Data Mining Project 2021/22

Recommendation system for Healthcare

Elisa Paolazzi
Data Science - Year II
elisa.paolazzi-2@studenti.unitn.it

ABSTRACT

Nowadays, an increasing number of decision processes are supported by computerized systems and large amounts of data. These elements may also be involved in the clinical decision-making field. A clinical decision support system (*CDSS*) aims to improve healthcare delivery by enhancing medical decisions with targeted clinical knowledge, patient information and other health data [10].
The aim of this project is to create a system to support medical equipe in suggesting the next most prominent therapy for the treatment of a patient who may have a specific condition. To accomplish this objective, a Recommendation System has been designed. The paper consists of several sections that illustrate the entire implementation process of the proposed solution.
In Section 1, an introduction to the problem is provided. Here, the aim of the project and the relevance of the task it addresses will be better described. Section 2 briefly describes the methods employed and their role. Subsequently, in Section 3, the formal definition of the problem, along with the description of the solution program's input and output, is provided. Section 4 describes the actual solution in detail, illustrating the algorithms and mathematical formulas involved. The next section (Section 5) presents the tools used to implement the solution. Section 6 provides a description of the data and the data collection and generation procedures. Finally, Section 7 illustrates the experimental evaluation, in which our solution will be compared with a baseline method and evaluated in terms of scalability and execution time.

1 INTRODUCTION & MOTIVATION

The aim of the project is to provide a solution to the problem of recommending the most suitable therapies to a patient suffering from a disease. The solution proposed in this paper involves the implementation of a Collaborative Filtering Recommendation System and the construction of a test dataset containing clinical data. These data involve a few key entities:

- **Conditions**, all the possible diseases;
- **Therapies**, all the possible treatments;
- **Patients**, individuals characterised by a medical history.

The patient's medical history is represented by the set of conditions he/she has experienced and the set of therapies (called trials) he/she has undergone in order to treat them. Each patient's condition may have been treated by none or several therapies; these trials have a testing period that ends with the assignment of a success rating. Based on the success rate, it can be determined whether the therapy actually cured the patient's condition.
In these terms, the recommendation system attempts to predict the patient's response to a given therapy based on the similarity of his/her medical history with that of other patients in the data. Ultimately, the system is able to recommend the top-5 therapies on the basis of these predictions.
Solving this task is important as it allows the selection of a subset of therapies (from the pool of possible therapies) that can be considered by the doctor for future patient trials. In this way, the medical equipe does not have to manually analyze each therapy; rather they can immediately evaluate the therapies proposed by the system, leading to a reduction in clinical decision-making and thus patient's trial times.
Furthermore, extending the solution to the use of a large number of real world data would allow the actual performance and reliability of the algorithm to be tested. Therefore, at the root of the solution presented in this paper lies a process of a large amount of simulated data generation.

2 RELATED WORK

In order to gain a full understanding of the implementation process proposed in the paper, it is essential to give a definition of *Recommendation System* and briefly describe some core concepts such as the utility matrix, the similarity metrics and the Weighted Average approach.

Recommendation systems

A Recommendation System is a subclass of information filtering systems that attempts to predict the rating or preference a user might give to an item. The most popular applications of these types of engines are in fact those found in online shops, streaming platforms or blogs, where products, movies or articles are suggested to the user. These recommendations can be based on various aspects such as the popularity of products, the peculiarities of users, their past actions and similarity with other users.
The main approaches in this field are represented by *Content-based* and *Collaborative Filtering* Recommendation Systems [8].
In the first approach, it is essential to build a user profile, which contains the items the user has preferred/bought in the past and their attributes. The system must be able to identify the main features that lead the user to appreciate the item; these features will then be crucial in the selection of items that will be recommended.
On the other hand, the second approach does not concentrate primarily on item features, it rather focuses on the similarity between users based on their peculiarity or previous item-related behaviours. In this context, there are two symmetrical perspectives:

- the*user-to-user*, the user-to-user collaborative filtering, in which the system suggests items that people similar to the user have liked/buy;
- the *item-to-item* collaborative filtering, in which the system suggests items similar to those already included in the profile that were appreciated by the same set of users.

These perspectives enable the problem to be addressed in a different (but symmetrical) manner. In particular, the solution described in this paper implements a Collaborative Filtering Recommendation System using the user-to-user perspective.

This choice stems from the fact that this approach tend to be more flexible and can be adapted to various types of items, which in this particular case are conditions and therapies. Furthermore, in the medical field, it seems to be preferable to recommend a therapy on the basis of how other patients have reacted to it (i.e. similarity between patients), rather than basing it on the few features that a treatment or condition has (in this case only the type) [13].

Utility matrix

In this context, an important structure that allows us to represent the data is the Utility Matrix. This structure contains the appreciation ratings given by users to items. Therefore, in general, the rows of the matrix represent the various users, while the columns represent the rated items. Finally, the cells contain the values of the individual ratings given by a particular user to a particular item; each rating will then be placed at the intersection of the user and the voted item. The structure of the Utility Matrix involved in the solution system is adapted to the entities in the data; it is illustrated in detail in Section 4.

Similarity metrics

All of the approaches presented involve the concept of similarity as one of the final quantitative criteria for deciding which item should be recommended to the user. According to the different methods, similarity with respect to item features, or user characteristics and behaviors, needs to be calculated.
It is worth recalling that similarity is calculated between two elements from the distance between their features in space ($similarity = 1 - distance$). Therefore, various types of similarity metrics are available and those most widely used in the context of recommendation systems are:

- *Jaccard similarity*, useful metric when dealing with two sets (collaborative filtering setting), which calculates similarity considering their intersection divided by the size of their union. However, it is an effective measure in contexts where binary behaviors such as "the item was purchased" or "the item was not purchased" are involved in place of ratings ratings.
- *Cosine similarity*, a metric which considers the ratings assigned to an item. It should be mentioned, however, that this metric fails to discriminate zero ratings of articles from those never expressed by the user (also marked 0). Hence, it is possible to normalise the votes by centring them in the same range (subtracting the row mean). This adjustment of cosine similarity is referred to as Pearson correlation and such variation allows to better capture the intuition by handling the so-called *tough raters* and *easy raters* [14].

The selection of the similarity metric is made during the design phase of the recommender system and according to the type of approach chosen and the nature of the data involved. In the case of the solution presented the similarity employed is the cosine similarity, given its properties.

Weighted Average approach

The last step in the implementation of a recommendation system is to define how to estimate the rating a user will give to a new item by means of the similarity factor (described in the previous section). In fact, in a collaborative filtering context, it is possible to consider an approach in which the rating of the most similar user matters more than the rating of less similar users [14]. This can be achieved by the Weighted Average approach, in which each rating is multiplied by the similarity factor (which indicates how similar users are). With this calculation, weights are added to the evaluations: the greater the weight, the greater the importance of the rating in the prediction. This was indeed the approach used by the solution in the final estimation phase.

3 PROBLEM STATEMENT

In this section, the definition of the formal model on which the solution algorithm is built will be stated. Moreover, the input, output and intention for the output will be specified.

3.1 Recommendation System problem formal definition

Considering the entities presented in the introduction section (1), the formal model can be defined as follows:

- X = set of patients;
- S = set of items defined as all the combinations of each patient's condition and the corresponding therapy(s) tested.

It follows the utility function U:

$$u : T \times D \rightarrow I \times D$$

where R = set of success ratings (from 0 to 100)

Hence, R determines how effective a therapy has been in treating a specific condition. For this reason, it is important to define a threshold that defines whether the condition has been actually cured or not. In this project scenario, the threshold is set to 90. This implies that if the intersection cell between a specific patient and a specific combination of condition and therapy has a value greater than 90, then the condition has successfully been cured by that therapy for that patient.

3.2 Input and output definition

As a result of the formal model definition, the input of the program is represented by:

- the **dataset**, containing the pools of all the possible conditions and therapies, and a set P of patients, their conditions, and the ordered list of trials each patient has done for each of his/her conditions (i.e, his/her medical history/utility matrix based on U);
- a **patient ID**, which medical history has to be already present in the data
- a **condition ID**, which has to be already present in the data

The dataset consists of a JSON file and all records stored within it (conditions, therapies, and patients) are identified by an ID, and therefore it is possible to find a match in the data for the input patient and condition IDs. Moreover, the records in the dataset contain other attributes, which are explained in detail in Section 6.

```
python ther_recommendation.py data.json pat_7 cond_96
```

Example of program input

Given this input, the program will return an ordered list of the top-5 most highly recommended therapies for the input patient and condition.

This output is generated on the basis of the collaborative filtering approach, formally defined as follows:

- consider input patient X;
- find set N of other patients whose success ratings are similar to patient X's ratings (similar medical history);
- estimate patient X's ratings based on ratings of users in N

As a result, the output will display the therapies that have treated patients similar to X; these therapies are therefore characterized by the highest estimates of success rate.

```
 > Recommended therapies for pat_7 to treat
   condition 'Spleen problems and spleen removal':

1^: ther_22
    |  Corticosteroids    |  Medic. and medical aids

2^: ther_57
    |  Psychotherapy      |  Counselling and therapies

3^: ther_36
    |  Home oxygen tr.    |  Medic. and medical aids

4^: ther_32
    |  Hand tendon repair |  Surgical procedures

5^: ther_10
    |  Beta-blockers      |  Medic. and medical aids
```

Example of program output

4 SOLUTION

As previously mentioned, the recommendation system solution developed applies the collaborative filtering approach and operates on a dataset whose specific structure is described in Section 6.

First, as the input includes a path to the dataset file, a patient ID and a condition ID, the system verifies that each of them exists and that the input is therefore valid. Once these verifications have been completed, the empty dataframe that will contain the solution (the ordered list of recommended therapies) is initialized. It consists of the columns of therapy `id`, `name` and `type` and will be filled in the next steps of the algorithm. However, it is important to give a structure that the therapies have to comply with in order to be added to the solutions (have an id, name and type).

The next step investigates whether the input condition has already been diagnosed and cured in the input patient. If the disease has already been cured, i.e. if the success of a therapy applied to it is greater than 90, the data for that therapy (id, name and type) is extracted and will be included among the candidates for the solution.

However, in many situations the disease is diagnosed for the first time in the patient, who has therefore never had treatment for that specific condition. For this reason, it is important to identify the subset of patients who have already treated the disease. This subset must be extracted even if the patient has already had a successful treatment in the past, since the solution has to choose among a set of candidates in order to give more treatment recommendations.

The following procedure represents one of the central steps of the algorithm, namely the calculation of the utility matrix and cosine similarity.

The utility matrix of our data is produced by populating with success rate values the cells of a matrix that has patient ids on the rows and each combination of condition and therapy as columns (Figure 1). However, the algorithm does not obtain the utility matrix for the entire set of patients in the dataset, but only calculates the rows corresponding to the subset of treated patients who actually have a therapy that can be included among the candidates for the solution. Furthermore, in the last step of its construction, the values of the matrix cells are normalised by subtracting the row mean from each value; this ensures that empty cells (filled with 0 values) do not affect the vectors.

	cond_0 ther_0	*cond_0 ther_1*	*cond_0 ther_2*	...	*cond_308 ther_65*	*cond_308 ther_66*
pat_0	0	-2	39	...	-37	0
pat_1	-31.66	-19.34	0	...	-12.34	0
...	...	...	...	...	...	...
pat_9999	0	-7.7	0	...	0	0

Figure 1: Extract of utility matrix

Once the utility matrix is generated, the algorithm calculates the similarity between the input patient and the subset of treated patients (included in the utility matrix). This process is performed using *Pearson correlation* 1, as a Cosine Similarity adjustment (Section 2), which is able to calculate the similarity between pairs of vectors, i.e. the row of the utility matrix corresponding to the input patient and the row of the utility matrix of each treated patient.

$$u(\boldsymbol{x}, \boldsymbol{y}) = \cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{x \cdot y}{\|x\| \cdot \|y\|} \tag{1}$$

where:

- x = input patient utility matrix normalised vector
- y = any other patients utility matrix normalised vector

As a result of this step, we are able to identify the patients who are most similar (in terms of medical history) to the input patient, thus identifying set N of the formal model definition (Section 3.1).

The final step of the algorithm consists of extracting the therapies that cured the most similar patients to the input patient. For each of these therapies it produces a rating estimate based on computed similarity (previous step), involving the calculation of a weighted average (Formula 2) [9].

$$R_{xi} = \frac{\sum_{y \in N} S_{xy} R_{yi}}{\sum_{y \in N} S_{xy}} \tag{2}$$

where $S_{xy} = \text{sim}(x, y)$ (1)

The ratings obtained for each therapy are then reordered (from

the highest).
Finally, for the top-5 therapies with the highest ratings, all information (`id`, `name` and `type`) is extracted. The top 5 most recommended therapies are then identified and they are finally added as rows in the solution dataframe.
The ultimate step consists of formally writing the result into a file, where the solution for the specific input can be consulted.

5 IMPLEMENTATION

This section aims to describe the tools used to implement the solution of the previous section (Section 4).

The programming language used for the development of the solution is Python (in particular version 3.9), as it represents one of the most suitable languages for the implementation of this type of algorithm, thanks to its simple syntax rules, which facilitate code readability [4].
The solution code utilizes both methods already embedded in the language as well as specific libraries. The employment of these libraries enables the use of some dedicated functions that provide a better optimisation of the code and thus a higher efficiency of the algorithm.
It follows the list of libraries involved and a brief description of the specific methods that play a central role in the solution algorithm.

`numpy`, a library that provides support for large matrices and multidimensional arrays along with a large collection of high-level mathematical functions to operate efficiently on these data structures [5]. In particular, vectorisation and matrix definition methods were crucial in the construction of utility matrices and their adaptation to the input required by the chosen similarity metric.

`pandas`, library for data manipulation and analysis which supported the construction of intermediate data structures and facilitated the data values extraction, joining and merging operation [12].

`tqdm`, library used to display smart progress bars that show the progress of Python code execution [3], which allowed the progression of the algorithm's computations to be monitored (both by the developer in the test phase and by the user in the code execution).

`scipy`, library dedicated to mathematical and statistical tools for machine learning [1]. It contains the implementation of the cosine distance calculation used in the solution algorithm (the formula used by the function is given in Section 4). In particular, the function calculates the cosine distance by taking two 1-D arrays as input and returning the result as a double:

```
scipy.spatial.distance.cosine(u, v)
```

For this reason, the solution applies the function in the following way:

```
cos_sim = 1 - scipy.spatial.distance.cosine(x, y)
```

where `x` is the input patient and `y` represents each patients of the treated subset.

Besides these libraries, some built-in Python module, not directly related to the final task, were used:

- `sys` and `os`, used to manage paths and the import of functions between scripts;
- `time`, used to keep track of the time required by the algorithm to solve the task.

Furthermore, it was necessary to write a useful custom function for the construction of the utility matrix: `utility_matrix()` takes as input the dataset and a patient id and populates the matrix, returning as output the normalised row of the matrix corresponding to the input patient.
Some other custom functions have been defined, however they are not directly related to and used by the solution algorithm, but rather by the data generation process. All custom functions, including the one already described, are defined in a dedicated sheet (`utilities.py`).
In addition to that, other libraries and resources have been used exclusively for data creation and are therefore reported in the next section.

6 DATASET

The dataset represents a fundamental element in the development of the solution. This section provides a detailed description of each of its components and the generation process devised for its creation.

6.1 Dataset description

The dataset is provided in JSON format, containing three lists of dictionaries namely Conditions, Therapies and Patients. The first two have a similar structure, which is described as follows.

The `Conditions` list defines pools of possible conditions and its dictionaries are characterised by the keys:

- `id`, generated identifier string
- `name`, the formal name of the disease
- `type`, which identifies the category of the condition

```
"Conditions": [
        {
            "id": "cond_0",
            "name": "Abdominal aortic aneurysm",
            "type": "Heart and blood vessels"
        },
        {
            "id": "cond_1",
            "name": "Acne",
            "type": "Skin, hair and nails"
        }, ...
```

Extract of Conditions list

The `Therapies` list defines pools of possible therapies and its dictionaries are characterised by the keys:

- `id`, generated identifier string
- `name`, the formal name of the treatment
- `type`, which identifies the category of the therapy

```
"Therapies": [
        {
            "id": "ther_0",
            "name": "Abortion",
            "type": "Surgical procedures"
        },
        {
            "id": "ther_1",
            "name": "Amniocentesis",
            "type": "Biopsies"
        }, ...
```


Extract of Therapies list

The third list of dictionaries has a slightly different and more complex structure compared to the previous ones, as it aims to describe the medical history of the set of patients. The dictionaries of the `Patients` list is in fact characterised by the keys:

- `id`, generated identifier string
- `name`, first name of the patient

Besides these keys, similar to those in the first two entities, there are two more keys called `conditions` and `trials`, that introduce two lists of dictionaries:

- `conditions` key introduces a list of dictionaries containing information regarding the diseases that have been diagnosed in the patient over time. These dictionaries contain the following keys:
  - `id`, which identifies the patient's condition;
  - `diagnosed`, indicating the date on which the condition was diagnosed;
  - `cured`, indicating whether, and if so, the date on which the condition was cured;
  - `kind`, which contains the id of the `Conditions` dictionary corresponding to the referred condition.
- `trials` key introduces a list of dictionaries identifying therapies tested on patients to treat diagnosed diseases (contained in the `conditions` key). These dictionaries contain the following keys:
  - `id`, which identify the trial tested on the patient for a specific disease;
  - `start`, indicating the date when the trial started;
  - `end`, indicating the date when the trial ended;
  - `condition`, which specifies for which disease the trial is used, and refers to the id key of the patient's condition key dictionaries;
  - `therapy`, which contains the id of the `Therapies` dictionary corresponding to the referred therapy;
  - `successful`, which contains a value from 0 to 100 identifying the success of a trial in treating a disease.


```
"Patients": [
        {
            "id": "pat_0",
            "name": "Aadam",
            "gender": "Male",
            "conditions": [
                {
                    "id": "c1",
                    "diagnosed": "2003-03-27",
                    "cured": null,
                    "kind": "cond_132"
                },
                {
                    "id": "c2",
                    "diagnosed": "2006-09-12",
                    "cured": "2019-12-31",
                    "kind": "cond_9"
                }, ...
            "trials": [
                {
                    "id": "c1_t1",
                    "start": "2017-07-30",
                    "end": "2018-02-19",
                    "condition": "c1",
                    "therapy": "ther_1",
                    "successful": 21
                },
                {
                    "id": "c1_t2",
                    "start": "2018-06-06",
                    "end": "2018-08-07",
                    "condition": "c1",
                    "therapy": "ther_48",
                    "successful": 46
                }, ...
```


Extract of Patients list

6.2 Data generation process

The project repository provides an already generated dataset containing 67 possible therapies, 309 possible conditions, 10,000 patients medical histories with a maximum of 20 conditions and 12 trials for each condition. This dataset includes enough observations to enable the user to optimally test the code, both from the point of view of results (we have enough patients, with enough conditions and trials to find meaningful similarities) and from the point of view of complexity and execution time (the execution time of the solution algorithm is reasonable). However, it is possible to reproduce the data generation process by choosing new parameters. Then, if the user wishes, he/she can create new datasets (containing more or less data) on which to test the solution, by running the data collection script specifying the path where to save the file, the number of patients to be included in the new dataset, the maximum number of conditions per patient and the maximum number of trials per condition.

```
data_collection data.json 20000 30 16
```

Example of data collection and generation process execution

The data sources used in the creation of the dataset are:

- NHS inform *Illness and conditions* section, from which the conditions data were extracted [6];
- NHS inform *Tests and treatments* section, from which the therapies data were extracted [7];
- `names-dataset` library, from which patients name were obtained [11].

Web page information is obtained thanks to web scraping techniques implemented with the aid of the `BeautifulSoup` library (`bs4`) [2]. Furthermore, the general data manipulation and generation is optimized by `pandas` (Section 5) and `random` (library for random entity creation). Besides that, some custom functions for the date creation (useful for some dictionaries keys field) were defined in the dedicated `utilities.py` sheet. The whole process structure is explained in the following paragraphs.

Conditions and Therapies data collection. The `Conditions` and `Therapies` dictionaries are generated by the same function using a web scraping technique to acquire data from the above-mentioned web pages. A brief description of the conditions and therapies data generation follows:

- first, we obtain the main content of the web page by means of the `requests` library;
- the logic for obtaining the information needed (the list of names and types of conditions and therapies) is then defined:
  - each item in the web pages lists of conditions and therapies has a link to the detail page, where type data can be found;
  - the lists are then scraped by means of the `BeautifulSoup` library and the data is stored in a dataframe;
  - the last step is the generation of the ids, which are created by joining the strings `'cond_'` for conditions and `'ther_'` for therapies to the index number from the order in which the records appear in the temporary dataframe (starting from 0).

Using the web page scraping technique it is common to encounter errors, such as missing pages or data, however the process is not interrupted by such errors, but reports them to the user.
At this point, the data is cleaned by removing condition or therapy records that have missing types. The cleaned dataframe is finally converted into a dictionary, yielding the result shown in Section 6.1 (for both conditions and therapies).

Patients data collection. The data collection and generation for the `Patients` dictionary is handled by two different functions: the first collects patients' names while the other generates the medical history of each patient by assigning conditions and therapies.
The first function exploits the `name_dataset` library using the built-in dataset pool for patients name extraction. As for conditions and therapies, the data is stored in a temporary dataframe and the id is generated by joining the string `'pat_'` to the order in which the records appear in it. This is then converted into a temporary dictionary by adding the conditions and trials keys as empty lists for each patient.
The empty lists of the `conditions` and `trials` keys are actually filled by the second function. The description of this function, which represents the core of the data creation process, follows.

It takes as input the `Patient` list of dictionaries, the list of `Conditions` IDs, the list of `Therapies` IDs, the maximum number of conditions per patient, the maximum number of trials per condition.

It modifies the `conditions` key's values:

- choose a random number N between 1 and the input maximum number of condition and create N conditions for that patient,
- for each condition:
  - create the `id` key (combining the string `'c'` with the number corresponding to the order in which they are drawn),
  - create a random diagnosed date for `diagnosed` key,
  - set `cured` key date as `NULL`,
  - choose a random condition ID from `Conditions` temporary dictionary, in order to fill the `kind` key.

It modifies the `trials` key's values:

- for each condition generated:
  - choose a random number N between 0 and input maximum number of trials and create N trials for that condition,
  - for each trial:
    - create an `id` (same procedure as for conditions but using the string "`t`" added to the corresponding `condition id`),
    - create `start` key date, which has to be greater than corresponding condition's `diagnosed` key,
    - create `end` key date, which has to be greater than `start` key,
    - store corresponding condition `id` key in `condition` key,
    - choose a random therapy ID from `Therapies` temporary dictionary, in order to fill `therapy` key,
    - choose a random number between 0 and 100 for `successful` key, if this number is greater than 90, the condition `cured` key date is updated with the `trial end` key date.

This algorithm will then populate the condition and trail keys for each patient also using some custom functions for dates generation (contained in the `utilities.py` file). Moreover, the process described is based on the following constraints:

- each patient has at least one condition, but a condition can have no trials (no therapy has yet been tested);
- each patient's conditions are ordered (the `diagnosed` date of the later condition is greater than the `diagnosed` date of the earlier one);
- each trial `start` date is greater than the corresponding condition's `diagnosed` date;
- each trials for the same condition are ordered (the `start` date of the later condition is greater than the `start` date of the earlier one);
- if the trial for a condition is successful (`successful` key's value is greater than 90), no further trials are generated for that condition.

7 EXPERIMENTAL EVALUATION

The experimental evaluation of the algorithm was carried out on a machine equipped with the following components:

- Intel Core i7-1065G7 processor
- DDR4-RAM 2667MHz 1x8GB
- SSD Intel 512 GB PCIe NVMe 3.0

The dataset used for the following tests is the one already present in the repository, which is described in Section 6. The algorithm was tested many times for several combinations of patients and conditions: the results of 15 test cases are saved in the `test.txt` (`results` folder), while some results of single-run tests are stored in the `results.txt` file (`bin` folder). With respect to the test cases, the algorithm works correctly and returns the 5 recommended therapies based on the input patient and condition in a reasonable time. In these tests, the running time of the algorithm varies between 19 and 30 seconds, and depends mainly on the number of similarity factors to be calculated.
Considering the artificial and random nature of our data, it is not straightforward to assess the performance of the algorithm in terms of the accuracy of the output produced. However, this problem is beyond the scope of this paper; other kinds of testing were performed, which will be explained in the following sections.

7.1 User evaluation

In order to conduct a user evaluation, the repository containing the algorithm code, the data generation scripts and the `README`

file (which provides the running instructions) was distributed among a group of 8 people. In particular, these users are represented by students of Computer Science, Human Computer Interaction and Communication Sciences, all of whom have good or discrete skills in handling the tools required for code execution.
None of them had any significant problem reproducing the data generation and executing the solution code, even though different machines and operating systems were used. Most of the group stated that the `README` instructions were, overall, clear and represent an essential support for the correct execution of the code. Two of the users had some difficulties with virtual environment creation and libraries installation. However, these users reported that they were quickly able to find a solution by doing a brief web searching.
Considering these difficulties and taking into account the usage scenario of the solution, in which users as doctors and nurses may not be confident with code execution, the design of a web application with an intuitive graphical interface would be appropriate. Furthermore, users noticed the slow execution of the data generation code, which is mostly delayed by the use of web scraping technique. A possible solution to the problem would be the parallelisation of web pages information retrieval which could potentially optimise the procedure.

7.2 Baseline method evaluation

An additional evaluation performed was the comparison of the proposed solution algorithm with a baseline method. As the model proposed in this paper, the considered baseline method implements a recommendation system using a collaborative filtering approach. The difference between the two models lies in the construction of the utility matrix and the calculation of similarities: whereas the solution only calculates the rows of the matrix of interest for a given input and computes their similarity to the input patient, the baseline method always precomputes the entire utility matrix and the similarity between the input patient and the whole set of patients.
The two programs were then executed 10 times monitoring their running times with identical inputs. In particular, the dataset considered was the one present in the repository (containing 10k patients) described in Section 6. Given the same input, the models return the same output. However, the execution time of the basic method is significantly slower. In fact, it reaches an average of 6.5 minutes compared to 22 seconds for the proposed solution (Table 1). This makes the baseline model practically infeasible on larger datasets.

Method	Avg running time (in seconds)
solution algorithm	22,3
baseline method	392,4

Table 1: *solution algorithm* and *baseline method* execution time

7.3 Scalability and execution time

The last evaluation conducted was a scalability test of the algorithm. The solution was run several times on an increasing sample of patients. For this purpose, 5 datasets containing different numbers of observations were generated, keeping the maximum number of conditions per patient and therapies per condition fixed at 5 and 10 respectively for all of them. The execution time on each dataset was then monitored over 10 executions each and averaged (Table 2).

The cost of this computation actually depends on how many observations (patients medical histories) are present in the data. The running time increases dramatically with a large number of patients. This is quite noticeable when observing the difference between the blue and orange lines in Figure 2; the latter represents a hypothetical linear growth scenario of computation. While for smaller datasets, the time varies by a few seconds, as the number of observations increases, the difference becomes a few minutes and more. This is natural since with increasing numbers of data, the algorithm has to calculate more similarities and compare more values.

Number of patients	Avg running time (in seconds)
1,000	0.546
5,000	2.703
10,000	7.485
25,000	89.203
50,000	331.078
100,000	1071.21

Table 2: Time of execution based on patients number

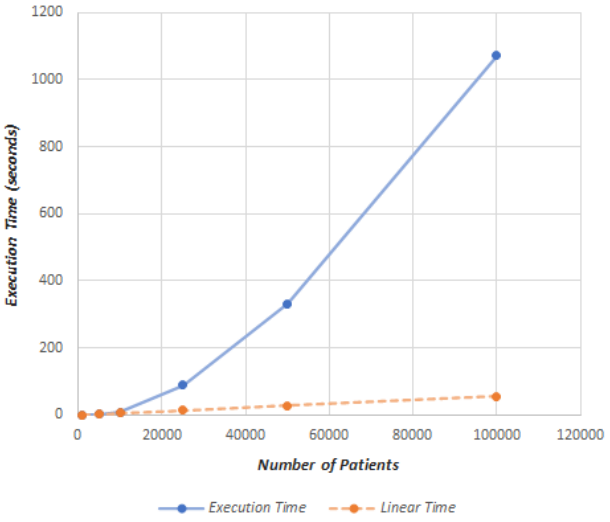

Figure 2: Time of execution based on patients number

Nevertheless, having a large sample to base decisions on is very important in the medical field and unfortunately the proposed solution appears not to be optimised for very large datasets (Big Data) in terms of execution time. For this reason, the algorithm should be further enriched with techniques such as Locality Sensitive Hashing (LSH), Clustering or Dimensionality Reduction, which represent clever ways of enabling much faster similarity computations and comparisons.
In addition to such possible extensions, it would also be worthwhile to consider the implementation of more complex recommendation systems algorithms such as the item-to-item approach

(Section 2) or the Global Baseline Estimates, which combines Content-based and Collaborative Filtering approaches to produce a very powerful framework. Finally, to enable the deployment of such alternatives, it would also be necessary to increase the number of attributes of each entity, in order for the items to have more features.

REFERENCES

[1] The SciPy community. [n.d.]. SciPy documentationn. https://docs.scipy.org/doc/scipy/. Accessed: 24-06-2022.

[2] Leonard Richardson contributors. [n.d.]. Beautiful Soup Documentation. https://github.com/philipperemy/name-dataset. Accessed: 01-05-2022.

[3] Casper da Costa-Luis contributors. [n.d.]. tqdm documentation. https://tqdm.github.io/. Accessed: 24-06-2022.

[4] Maria Weinberger Dataconomy. [n.d.]. Why you need Python machine learning to build a Recomendation System. https://dataconomy.com/2018/01/need-python-machine-learning-build-recommendation-system/. Accessed: 02-06-2022.

[5] NumPy Developers. [n.d.]. NumPy documentation. https://numpy.org/doc/stable/index.html. Accessed: 24-06-2022.

[6] NHS inform. [n.d.]. Illiness and conditions. https://www.nhsinform.scot/illnesses-and-conditions/a-to-z. Accessed: 24-06-2022.

[7] NHS inform. [n.d.]. Tests and treatments. https://www.nhsinform.scot/tests-and-treatments/a-to-z. Accessed: 24-06-2022.

[8] Anand Rajaraman Jeffrey D. Ullman Jure Leskovec. [n.d.]. *Mining of Massive Datasets.* 2021.

[9] Abhinav Ajitsaria Real Python. [n.d.]. Build a Recommendation Engine With Collaborative Filtering. https://realpython.com/build-recommendation-engine-collaborative-filtering/. Accessed: 25-05-2022.

[10] Daniel C. Baumgart Daniel C. Sadowski Richard N. Fedorak Karen I. Kroeker Reed T. Sutton, David Pincock. [n.d.]. *An overview of clinical decision support systems: benefits, risks, and strategies for success.* npj Digit. Med. 3, 17 (2020), https://rdcu.be/cQXDA.

[11] Philippe Rémy. [n.d.]. name-dataset. https://github.com/philipperemy/name-dataset. Accessed: 01-05-2022.

[12] the pandas development team. [n.d.]. pandas documentation. https://pandas.pydata.org/docs/. Accessed: 28-06-2022.

[13] Christoph Trattner Andreas Holzinger Thi Ngoc Trang Tran, Alexander Felfernig. [n.d.]. *Recommender systems in the healthcare domain: state-of-the-art and research issues.* J Intell Inf Syst 57, 171–201 (2021). https://doi.org/10.1007/s10844-020-00633-6.

[14] Leland Stanford Junior University. [n.d.]. Recommender Systems: Content-based Systems Collaborative Filtering. Yannis Velegrakis, Data Mining course slides.