Recommendation system for mediacal domain

Andrea Frigo
mat: 223813
2nd year
Computer Science
andrea.frigo@studenti.unitn.it

ABSTRACT

This paper describes an implementation of a recommendation system that is able to suggest a therapy to a patient, given his actual conditions, his history and a dataset containing the information about other users.

1 INTRODUCTION

Nowadays the role of doctors is important for everyone, most of the people go to the doctor to understand how to cure their conditions. The COVID-19 pandemic has increased consistently the request for medical assistance, creating some diseases to the doctors that need to work harder. The system described in this paper will help the doctors on suggesting the therapies to the patients, and could also allow the creation of a website for automated suggestions based on the data provided. The data needed for this system are confidential data, but they can be obtained from the already available data in the medical domain, in particular it is needed:

- set of patients: the set of all the patients available, with the information from this list about each patient;
- patient's condition: all the conditions that the patient has/had, with temporal information on the start and possible end of the condition;
- trials for each condition: all the therapies tried (with temporal information) and success rate for each condition.

Of course also the list of all possible conditions and therapies has to be provided to the system, but all these data are not difficult to find with the right access to the public medical health.

2 RELATED WORK

The solution to the challenge provided in this paper is a Recommendation System, that is a system capable of predicting some results, given a lot of data. It is based on the similarity between users and/or items. The two main types of recommendation systems are the following:

- Content-based: tries to recommend items similar to old items that the specific user rated highly;
- Collaborative Filtering: finds users similar to the specific user and estimates the rating of the user based on the ratings of the similar users.

From the previous definitions two terms have do be defined in the specific domain, that are user and item. Recommendation systems are often used for online shops or to suggest articles to read, in these domains the definition of user and item is simple. For the challenge of this report instead users are simply the patients, while items are combination of condition and trials. This choice has been made in order to keep the relation between one condition and all the therapies applied, and also allows to consider the order in which the different therapies have been applied. The success rate of each trial is also important, and it is included in the trial's information.

The solution described in this paper has been developed to solve the problem with the given data, the solution design is created considering the dataset provided. Analyzing the dataset one important consideration can be made: in most of the case the same condition does not occur multiple times to the same patient. This consideration is important because it helps choosing the approach, the content-based approach is not useful in this case, while the collaborative filtering technique could be a good solution. The solution described in this paper is a specific solution, based on the collaborative filtering, that will be described better later in this document.

As said before, recommendation systems are based on similarity, there are different ways to compute similarity between users or items, in this paper two different measures are described:

- Cosine similarity (and Pearson correlation);
- Longest common sub-sequence (LCS).

The first can be used to compare users, it returns a value where higher is better. In this challenge there are many data that are missing, this is due to the fact that different users often have different conditions and are treated with different therapies, so an evolution of the cosine similarity is used. The Pearson correlation is better, because it has the same advantages as the cosine similarity, but also treats the missing values as average, while the other method treats them as negative rating. The Pearson correlation is simply obtained normalizing the data for each user and then calculating the cosine similarity, the values obtained are in the range [-1; 1].

The LCS method instead is used often in the bioinformatic domain to compare strings, but it can be adjusted in order to compare lists of elements. In particular the LCS returns the longest common sub-sequence, where a sub-sequence is a list of common elements between the two lists, where the elements maintain the order. Some examples can be the following (each letter is a different element of the list):

- LCS(ABCD, ABC) = ABC
- LCS(ABCDE, BADEF) = ADE or BDE
- LCS(ABCD, FGHA) = A

In the solution presented in this report the LCS method is used to compare different list of trials, the procedure is described in detail later in the document.

A recommendation system developed using the techniques stated before can suggest an ordered list of therapies that could be applied in order to try to solve a specific condition for a specific patient. The idea of the system is, given a specific patient P and one of his condition C:

- Order all the patients by similarity to P
- Search for patients that has/had the condition C
- For each patient Psim found before, check if the list of trials made by P and Psim is compatible, if it is suggest the therapy based on the trials made by Psim
- Suggest the best 5 therapies

All these operations are better explained in next sections.

3 PROBLEM STATEMENT

The challenge is the creation of a system to help doctors suggest the next most prominent therapy for a patient to try a specific condition he or she may have. In particular, given the following:

- A set P of patients, their conditions, and the ordered list of trials each patient has done done for each of his/her conditions
- A specific patient p , his/her conditions, the ordered list of trials he/she has done for each of these conditions
- A condition c

the system has to return a list containing the 5 best therapies that can be applied, ordered by chance of success.

The input dataset contains the following:

- set of conditions: each condition is an entity that represents a real possible condition (i.e. medical problem to be cured), it has attributes as id;
- set of possible therapies: like for the conditions, each therapy has an id;
- set of patients: each patient is an entity with an id and name (and other minor attributes). It contains also a list of conditions that he has/had, with start date and possible end date, with a Boolean that tells if the condition has been cured or not. It contains also a set of trials;
- trials: each trial has an unique id and it represents the trial done trying to solve a specific condition using a specific therapy, it has information like start and end date and it also contains the success rate value for that trial (this represents how successful that trial has been, the value can also be 100, meaning that the condition has been cured by that trial).

Two important considerations have to be made:

- the success o a trial may depend on what has been tried in the past: this means that the order of the trials done is important and has to be taken into consideration (it is assumed that past trials can affect positively the success rate of the last, not negatively, this means that, if a therapy solves the condition, it solves the same condition also with other therapies applied before it);
- the success of a trial depends on each patient (what other condition he/her has): this means that all the conditions that a patient has at some time have to be taken into consideration (and so also the therapies applied).

This two important considerations brought to some implementation choices that will be discussed later.

4 SOLUTION

As introduced in the sections before, the system designed is basically a recommendation system. The different parts are described in this section, while the implementation is discussed in the next one.

4.1 Dataset manipulation

The first operation to do is the reading of the dataset and its transformation into a useful structure. This seems to be a trivial operation, but it is not. As stated before, the dataset contains for each patient a list of conditions and a list of trials, this information has been used to create a structure where the different trials are divided by the conditions and where the order is kept. In order to make things clear this is a possible example, given:

- A patient P
- Two conditions C1 (start: 20220101) and C2 (start: 20220102)
- Some therapies Th1, Th2, Th3, Th4

The condition C1 has the following trials (therapy ID, start, success rate):

- T1: (Th1, 20220101, 40)
- T2: (Th2, 20220103, 80)

While the condition C2 has the following trials (therapy ID, start, success rate):

- T3: (Th3, 20220102, 50)
- T4: (Th4, 20220105, 100)

To satisfy the considerations made in the section 3, the data structure has to consider the order of each trial and also the conditions that overlap. Consulting the dataset as it is, it is possible to think that, to solve C2, the following trials have to be done in order: T3-T4, with final success rate of 100%, but it is not true, because C1 and C2 overlap. To solve C2 the following trials have to be done, and they have to keep this order: T3-T2-T4, with final success rate of 100%.

In order to keep these information and avoid to check every time for overlapping conditions a special structure has to be used, a structure that allows to store for each condition of each patient an ordered list of all the therapies/trials done with the corresponding success rate. To keep also the information of intermediate steps of curing, with their success rate, also all the possible list of trials have to be kept. Considering the example above the structure should contain for patient p and condition C2 the following ordered lists:

- T3, success: 50;
- T3-T2, success ?;
- T3-T2-T4, success 100.

It is important to note that it can not be said that T4 solves C2, because it is not true, only T3-T2-T4 solves C2, it is not known what could happen trying only T4. Also for the second list (T3-T2), the success value is not known, because the trial T2 has a success value that refers to another condition.

4.2 Finding similar users

As said before, the system should be able to suggest a therapy for a specific patient P, in order to do so the system has to been able to find other patients similar to P. To do this, a specific technique should be used, the output of this phase is the list of patients (all patients but P), ordered decreasingly by similarity with P.

To compute the similarity between two patients P1 and P2 it has been used a technique based on cosine similarity. Two patients are considered similar if they have some conditions in common and they treated them in a similar way (list of therapies done), obtaining similar results (similar success rate). The cosine similarity is perfect in such a case, but some considerations have to be made:

- If P1 and P2 do not share any common condition then it is impossible to know if they are similar or not, so a neutral value should be returned;
- If they share some conditions they could have done different therapies, so there is the need to normalize the values for each common condition for P1 and P2, in order to avoid to treat missing trials as negative, this brings to Pearson Correlation;
- As stated before, the system does not use single trials, but sequence of trials, so the similarity between trials should be computed in a special way (described later), the

similarity value between the lists of trial should be taken into account.

All these considerations bring to the following algorithm:
For each common condition Ci:

- For each trial list of P1, having success rate SR1:
- Find the numerator doing:
  - Compare with all the trial lists of P2 and return the results of the best comparison (similarity S and success rate SR for the specific trial list)
  - Sum over the following: SR1 * SR * S
- The denominator is found following the standard Cosine similarity formula (using the non-common trial lists)

The final result is obtained simply dividing the numerator by the denominator (both obtained analysing all the common therapies). The result is a value that can be positive or negative, 0 in case of unknown similarity. The patients are then sorted based on this values, from high similarity to low (negative) similarity. Of course for this algorithm to work it is needed another algorithm to compare the trial lists for the same condition, that will be discussed in the following subsection.

4.3 Comparing trial lists

The comparison between the therapies/trials done to solve the same condition is one of the key parts of the system. The choice to use list of trials to keep the strict ordering between them is a good choice to respect the project requests, but it made things more difficult.

The algorithm described will be used only to compare one single trial list T1 with an entire list of trial lists TL and it is the following:
For each trial list T2 of TL:

- Check if the last therapy of T1 and the last therapy of T2 are the same, if they are not just skip that T2 and try the next (this is done because the system can suggest a therapy basing only on those trial lists that contain that therapy, to preserve the temporal condition described above)
- Find the LCS between the two list and store the length of the LCS, the success rate of T2 and the length of T2
- Continue like this and keep the similarity, length and success rate of the T2 that has the greatest similarity value (and a significant success rate, remember that some success rate are not known, in this case the element is skipped)

This similarity measure returns the similarity (normalized over the length of the longer list between T1 and the stored length, it is a number between 0 and 1) and the success rate stored.

The algorithm require the usage of the LCS algorithm, which uses dynamic programming and can return the length of the LCS and/or also the list corresponding to the longest common sub-sequence.

In order to make things clear this part will be described using an example, suppose there are two different lists (TL1, TL2) of trial lists for the same condition:

- TL1: [Th4, 10], [Th4-Th1, 40], [Th4-Th1-Th2, 100]
- TL2: [Th5, 12], [Th5-Th4, 20], [Th5-Th4-Th1, 50]

Assuming I want to compare the last trial list of TL2 (lets call it T) with TL1, in order to try to suggest a therapy to solve the condition for the patient having TL2, then the algorithm works like this:

- At the beginning the length of LCS is 0, same for success rate and length of second list
- Start with [Th4, 10], the last therapy is Th4, and it is different then the last of T (that is Th1), so skip this element and check the next one
- The next one is [Th4-Th1, 40], this one has the same last element Th1, so do the next steps
- The LCS between [Th5-Th4-Th1] and [Th4-Th1] is [Th4-Th1], so its length is 2. It is bigger than the one stored (0), so store the length of LCS (2), the success rate (40) and the length of the list (2)
- The last one is [Th4-Th1-Th2] that does not have the same last element
- All the different lists have been analyzed, so the algorithm returns LCS length/length of list (2/2 = 1) and the success rate (40).

From the result obtained it can be said that T and TL1 are very similar (the similarity value is 1 that is the max possible), and this value with the success rate can be used for comparing the users (as done in the previous subsection) or also to suggest a therapy to apply (as will be done in the following subsection).

4.4 Therapy suggestion

Having the list of Patients similar to the input patient P and a way to compute the similarity between trials, it is possible to suggest a therapy to try to solve the condition. The algorithm to suggest a therapy to solve condition C for patient P, suggested using patient P2 is the following:

- find the last trial list LT of P2 for condition C that has a success rate known
- find the LCS CT between the therapies done by P and LT
- check if LT is like CT + one single therapy T, if this is true return t and the success rate of LT, if this is false then do the same for the previous trial of P2
- if P2 has done one single trial to solve C, then simply return that trial and its success rate (this means that the condition can be solved with just one therapy, so that therapy can be suggested regardless the past trials of P)
- If none of the condition above are true then return none

With this algorithm it is possible to compute the suggested therapy to solve the given condition, comparing to one patient. Of course it can also be that the output of this algorithm is none.

To perform the correct choice of therapies, the above algorithm has to be run for each patient, and considering the similarity between patients and the results obtained by the above algorithm, the 5 "best" therapies are suggested. The therapies are chosen based on the patients similarity (30% of the importance) and the success rate of the trial list (70% of the importance). The values for importance of those attributes have been derived testing the performance of the system, as stated in the last section of this document.

5 IMPLEMENTATION

In this section some details on the implementation of the system are described. I decided not to use any data mining library, because I created a non-standard data structure for this system, in order to satisfy all the requirements and increase the performances. All the methods described in this section are present in the code of the system, and are described also there.

5.1 Data structure

To read the JSON dataset provided I used the JSON.load() function. The important part comes for the dataset manipulation, because, as written before, a particular data structure has been created in order to keep also temporal information. Recommendation system are based on the Utility matrix, that compares users with items. For this system the items are represented by a combination of conditions and therapies/trials lists, with respective success rate. To store all these information a standard matrix would have been difficult to use and would have been very sparse (because of all the possible lists of therapies/trials done to try to solve a single condition).

The structure D created has the following properties:

- D is a list of dictionaries
- D[i] represents the patient with ID i
- D[i] is a single dictionary, where the keys are the conditions of that patient and the values represent the different trials list done trying to solve the condition
- Given a patient p and a condition c, D[p][c] contains a list of lists
- D[p][c][i] is a list containing the i-th list of trials and its success rate

For example, if for a patient p and condition c there are the following trials (in temporal order):

- Th1, 30%
- Th2, 50%
- Th3, 100%

Then D[p][c] = [[[Th1], 30], [[Th1, Th2], 50], [[Th1, Th2, Th3],100]]

A structure like this one allows to analyze the different trials keeping their order. In section 3 another two considerations were stated: the success of a trial may depend on what was tried in the past (this is considered by the temporal information in the data structure, given by the lists of ordered trials), but also that the success of a trial may also depend on what other conditions the patient has in that moment. In order to consider also this second statement, the list of trials is created considering all the trials that a patient has done from the beginning of the condition to its possible end. The example shown in section 3 is reported also here to make things clear, given:

- A patient P
- Two conditions C1 (start: 20220101) and C2 (start: 20220102)
- Some therapies Th1, Th2, Th3, Th4

The condition C1 has the following trials (therapy ID, start, success rate):

- T1: (Th1, 20220101, 40)
- T2: (Th2, 20220103, 80)

While the condition C2 has the following trials (therapy ID, start, success rate):

- T3: (Th3, 20220102, 50)
- T4: (Th4, 20220105, 100)

Condition C2 starts in 20220102, so all the trials done between that date and its possible end (in case there is no end it is from the beginning to now) are T3, T2 and T4. So the structure will be: D[p][C2] = [[[Th3], 50], [[Th3, Th2], None], [[]Th3, Th2, Th4], 100]]. This part in the system is done by the function ***utils.createDataset()***.

5.2 Finding similar patients

The general algorithm to find similar users has been already described in section 4, in order to do it some different algorithm are required.

5.2.1 Patient normalization.

Normalizing the patients (just for this phase) is important in order to consider missing values as mean values. The normalization should not modify the original data structure, but simply return a copy of the patient, but normalized. To copy the single patient, the *copy.deepcopy()* function has been used, it requires some time, but creates a real copy of the given structure, so modifying the copy will not affect the original structure. The normalization is then done subtracting the success rate mean for each trail list (considering of course only trial lists with significant success rate). The normalization is done by the function ***utils.normalizePatient()***.

5.2.2 LCS.

The LCS is used in a couple of functions, it is needed also to compare the patients. The algorithm for doing it uses dynamic programming and has been modified in order to use lists instead of strings. It can return the length of the LCS and/or the LCS itself, for the patient similarity only the length of the LCS is used. The LCS is done by the function ***utils.lcs()***, if the LCS list is needed, also the function *utils.lcs_list()* is used inside it.

5.2.3 Trial comparison.

It is used to compare one single trial list with all the trial lists of another patient for a specific condition. This is needed in order to understand how similar a patient is to another patient that has some conditions in common. The algorithm is described in section 4.3, it uses the *utils.lcs()* function to compute the length of the LCS between the single trial list and the different trial lists of the other patient. This part is done by the function ***utils.compareTrials()***.

5.2.4 Calculate patients similarity.

To compute the patient similarity all the functions described before are used, the algorithm is described in section 4.2. The *math.pow()* and *math.sqrt()* functions are used.
The function that calculates the similarity between two patients is ***utils.pearsonCorrelation()***.

5.2.5 Sort patients by similarity.

This is the final function regarding the patient similarity, given a single patientID P it returns a list of patientIDs ordered by similarity with P (of course P is not included in this list), it calculates the similarity using the *utils.pearsonCorrelation()* described before. The order is from high to low, so the first elements represent the most similar patients. The function is ***utils.similarPatients()***.

5.3 Suggest therapies

This is the main part of the recommendation system, in this section there are the functions that allow to suggest therapies for specific patient and condition.

5.3.1 Suggest one therapy for one patient.

The basic function allows, given a list of trials done (by the patient for the condition given in input) and another patient, to suggest a therapy. The algorithm has already been described in section 4.4. For this function both the LCS length and the LCS list is needed. The function to suggest the single best therapy (comparing with one single patient) is ***utils.suggestTherapy()***.

5.3.2 Suggest the 5 best therapies.

This is the part where the suggestion of the best five therapies is done. In order to do so, the *utils.similarPatients()* and the *utils.suggestTherapy()* functions are used.

To decide which are the best therapies, a particular scoring function *utils.scoreSuggestion()* is used, this function computes the final score for each therapy suggested, the score is the measure that is used to decide which are the best therapies and it is based on the similarity between patients and the score of the therapy suggested (remember that, to compute the patient similarity, also the trial list similarity is considered), this function will be described also in section 7.

Having the list of all the therapies suggested and their score, the best five therapies are simply the first five of the list (removing duplicates). The suggestion of the best therapies is done by the function ***utils.therapyList()***.

6 DATASET

The dataset created contains only the mandatory information, but has some additional properties that makes it better according to my interpretation. Given the fact that the success of a trial may depend on past trials, I created the trials in such a way that, given two trials T1, T2 that try to solve the same condition, if T1 has been done before T2, then the success rate of T2 is greater or equal than the success rate of T1. This is done considering the assumption that a therapy can increase the chance of success, or do nothing, but not decrease it. The code for generating the dataset can be found in the src/data-mining-dataset folder. The script is able to read the different input files (html and csv) using the functions *readNames()*, *readConditions()*, *readTherapies()* and the csv, bs4 and codecs libraries. The script uses all the condition and therapy names and generates the required attributes. It then creates 1000 patients, each one having from 1 to 5 conditions, each one with diagnosed date and cured date (10% chances to be null). For each conditions some trials are generated (from 0 to 50), each one with start and end date (starting from when the condition was diagnosed and ending to its cured date or now if the condition was not cured). The success rate of each trial increases randomly for each new trial, up to 100% in case the condition was cured. If the last trial is successful then the cured date of the condition is the same as the end date of the trial.

The script for generating the dataset allows also to generate 3 test cases, this is simply done by adding a non cured condition to 3 patients, this was not an optimal solution since these test cases do not include any condition with some unsuccessful trials applied, but the recommendation system was not complete when I generated the dataset, so this case did not come in my mind.

The script generates also a README.txt file that contains the id of the patients corresponding to the test cases.

7 EXPERIMENTAL EVALUATION

In this section some consideration about evaluation and performance of the system are stated.

7.1 Evaluation

Given the fact that the system does not implement a standard utility matrix, evaluation using techniques like RSME are not easy to perform and not useful. A recommendation system is a software that helps to predict some results, so one way to test it is simply to try to guess something already known.

The function ***test.test()*** is used to perform a defined number of tests, each test works in this way:

- select randomly a patient and one of its conditions
- store the last trial list T
- remove T from the dataset
- use the *utils.therapyList()* function to compute the best 5 therapies TH
- compare the last therapy of T with each therapy of TH, if they are the same the prediction is correct, else the prediction is not correct
- insert again T in the dataset and proceed with the next test

The algorithm takes some time to run, because computing the 5 best therapies is not fast. Maybe the best way to do this was to copy the data structure and working only with the copy, without modifying the actual data structure, but this would have taken too much time (this is why there is the removal and appending of T).

This testing measure is not perfect, it does not consider the success rate predicted (but this is not a big issue, the important thing is to guess the therapy, not its success rate, and the system suggests effective therapies, so the success rates will always be high, probably 100% when possible) and it does not consider the position of the correct therapy inside the therapy list (so predicting the correct therapy as first or as fifth does not change the result of the algorithm), in particular this second choice has been made to keep the evaluation simpler to analyze.

Using this evaluation measure it has been possible to derive the best score suggestion parameters, for the *utils.scoreSuggestion()* function presented in section 5.3.2, in particular the best performance is reached giving the following weight to the parameters:

- similarity between patients and trials lists: 30%
- success rate of the trial lists: 70%

The big problem about the evaluation performed are the results, performing the tests on random patients and conditions, it has been seen that the mean value of prediction correctness is 12/100, that is very low (this has been found performing 50 test with different random seed, in total 600 predictions have been tried).

This result is bad, the motivations could be some of the following:

- As written before, the recommendation system designed works well with the assumption that a therapy can not make a condition worse, the performance with a dataset where the success rate increase with the number of trials performed could be better.
- This is maybe the most important factor, the dataset has been generated randomly, this means that there are no real correlations between patients or therapies. The correlation between items or users is at the base of recommendation systems, it is expected that, without these correlations, the performances will drop consistently.
- There could be some errors in the algorithms used to create the system, I do not think this is the reason, but it could be possible.

Running the system with the dataset generated by me the results are even worse, this is due to the fact that I generated too many trials for each condition (up to 50 trials for condition), so taking into consideration also the history makes it difficult to

guess correctly the therapy. The results obtained, running 600 tests, is 1/100 correct.

7.2 Performance

The system execution for predicting the 5 best therapies for one single patient and condition can be divided in three phases:

- Read the dataset
- Create the data structure
- Predict and return the best therapies

7.2.1 Read the dataset.
There is not much to say about the first phase, reading the dataset takes time, but it is an operation that has to be done and can not be improved, it takes O(n), where n is the length of the dataset, this means that the computational time to do it is linearly dependent on the dataset size.

7.2.2 Create the data structure.
The data structure used by the system is not a standard structure, there are different reasons for this choice (stated before), one of these is the performance for creation and use of the structure. For creating the data structure only one read of the dataset structure loaded before is needed (not all the dataset, just the patients), creating the different lists of trial lists takes time (this is the most expensive operation), but allows to have a structure that allows to run the algorithms in a faster way. The time needed for this operations is not much greater than O(n), this is because the worse part is creating the lists of lists for trials, but for many conditions there are not many trials.

Also the space is important, the created structure does not have any empty field inside it (this means that no memory has been reserved for nothing), this helps to avoid to waste space like in case of sparse matrices.

7.2.3 Predict the therapies. It is hard to specify the right amount of time needed in terms of dataset length, but some considerations can be made. Let p be the number of patients, c the maximum number of conditions that a patient can have and t the maximum number of trials performed for a condition.

- The *utils.similarPatients()* function is called just one time, this functions calls the *utils.normalizePatient()* and the *utils.pearsonCorrelation()* for each patient, then it orders patients.
- The normalization is linearly dependent on the number of trials for each patient, so the normalization function takes O(c*t) for each patient, applying the function to each patient the overall result is O(p*c*t).
- The Pearson correlation function (forget about the normalization inside it, because it has been considered in the previous point) is linearly dependent on the number of common conditions and trials for each patient, and also it calls each time the *utils.compareTrials()* function.
- The function to compare trials just calls the LCS function on one trial, the LCS function has complexity O(l1*l2), where l1 and l2 are the length of the two lists, but the list of trials are not long, so this call can be ignored since it will not affect much the performance.
- The complexity for the Pearson correlation function is then O(c*t), since it is called for each patient the overall complexity is O(p*c*t).
- Ordering the list of patients takes O(p* log(p))

Given these analysis, the overall computation for this phase is something like O(p*c*t) + O(p*c*t) + O(p*log(p)).