by the middle line, lower and higher edge of the boxes. It could be also noted that, for all the values of K considered, as Table 5 shows, the mean of the Jaccard similarity coefficient is always **slightly higher** for the solution.

5.7.2 *Part B.*

Regarding the comparison of the second baseline chosen with our solution, a similar approach to the one described in the subsection 5.7.1 was followed. So, given some new and unseen queries, there were built 3 new utility matrices:

- A *New Real Complete Utility Matrix*, obtained by comparing the results of each new query with the existing users profiles as described in the Algorithm 7.
- A *Baseline Complete Utility Matrix*, obtained by computing a prediction for every user-query combination as described in the Algorithm 5. The intermediate *Baseline User-Relational Item Utility Matrix* was the same obtained by the expansion described in the subchapter 4.2.1 and the users and queries taken into account during the compression were the same of the *New Real Complete Utility Matrix*.
- Another *Hybrid Complete Utility Matrix* built by expanding and then compressing the Utility Matrix computed by our solution described in the subchapter 5.7.1. The expansion was done considering the definition of the original Utility Matrix U being part of our problem statement while the compression was done considering the new queries proposed during the PART_B, as described in the subchapter 4.2.3.

Once obtained those three Utility Matrices, the *New Real Complete Utility Matrix* is compared with the baseline one and with the hybrid one (representing our solution), using the three evaluation metrics described in the subsection 5.4. The results are shown in Table 6 where it can be seen that the final solution has performed **slightly better** in 2 out of 3 evaluation metrics. However, the values gathered at this stage are all very similar to one another.

As done for the PART_A, in order to evaluate the solution in a more practical term, the *TOP_K* queries from both the Complete Utility Matrix computed by the final solution and from baseline are retrieved. After those query identifiers have been stored in two sets for each user (baseline, solution), they are compared with the actual *TOP_K* queries set belonging to the Real Complete Utility Matrix that has been created for PART_B, using the Jaccard similarity coefficient as distance.

As for PART_A, the minimum, maximum and mean value of the Jaccard index for each value of K considered are reported in Table 7. Another **Dot-Box plot**, represented in Figure 10, is used to visualize the distribution of the data. Looking at the latter, the results obtained are **nearly identical** for each value of k considered. One thing to note is that in the case of maximum k tested (25) the 25th quartile associated with the hybrid method has a slightly higher jaccard index than the 25th quartile associated with the expanded method.

PART B - Utility Matrix: Expanded Item-Item CF vs. Hybrid			
Method	**MAE**	**RMSE**	**MAPE**
Baseline	4.0507	6.1362	**8.6332**
Final solution	**4.040864**	**6.1086**	8.6342

Table 6: PART B - Utility Matrix: Baseline vs. Final solution

	Jaccard similarity coefficient, versus real dataset		
TOP_K	**Min**	**Max**	**Mean**
1	E: 0.0	E: 1.0	E: 0.1112
	H: 0.0	H: 1.0	H: 0.1148
2	E: 0.0	E: 1.0	E: 0.1497
	H: 0.0	H: 1.0	H: 0.1544
3	E: 0.0	E: 1.0	E: 0.1725
	H: 0.0	H: 1.0	H: 0.1781
4	E: 0.0	E: 1.0	E: 0.1935
	H: 0.0	H: 1.0	H: 0.1998
5	E: 0.0	E: 1.0	E: 0.2083
	H: 0.0	H: 1.0	H: 0.2120
10	E: 0.0	E: 0.8181	E: 0.2722
	H: 0.0	H: 0.8181	H: 0.2802
15	E: 0.0714	E: 0.7647	E: 0.3596
	H: 0.0714	H: 0.7647	H: 0.3671
20	E: 0.3513	E: 0.9230	E: 0.5917
	H: 0.3513	H: 0.8518	H: 0.5972

Table 7: PART B - TOP-K queries: Baseline (E) vs. Final Solution (H)

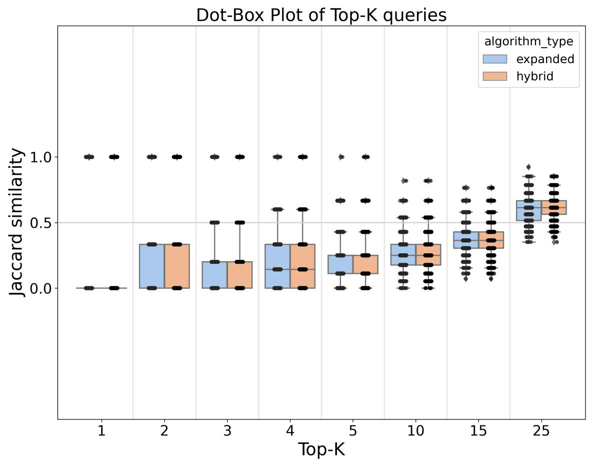


Figure 10: PART B - Jaccard similarity for the Top-K queries sets

6 CONCLUSION

To conclude, the main underlying philosophy followed during the development of all the approaches tried was related in finding a valid solution through iterations, each of which was capable of improving the previously best one found. This process resulted in the discovery of a solution composed of a **hybrid recommendation system** leveraging a linear combination of **Expanded-Item-Item Collaborative Filtering** and **Compact Item-Item Collaborative Filtering** based on **query result cardinality**. There was also the intent of the development of a modular solution. The use of shared components and algorithms between the various described sub-tasks demonstrates this.

The proposed solution was able, at the expense of slower performance, to **slightly improve** in correctness all the other solutions with standard implementations or developed from scratch. The improvement was most noticeable during evaluations in which the *Utility Matrix* filled by the proposed solution for both PART A and PART B was compared, using the **three evaluation metrics**, with the actual *Real Complete Utility Matrix* generated during the dataset construction. In terms of the evaluation of the *TOP_K* queries of each user computed by the proposed solution, performed using the **Jaccard Similarity**, there was almost **no significant advantage** in using the hybrid solution over the baselines.

In conclusion, the project's development was useful in understanding that in order to create a successful recommendation system, it is often necessary to rely on multiple approaches to achieve a better final outcome. There is definitely **room for improvement**, both in terms of the analysis of smart combinations of components and larger-scale testing.

REFERENCES

[1] [n. d.]. Evaluation metrics. https://www.freecodecamp.org/news/evaluation-metrics-for-regression-problems-machine-learning/. last access 26/01/2023.

[2] [n. d.]. Jaccard index. https://en.wikipedia.org/wiki/Jaccard_index. last access 27/01/2023.

[3] [n. d.]. Mean Absolute Error. https://en.wikipedia.org/wiki/Mean_absolute_error. last access 26/01/2023.

[4] [n. d.]. Recommendation Systems. http://infolab.stanford.edu/~ullman/mmds/ch9.pdf. last access 19/12/2022.

[5] [n. d.]. Root-mean-square error. https://en.wikipedia.org/wiki/Root-mean-square_deviation. last access 26/01/2023.

[6] Commons. 2017. The importance of recommender systems. https://medium.com/@Commons/the-importance-of-recommender-systems-36f86f92181. last access 11/12/2022.

[7] Qazi Ilyas, Abid Mehmood, Ashfaq Ahmad, and Muneer Ahmad. 2022. A Systematic Study on a Customer's Next-Items Recommendation Techniques. *Sustainability* 14 (06 2022), 7175 (1–31). https://doi.org/10.3390/su14127175

[8] Richard Macmanus. 2009. 5 Problems of Recommender Systems. https://readwrite.com/5_problems_of_recommender_systems/. last access 11/12/2022.

[9] Netflix. 2022. How Netflix's Recommendations System Works. https://help.netflix.com/en/node/100639. last access 11/12/2022.

[10] Sandeep Raghuwanshi and R. Pateriya. 2019. *Recommendation Systems: Techniques, Challenges, Application, and Evaluation: SocProS 2017, Volume 2.* 151–164. https://doi.org/10.1007/978-981-13-1595-4_12

[11] David A Swanson. 2015. On the Relationship among Values of the same Summary Measure of Error when used across Multiple Characteristics at the same point in time: An Examination of MALPE and MAPE. *Review of Economics and Finance, 5(1)* (04 2015).

[12] A. Tuzhilin and G. Adomavicius. 2005. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. 17, 06 (jun 2005), 734–749. https://doi.org/10.1109/TKDE.2005.99