# Therapy Recommendation System

Collaborative Filtering based Recommendation System to advice the best possible therapies

Davide Piombini
University of Trento
davide.piombini@studenti.unitn.it

Master's Degree in Data Science
Academic year: 2021-2022
Mat: 227956

## ABSTRACT

This report explains the process by which the Recommendation System is built. Given a dataset of patients ans their medical history, the goal is to create an algorithm that advices the best five therapies.
To do that, the idea is to create a collaborative filtering user-based system using similarity/distance to give "weight" to every possible therapy.

## DEFINITIONS

Before starting to analyze the problem, is important to specify some definitions:

- **Target patient**: the patient that has to be adviced with a therapy
- **Target condition**: the condition that the target patient has to cure

## 1 INTRODUCTION & MOTIVATION

We live in a Data driven world in which data are in the center of most important decisions. We use data to predict meteorological phenomenon, rivers water height levels, or most recently the spread and evolution of the pandemic. Data collects the history and through that it is possible to try to estimate the success of every possible statistical event.
In our case, data are used to predict the best 5 therapies for a given condition of a known patient.
The aim of this reccomendation system is to be used to work alongside (and not to replace) doctors. In fact, doctor have surely a better knowledge and experiece that an algorithm, but as humans, they cannot compare the subjects clinical history with an (ideally) complete dataset of other patient to know what worked with them and what doesn't.

## 2 RELATED WORK

This section is a general overview of the theory at the base of the choosen implementation of the algorithm. In particular, we focus on Recommendation systems, their two possible implementations and pros and cons of the three similarities generally used for recommendation systems.

### 2.1 Recommendation Systems

Recommendation Systems are machine learning engines that are used to advice the best possible items to a user. To do that, recommenders try to predict a possible **rating** that a user would give to a certain item, such as movies (Netflix), songs (Spotify) or everything else, using his/her or others previus rating to similar (or the same) items.
There are two main approaches to build recommendation systems, that differ in the focus of the algorithm:

- *Content-based systems*:
  these type of systems have the peculiarity that do not care about any other user but the user itself. In fact, they try to recommend an item to a certain costumer based on the rating the same costumer gave to similar items.
- *Collaborative filtering systems*:
  these type of systems, instead, require a lot more data on other users. The idea is to find a set of user that rated similarly to the target the same objects, and try to estimate target's rating based on a weighted average of other user's ratings, where the weight, in this formula, are similarities.

### 2.2 Similarities

Similarities, in general, are a measure of how something is closer to something else, in our case, items or users. To have a specific metric to understand and compare similarities, it is also important to talk about **distances**, where a small distance mean high similarity and viceversa, so the formula is:

$$s = 1 - d \qquad (d = 1 - s) \tag{1}$$

There are three main types of similarities:

- *Jaccard similarity*:
  Jaccard similarity is the most "rudimental" of the three because it does not care about ratings. In fact, Jaccard similarity is based on counting how many times two users have rated the same item, on the total of items rated by the users. So, in this case the similarity between two users is:
  $$sim(A, B) = |r_A \cap r_B| / |r_A \cup r_B|$$
- *Cosine similarity*:
  Cosine similarity use the cosine distance between two items in the space.
  $$sim(A, B) = \cos(r_A, r_B)$$
  In this case, ratings are obviously considered so how much a user gave to an item care more that the fact that both user rated the same item. This makes cosine similarity more precise than Jaccard's one, but also this one has a problem: it treats missing ratings as negative, meaning that not having seen a film is considered worst than having rated it 1 out of 5 stars.
- *Centered Cosine similarity* (also known as *Pearson Correlation*):
  Centered cosine similarity solve the problem of the previous one **normalizing** ratings, in order to treat missing values as average. To normalize, every rating is subtracted by the mean of the ratings of the same user:
  $$r_{Ai} = r_{Ai} - mean(r_A) \quad \forall r \in r_A$$

## 3 PROBLEM STATEMENT

The idea is to create a Recommendation System that advices the best 5 therapies for a given condition to a patient. To do that, it requires a pretty large dataset of **patients**, each with his/her set of **conditions**, namely their clinical history, and a set of **trials**, that are all the therapies previously adviced for each of the conditions.

## 4 SOLUTION

Before talking about the actual solution chosen to have the best possible therapies adviced, is important to specify also why a possible option was then discarded and motivate it.

### 4.1 Why not clustering?

The initial idea was to use clustering to reduce the dimensionality of the problem using only the cluster in which the target patient, namely the patient that requires a new therapy, would have been located. But two "problems" were found:

(1) **Not mandatory variables**
All the variable that were taken into consideration for clustering, such as age and sex, are not mandatory variables in the assignement. Hence, as the general idea of the project is to create an algorithm that works with every possible dataset configuration (as soon as all mandatory variables are correctly in the dataset itself), the decision was to not base our algorithm on those variables that, not being present, would have cause a casual clustering that surely would have discarded important informations.

(2) **Another way to reduce dimensionality**
The most important point that lead to the exclusion of a clustering is that this is a particular case of recommendation system.
To be more specific, instead of using clustering to "skim" relevant patient, it is better to take in consideration only patients that has cured the target's condition the therapy is for.

Now that the decision on clustering is explained, from now on what is written is the actual solution that was implemented.

### 4.2 Recommendation System type choice

The choice between the two types of recommendation systems, in this case, was pretty obvoius. In fact, the content-based system is based on the fact that there is a computable similarity between items. In our case, with therapies being assigned randomly and without a "type" hierarchy, namely how two therapies or therapy types are closer to each other, the choice fell inevitably on a **collaborative filtering system**.

### 4.3 Classes

To handle data the decision moved to read from single dictionary for every user to uses classes and istances of that classes. The decision was made to better link conditions and trials in a patient. Doing this, every trial has linked through one of its istances not only the condition id, but an instance of the class "Condition" with all its attributes and viceversa.
This approach uses a little bit more memory but avoid using "for" cicles to read dictionaries.
Another important point that promote the choice of using classes is the fact that not all therapies and conditions that are in the lists of selected patients are important for our algorithm:

- For conditions, it is useful to consider only conditions that happened before the target condition.
- For therapies, it is useful to consider only trials that were made on the target condition, not taking care of other trials, reducing a lot the dimension of this list.

### 4.4 Two different similarities

While trying to figure out the solution, and considering how two patients would have been considered "similar", two different approaches were found:

(1) **Past trials similarity**
This similarity is based on the assumption that if the target patient and another patient have tried the same therapy for the target condition, the success percentage of the trials could be used to compute a *centered cosine similarity* on each patient and the target patient. Using the centered cosine allow us to consider non-tried therapies as average and not as negative, because non having tried, in this case, doesn't mean that it is bad.
The only problem with this approach is that, being trials randomly assigned, a big subset of patients has no more than 1 therapy in common with the target.

(2) **Clinical history similarity**
This similarity, instead, is based on the clinical history of each patient and the target, meaning that every condition that each patient on which similarity is computed had before getting affected by the target condition is compared with the clinical history of the target patient. This because of the consideration that every condition, also already cured ones, could affect the efficiency of the immune system.
In this case, ratings are not important. What matters in the computation of the similarity is if a patient had or not the same condition of the target patient. Using only binary numbers instead of rating, less memory is used and a *Jaccard similarity* could be computed.

### 4.5 How to combine these similarities?

Reflecting on how to combine these similarities, it was figured out that possibly these two similarities could have not the same "weight" in the final computation of similarity. The decision was at the end to do a **linear combination** of these similarity, in the following way:

$$sim = \alpha + 2\beta$$

with $\alpha$ = past trials similarity and $\beta$ = clinical history similarity. This means that in this recommendation system the clinical history of a patient is more important that the previous fails or success of same trials, also because, as stated previously, past trials similarity would have been more important if the dataset would not have been random, because in the real world for each condition there are obviously not much therapy to try, and this would have increased a lot the number of similar trials and consequently the importance of this particular similarity measure.

### 4.6 The discarded solution for similarity

Another thought possible solution for similarity was to compare patients by how the same therapy has worked on the same condition, for all the therapy and all the conditions that a patient has had in the past.
This solution was then rejected because the total amount of memory used would have been incredibly bigger, because the comparison wuold have been between matrix with therapies in the columns and conditions as rows, instead of "simple" vectors.
Further, i was not sure that having cured the same condition with the same therapy means that probably the two will cure another condition with the same therapy.

### 4.7 Ratings prediction

To predict ratings and consequently to advice the best possible therapy, namely the 5 with highest rating, a simple mathemtical equation was done:

$$r_{th_i} = \frac{\sum_{j \in N} sim_{U_t,U_j} * r_{iU_j}}{\sum_{j \in N} sim_{U_t,U_j}}$$

that is a classical **weighted average** of ratings, in which $U_t$ is the target user, $U_j$ is the j-th user in the list of users and $r_{iU_j}$ is the success percentage of the therapy $i$ for the patient $j$.
About this last and maybe most important section of the algorithm, as is the core of a recommendation system, it is important to point out another decision that has been made. The decision is to not consider therapies that are already been tried on the patient. This because, in a real world, it is supposed that both the patient and the doctor knows the therapies that are already been tried and the doctor and only him/her itself could be able to know if a therapy that failed before could now be successful, maybe because conditions are changed. But this requires a knowledge that is not in the competence of this algorithm.

## 5 IMPLEMENTATION

This section is to explain how the ideas described in the previous section were turned into code, explaining the main function's work.

### 5.1 Data filtering

(1) Firstly, trough the function called `filterData()` the algorithm extract the list of conditions and the list of therapies ahd save them into two variables.
(2) Then, with the same function the algorithm read all the dataset and extract only patients that has cured the condition, inizializing a new object of the class "Patient". It then inizialize a "Therapy" class object for all the therapies used for that condition and a "Condition" class for all the condition that happened before the target one, saving them into a list. Finally add this two lists as istance variables of that "Patient" object. Each patient is added to a list that is returned in the function
The control of considering or not a condition is done trough a function called `isBefore()` that compare the starting date of the condition with the date in which the target condition was cured. If the first one is before the second one, the algorithm instances an object for that condition.
(3) In the meanwhile, when the algorithm find the id of the target patent (given as input of the program), it instances an object of of that patient but in this case with all his/her conditions, and return this patient as a single variable.

### 5.2 Similarities

(1) Having the list of trials and conditions and the list of patients, the algorithm creates two matrices, called `T-matrix` and `C-matrix`, both with patients as rows and respectively therapies and conditions as columns.
(2) Subsequently, the function `computeTherapySimilarity()` is called. First of all, this funtion creates a `numpy` vector of zeros trough `numpy.zeros()` that is a vector of therapies, with the success rate of the therapy (for the target condition) if the target patient has already tried and 0 elsewhere.
(3) Then the function , reading the list of patients, row by row instert the success rates under the right column (i.e. the therapy number) and normalize both the therapy matrix and the previous vector, through the function `normalize()`, that simply compute the mean of each row (or of the vector) and subtract from each value of every row the mean of the row.
(4) Finally, the function compute the distance between the target vector and each row (meaning each patient), using the package `scipy.spatial` and its function `distance.cosine`. As the bigger the distance the less the similarity, to compute the similarity we simply do $1 - distance$. All these similarities are stored in a vector that is returned as output of the function together with the target vector that is going to be used afterwards.
(5) After, the function `computeConditionsSimilarity()` do a similar job for the `C-matrix`. In this case, however, substitute the zero with 1 in each row if that patient had in the past such condition (column). In this case, there's no normalization beacuse the similarity is computed using *Jaccard similarity*, obtained trough the function `jaccard.score()` from the package `sklearn.metrics`.
(6) At the end, the algorthm calls the function `combineSimilarities()` that takes as input the two similarity vectors, anche return a new vector of the same lenght in which each value is the linear combination of the two similarity, as stated previously in the "Solution" section.

### 5.3 Rating prediction

Having obtained the similarity array, the next step of the algorithm is to call the function `predictRatings()` that, as the name itself tells, is going to return an array of predicted ratings for all the possible therapies.
This algorithm simply, for each column of the `T-matrix`, compute the rating prediction computing the weighted sum of ratings and the sum of similarities, and then divide the fist by the second and put the predicted rate in the same column number in the target vector, and put 0 instead if the column has already a success rate (meaning has already been tried).

### 5.4 Output

(1) Now that the algorithm has a complete vector of ratings, it send it to a function called `fiveMostEfficient()` that creates a copy of the vector, sort that copy in descending order and return an array with the index position in the original vector of the first five of the copy sorted vector.
(2) Finally, a simple function called `produceOutput()` simply uses the list of therapies obtained in the first steps of the algorithm and obtain the names of the therapy that has as name "Th"+ each of the indexes in the vector returned

by the previous function, and give them as final output of the algorithm.

## 6 DATASET

As stated previously, the choice was to use only mandatory variables.

### 6.1 Therapies and Conditions

First, for the list of therapies and conditions, that have both only a name and a "id", The names were taken trough a list of real names obtained at this links:

- conditions
- therapies

The choice was to select not many different therapies and conditions, because again, being the assignment random, having lot of both would have mean that very few patients had similar treatments for the same condition, and that would have reduce the effect of using similarities. In particular, 50 different therapies and 50 different conditions were randomly assigned. For ids, are simply numers from 0 to 50 for both.

### 6.2 Patients

Talking about patients, many choice had to be made. Summarizing:

- The **number of patients** selected was 25000, after many trials, this number was selected because the number of patients that remain after the filtering is always around 4/5000.
- The name of the patients were choosen randomly from a set of more than 300.000 names obtained from https://github.com/philipperemy/name-dataset
- For each patients, the number of conditions was randomized with a minimum of 0 and a maximum of 20.
- For each condition created, the starting date (the variable `"diagnosed"` in the dataset) is randomly created from 01/01/1972 to 31/12/2021, so 50 years of data collected. The choice is a compromise between using "realistic" dates and not having most of the conditions that lasts year or even dacades, while few cured in a reasonable time.
- Again, for each condition, to select if it has to be cured or not, a random number from 0 to 4 was estracted and if 0 "Null" is written in the dataset in the variable `"cured"` , else a random date between the start and, again, the 31/12/2021.
- For each condition, a random number from 0 to 10 is estracted, that is the number of trials for that condition (not the same for all).
- For all trials except for the last one, the success number is again randomized number between 0 and 75, that is the percentage range in which the condition is not cured. Then, if the condition is cured, the last one have a random number between 75 and 100.
- For each of this trials, a start and end date was randomized. For each condition, the order of trails is an ascending order of starting dates. This means every starting date was randomized starting from da day of start of the previous one, while, instead, the end date is simply randomized between the starting date of the same trial and the date in the variable `"cured"` of the condition if the condition was cured, and again the 31/12/2021 elsewhere.

No therapies are still ongoing on the dataset because a therapy that has no finish could not have a success rate, making it meaningless for our algorithm. Also in this case, if the condition is cured, the last trial's end date is the date in which it was cured.

## 7 EXPERIMENTAL EVALUATION

In this case, doing an evaluation on the result is difficult, both because there is no domain expertise (i.e. I'm not a doctor) and because therapies are randomly assigned for random conditions. So told, the first experiment was taken on both the dataset described before and the assigned dataset, that has the same mandatory variables, but 100000 patients, 300 possible conditions and 50 therapies, but with less conditions and therapies for each patient.

| | Exec1 | Exec2 | Exec3 | Exec4 | Exec5 | Exec6 | Exec7 | Exec8 | Exec9 | Exec10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Time | 6,85 | 6,34 | 7,39 | 6,89 | 7,23 | 7,03 | 6,45 | 6,44 | 6,89 | 6,57 |
| | | | | MEAN | | | | | | |
| | | | | 6,808 | | | | | | |

**Figure 1: Time expressed in seconds of ten executions on different patients and conditions on assigned dataset**

| | Exec1 | Exec2 | Exec3 | Exec4 | Exec5 | Exec6 | Exec7 | Exec8 | Exec9 | Exec10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Time | 12,4 | 13,67 | 11,72 | 13,56 | 11,33 | 10,92 | 12,13 | 11,43 | 13,19 | 11,65 |
| | | | | MEAN | | | | | | |
| | | | | 12,2 | | | | | | |

**Figure 2: Time expressed in seconds of ten executions on different patients and conditions on my dataset**

This experiment that is shown in `Figure 1` and `Figure 2` is about scalability. The average time of 10 different execution was 6.8 seconds on the assigned dataset and 12,2 seconds on mine, calculated on a Intel Core i5-8250U personal computer with 1.6 GHz, 4 core, 8 threads, 8 GB ddr4 RAM and 256GB SSD.

Another experimental evaluation that was taken was the comparison with a base method that simply count how many times each condition cured a patient, without considering similarities and ratings, and advices the 5 therapies that has cured most patient from the target condition. This experiment is important for two factors:

```
PS C:\Users\piomb\Desktop\Progetto DM> python 'src/test.py' 'data/datasetB.json' '6' 'pc32'
[ 0. 35. 22. 28. 24. 29. 26. 29. 27. 25. 16. 20. 28. 20. 20. 26. 24. 24.
 23. 31. 31. 28. 18. 20. 24. 22. 31. 28. 27. 27. 28. 28. 35. 31. 27. 20.
 27. 33. 32. 30. 32. 34. 23. 26. 29. 29. 31. 28. 32. 29. 31. 28.]
Top 5 advised therapies (descending order):
1) abortive therapy
2) molecular chaperone therapy
3) protein therapy
4) phytotherapy
5) polytherapy
PS C:\Users\piomb\Desktop\Progetto DM> python 'src/main.py' 'data/datasetB.json' '6' 'pc32'
Top 5 advised therapies (descending order):
1) monotherapy
2) sound therapy
3) aurotherapy
4) antibody therapy
5) dietary therapy
PS C:\Users\piomb\Desktop\Progetto DM>
```

**Figure 3: Comparison of executions of the main algorithm and the test on the same dataset and with the same target patient and condition**

- Firstly, the average time of execution of this simple algorithm is around 5.5 for both datasets. This means it is really similar to the time on assignet dataset and about a half of executions on mine. This explains that at least 5 seconds are only to read data and produce the output, meaning that the pure recommendation system works in about a second for assinged dataset and about 6.5 seconds on mine. Further, for this problem, advicing the best therapy is really more important that having it in 6 seconds less.
- Secondly, as you can see from `Figure 3`, the adviced therapy in the `test.py` execution (i.e. this experiment) and `main.py` execution (the real recommendation system) **are different**. This demonstrate the efficacy of similarities and ratings prediction.

Finally, the `test.py` executions also shows the array of count for each therapy, as stated before. This not only means that the dataset is pretty complete (my dataset, that is not shown, has generally bigger numbers) because every therapies has many successes (and so trials in general) for the target condition to compute similarity, but also that inevitably, while handling random datasets, there aren't therapies that worked much more than others and viceversa. This reflects in the predicted ratings array, shown in `Figure 4`, in which the difference between ratings are very thin.

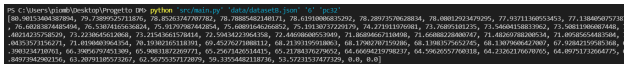


**Figure 4: predicted ratings sorted in descending order**

## 8 CONCLUSIONS

In conclusion, this work should have been tested on a real dataset and with a domain expert to really evaluate its effectiveness.
Furthermore, having a domain expert to work with would have permitted to use many other aspects in the evaluation such as contraindications of each therapy related to every active condition of the target patient (ibuprofen based therapies has to be avoided if a patient has also heart deseases) or using a "weight" also for conditions (having had both headache or having had both diabetes doesn not count the same in the similarity).
Nevertheless, the experimental evaluation shows that using similarity and rating prediction changes the output of the algorithm, and as time is not much bigger the use of a recommendation system that could have more precision is possibly worth it, as we are dealing with human healt.

## REFERENCES


[1] Vatsal, 12/7/2021. *Recommendation Systems Explained.* towardsdatascience.com
[2] Baptiste Rocca, 03/06/2019. *Introduction to recommender systems.* towardsdatascience.com
[3] Unknown Author, 2010. *Data Mining Portfolio, Similarity.* humanoriented.com
[4] Jure Leskovec, Anand Rajaraman, Jeffrey D. Ullman, 2019. *Mining on Massive Datasets*