Therapies Recommendation

Data Mining 2021 - Course Project

Matteo Zanella
University of Trento
2nd year - AIS
matteo.zanella-2@studenti.unitn.it

ABSTRACT

We are currently experiencing an unprecedented increase in the volume of data gathered. There is undeniably a recognition of the importance of this data, as well as the ongoing development of new tools for collecting and analyzing it in an exceptionally effective manner. While the public is concerned that transferring their data may lead to unethical exploitation, it is apparent that depending on data allows for more informed judgments in areas where they can benefit.

In particular, in the medical profession, we are moving toward more individualized medicine, in which automated technology assist doctors in making faster and more conscious judgments. This evolution would be impossible without a substantial amount of data to back up these decisions and a system capable of extracting the most meaningful patterns.

The goal of this project is to present an innovative approach to assisting a physician in prescribing the best therapy for a patient's disease, taking into account both the patient's medical history and the success of other patients' therapies.

KEYWORDS

Data Mining, medical application, therapy suggestion, collaborative filtering, LSH, SVD

1 INTRODUCTION

It is evident that advances in medicine have a direct impact on individuals, and numerous attempts are being made to improve the speed and fairness of the health-care system, especially in the field of information technology. With all the data acquired from service users on a regular basis, one wonders if health data may also be utilized to assist physicians in their work. This project, created for the Data Mining exam in 2021/2022, aims to develop an automated system based on traditional data mining techniques that recommends to physicians the best therapies to treat a sick patient. This kind of recommendation system necessitates the processing of a large quantity of data. As a result, the project's second purpose is to build a random but realistic dataset on which to test the system.

The goal of the dataset construction was to produce a dataset at random that had a list of medical conditions, a list of therapies, and a list of patients. All entities are identifiable by an ID and had a name attribute. Conditions also include their type as an attribute, which specifies the macro-categories to which they belong. Similarly, also the therapies express the same attribute to further divide them in categories. Regarding the patients, their additional attributes are the list of medical conditions and the list of trial. The patient conditions are defined as particular instances of a condition from the condition list, that impact the patient throughout a certain time period, which is expressed by the diagnosis date and the date of cure. The absence of a cure date signals that the diagnosed condition has not yet been cured. A realistic generation should also strive to keep a patient from being affected by two different instances of the same condition at the same time. The trials, on the other hand, are therapies for a specific patient condition that begin on one date and end on another date with a certain numerical success. The ending date of a 100% success trial is assumed to be equivalent to the date of cure of the corresponding patient condition. For the same reason, after a completely successful therapy, no further therapies are tried. Even if not explicitly stated in the problem issue, it has been assumed that the last trial for a patient condition that has yet to be cured could, similar to the condition's date of cure, to indicate ongoing therapy.

Other attributes were permitted in the dataset generation, but not established by the problem definition and therefore were not guaranteed to be present or to be meaningful for the problem. For this reason, it is clear that basing the system analysis on extra attributes might improve the findings, but it makes the results dependent on the attributes themselves, which may change based on the dataset under consideration. To avoid such a problem, this study focused on the information explicitly stated by the issue statement, in order to build a method that worked well in as many circumstances as possible, even with scarcity of attributes.

The primary purpose of the project task was to recommend the five best therapies for treating a patient suffering from an illness, taking into account both the patient's personal history, the demonstrated effectiveness of the therapies in other patients, and taking into account the order in which they were prescribed.

If in fact a simpler approach might consider diseases as entities in their own right and investigate the most appropriate therapies for them, modern treatments are expected to focus instead on a more personalized medicine. For this reason, it was decided to include in the decision-making process also the characteristics of the patient himself. As a result, the first difficulty is that the therapy is associated with a medical case, which is a pair of patient and disease, two orthogonal components that can be present independently and bring two different aspects on which to measure the similarity of the medical cases. To simplify the model's underlying hypothesis, the patient is assumed to have only one instance of a specific condition in his entire life. Because the database specification does not guarantee this, a synthesis strategy is required to compress all the trials sequences for different instances of a condition into a single sequence associated with the corresponding condition. For the same reason, the patient condition specific instance passed to the system as a query is treated as the general medical condition.

The second difficulty lies in having to consider the order in which the therapies have been suggested in the available medical cases, as well as their success. A therapy could be very effective but have serious side effects and therefore be used as a last resort. Still, that information likely depends on the specific case under examination and there isn't a universal right answer. Asking what is the right balance between success and order of therapies is most likely an ill-posed question. The aim of this project is to propose a solution that permits to choose the desire balance of the two different approaches.

A third challenging aspect is that the therapy recommendation system should be scalable in terms of memory and run-time. In fact, in a real-world application, it should be able to handle a large number of citizens and provide suggestions in real time. Although the system initialization time is important, it is secondary to the actual query response time.

With these considerations, it is clear that such a system cannot replace a physician who is capable of taking into account all the variables of the case. It can, however, be used to provide helpful hints and suggestions. It should also be noted that aspects of patient privacy were not taken into account in the project development. Given that extremely sensitive data is being manipulated, the actual application of this suggestion system should provide solutions to tackle also this aspect.

2 RELATED WORK

The therapy suggestion problem studied in this project is very similar to the product recommendation problem, which is a traditional application of data mining techniques. When considering the goal of producing suggestions that are specifically tailored to individual users, the parallels are obvious. Recommendation Systems, which are a family of algorithms that aim to predict the rating a user would give to an item, are used to solve the product suggestion problem. They are typically divided into Collaborative Filtering (CF), Content-based, and Latent Factors. These algorithms make use of a utility matrix, which contains the user ratings for the items. This matrix is typically sparse because it is unlikely that a user will interact with all of the items.

When applied to the problem at hand, the rating would be the success of a therapy, which is the item, applied to the patient and the condition, who play the role of the user. The requirement of three variables (therapy, patient, and condition) to obtain the rating, rather than just the item and the user, resembles the Context-aware Recommendation System (CARS) subcategory, which takes additional dimensions into account for the suggestion. In the product suggestion settings, this could be the time of day or the user's mood. Such Recommenders are usually implemented with Latent Factors models, which directly optimize a cost function with gradient descent on a utility tensor, which is a multidimensional array.

To address the issue of therapy order, there are Sequence-aware Recommender models in the literature that take into account all previous interactions as context information. Recurrent Neural Networks or Markov Chain Models are commonly used to implement such Recommenders.

To concentrate on more traditional data mining techniques, the proposed strategy is a novel solution that differs from the examples presented in the preceding paragraphs. The solution is based on a modification of traditional Collaborative Filtering Systems, which suggest to the user items that have received high ratings from other users similar to him. More specifically, the vector containing the user's predicted item ratings is the average of the item ratings of the most similar users weighted by user-user similarity. The similarity of the users is determined by the similarity of their ratings. To further improve the predictions of CF when there is scarcity of similar elements, it is usually combined with Global Baseline method, which gives an approximate baseline rating to start from.

Content-based methods have been discarded because the suggested items have features that are similar to other items rated highly by the user, but in this scenario, the availability of the attributes for the therapies is not guaranteed.

The modifications to the collaborative filtering systems include the use of a ratings tensor instead of a ratings matrix and a method to integrate both patient and condition similarity. The discount factor from the reinforcement learning field is used to deal with the order in the sequence of therapies. Despite its simplification, it accurately models the tradeoff between immediate and long-term benefits of therapies.

The Singular Values Decomposition is a dimensionality reduction technique used to factorize a matrix, according to its eigenvalues and eigenvectors, into three matrices U, Σ, V. The Σ diagonal matrix holds the eigenvalues sorted by value. The best lowest rank approximation can be easily obtained by excluding the lowest eigenvalues, carrying the least information, and the corresponding columns and rows in the other two matrices. This property can be used in the Collaborative Filtering setting to decompose the sparse ratings matrix into denser embeddings for users and items, representing their association with a certain number of factors.

To compute the top most similar neighbors required for the ratings average of the CF technique, you could naively compute all possible embedding combinations. Because the complexity of doing so is quadratic, the faster Locality-Sensitive Hashing technique is commonly used to generate a list of candidate pairs with a high probability of being similar. The MinHash variant is frequently used for set similarity. The SimHash is more appropriate in our case, where we want to measure cosine similarity. The naive search can then be used to find the most similar neighbors among the smaller number of candidates.

3 PROBLEM STATEMENT

A formal reference model has been defined in order to treat the problem in a more systematic manner:

- P = Set of Patients
- C = Set of Conditions
- T = Set of Therapies
- C_P = Set of Patient Conditions
- $S = \{0, \dots, 100\}$ Set of values the success of a trial can assume
- $R = \{-127, \dots, 127\}$ Set of values the rating (the normalized success) of a trial can assume.
- Utility function $u : C \times P \times T \longrightarrow R$
- Patient condition to condition function $f : C_P \longrightarrow C$
- System $S(p, c_p) = \{t_i | i \in \arg\max_t^5 [\hat{r}_{f(c_p)pt}]\}$

Given a query composed of a patient $p \in P$ and a patient condition $c_p \in C_P$, from which you can derive the medical condition $f(c_p) = c \in C$, the recommendation system S returns the 5 suggested therapies $\{t_1, \dots, t_5\} \in T$ with maximum predicted rating $\hat{r}_{cpt}$, according to the utility function u.

4 SOLUTION

The available dataset is first structured into the rating tensor R_{CPT}, a three-dimensional vector that serves the same purpose as the ratings matrix in Collaborative Filtering. Then, the tensor is averaged across the conditions and the patients to produce the patients-therapies ratings matrix R_{PT} and the conditions-therapies ratings matrix R_{CT}. SVD technique is used to compute, from these matrices, dense embeddings for the patients and the conditions. From the embeddings, SimHash LSH produces the candidate pairs for the most similar patients and conditions according to the cosine similarity. The topNN search computes among the candidate pairs the most similar ones, referred to in this context as the nearest neighbors and abbreviated to NN, as well as their similarity scores. This completes the initialization procedure. At query time, when a patient and a condition are given, the nearest patients, the nearest conditions and their similarities are readily obtained from the previously computed nearest neighbors. The corresponding ratings are averaged in the modified version of Collaborative Filtering, combined with the Global Baseline method, to compute the predicted ratings. The overall procedure is summarized in the Algorithm 1.

Algorithm 1: Recommender System

```
Data: $p, c, dataset$
Result: $\{t_1, \ldots, t_5\}$
$R_{CPT} \leftarrow$ RatingsTensor($dataset$);
▹ Compute most similar patients
$R_{PT} \leftarrow R_{CPT}$.averageConditions();
$patientsEmbed \leftarrow$ Svd($R_{PT}$);
$candidatePatientsPairs \leftarrow$ Lsh($patientsEmbed$);
$paNN, paSim \leftarrow$ TopNN($candidatePatientsPairs$);
▹ Compute most similar conditions
$R_{CT} \leftarrow R_{CPT}$.averagePatients();
$conditionsEmbed \leftarrow$ Svd($R_{CT}$);
$candidateCondPairs \leftarrow$ Lsh($conditionsEmbed$);
$condNN, condSim \leftarrow$ TopNN($candidateCondPairs$);
▹ Predict the therapies ratings
$R_T \leftarrow$ CF($R_{CPT}, p, c, condNN, condSim, paNN, paSim$);
return $argmax(R_T, 5)$;
```

4.1 Ratings tensor

At the start of the initialization procedure, the dataset is organized as a 3D tensor holding all trials data. The scores of a sequence of trials for a specific patient condition instance of a patient are multiplied by a discount factor γ. To penalize later therapies, the factors are raised to the trial order index, where the first therapy tried is the number zero. The discount factor can be adjusted to control the importance of the order with respect to the success of the therapies. A discount equal to 1 leaves the scores unchanged, promoting the most successful, whereas a value of 0 emphasizes only the first therapy tried, leaving the others with a discounted score of zero.

$$s'_{c_p pt} = s_{c_p pt}(\gamma)^{o_t} \qquad o_t = \{0, 1, 2, \ldots\}$$

To make the problem simpler, if the patient has presented the same condition more than once in different patient conditions instances, the available discounted scores are averaged before populating the ratings tensor. The order in which the conditions manifest is ignored.

$$s_{cpt} = \frac{1}{k} \sum_{c_p \in c} s'_{c_p pt}$$

The final ratings are calculated by subtracting the expected success on untested therapies, which must be determined a priori and is most likely close to zero. Typically, the subtracted value is the data's average rating, so that missing ratings, saved as zeros, appear as average values. Unfortunately, in practice, the therapies given to patients are chosen by an expert from among the most likely to succeed. As a result, subtracting their average would vastly overestimate the ratings for the therapies that were not included. The scores are also multiplied by 1.27 to increase the occupied interval.

$$r_{cpt} = (s_{cpt} - \mu_{\text{untried therapies}}) \cdot 1.27$$

The procedure for building the ratings tensor is summarized in the Algorithm 2.

Algorithm 2: RatingsTensor

```
Data: $dataset, \gamma$
Result: $R_{CPT}$
$numPa \leftarrow |dataset.patients|$;
$numCond \leftarrow |dataset.conditions|$;
$numTh \leftarrow |dataset.therapies|$;
$R_{CPT} \leftarrow$ new Tensor($numCond, numPa, numTh$);
foreach $patient \in dataset$ do
    foreach $condition \in patient$ do
        ▹ Average duplicate patient's conditions
        $discountScores \leftarrow$ new List();
        foreach $patientCond \in condition$ do
            $scores \leftarrow patientCond.trials.scores$;
            ▹ Discount the scores
            $discountScores$.append($scores \cdot \gamma^{[0,1,2,\ldots]}$);
        end
        $discountScores$.average();
        ▹ Normalize the scores
        $ratings \leftarrow (discountScores - \mu_{\text{untried th}}) \cdot 1.27$;
        foreach $therapy, rating \in ratings$ do
            $R_{CPT}[condition, patient, therapy] \leftarrow rating$;
        end
    end
end
return $R_{CPT}$;
```

4.2 Singular Values Decomposition

Because the SVD method can only be applied to matrices, it cannot be directly applied to the ratings tensor. To solve this problem, instead of modeling the patient, the condition, and the therapy all at once, they are split into two parts.

The patients-therapies ratings matrix R_{PT} is calculated by averaging the various conditions. It models patients' reactions to various therapies without taking into account the specific condition under which they were administered. The idea is that similar patients will respond similarly to the given therapies.

The conditions-therapies ratings matrix R_{CT} is computed by averaging together the different patients. It models the efficacy of the various therapies on the conditions without considering the specific patient affected by the condition. The idea is that similar conditions are cured well by similar therapies.

The reduction of the ratings tensor into two matrices loses some information but allows the SVD technique to be applied to the two matrices separately.

$$R_{PT} = U\Sigma V^T \qquad R_{CT} = U'\Sigma' V'^T$$

The total eigenvalues energy is computed from Σ and Σ' as a linear sum $\Sigma_i \sigma_i$ instead of a sum of the squares. The eigenvalues with the lowest values are discarded, keeping only a fraction of the total energy according to a parametric threshold. With the corresponding columns trimmed, the resulting U and U' represent the dense embeddings of patients and conditions, respectively.

4.3 SimHash LSH

Because SVD produces embeddings of real values and the similarity measure used is cosine similarity, computing LSH signatures with MinHash is not ideal because it is designed for sets.

Given the requirements, it is preferable to use SimHash instead to generate the signatures. A certain number of random hyperplanes passing though the origin are randomly generated. The number of hyperplanes determines the signatures' length. The signature of an embedding vector e is computed using its dot product with each of the hyperplanes f_i. If the sign of the value is positive, the i$^{\text{th}}$ signature feature s_i is one, otherwise is zero.

$$s_i = \begin{cases} 1, & \text{if } e \cdot f_i > 0 \\ 0, & \text{otherwise} \end{cases}$$

The idea is that the likelihood of embedding vectors being separated by a random hyperplane, resulting in a different signature feature, is proportional to their cosine distance.

The signatures are divided into B bands composed of R rows, and bucketized by LSH. Embeddings added to the same bucket are proposed as candidate nearest neighbors.

4.4 Top Nearest Neighbors

The candidate pairs produced by LSH are tested for their actual cosine similarity. For each embedding, it is produced a priority queue of all the candidate neighbors ordered by the corresponding cosine similarity. The parameters $K_{patients}$ and $K_{conditions}$ define how many neighbors to keep for each embedding among the most similar ones. The remaining are discarded. If an embedding does not have enough candidates, fewer neighbors are extracted for it.

4.5 Collaborative Filtering + Global Baseline

A patient and a patient condition instance are provided at query time. The medical condition of the patient can be easily deduced from the dataset. The system should be able to predict the ratings of the corresponding therapies, allowing the highest scoring to be returned. Standard Collaborative Filtering cannot be used in our problem, since it only expects two dimensions, as illustrated by Figure 1. To address the third dimension, the proposed solution is to find the $K_{patients}$ and $K_{conditions}$ nearest neighbors of the query patient and the query condition. The predicted ratings are the weighted average of the ratings for the corresponding therapies, with weights equal to the product of the patient's and the condition's cosine similarity. The number of weighted rows is $K_{patients} \times K_{conditions}$, as illustrated in Figure 2.

$$\hat{r}_{cpt} = \frac{\Sigma_{p' \in N(p)} \Sigma_{c' \in N(c)} s(p,p') \cdot s(c,c') \cdot r_{c'p't}}{\Sigma_{p' \in N(p)} \Sigma_{c' \in N(c)} s(p,p') \cdot s(c,c')}$$

The CF predictions can be improved with the Global Baseline method. The baseline ratings on the patients and the therapies b_{pt} are computed by adding to the mean rating the patient's ratings deviation from the mean and the therapies' ratings deviation from the mean.

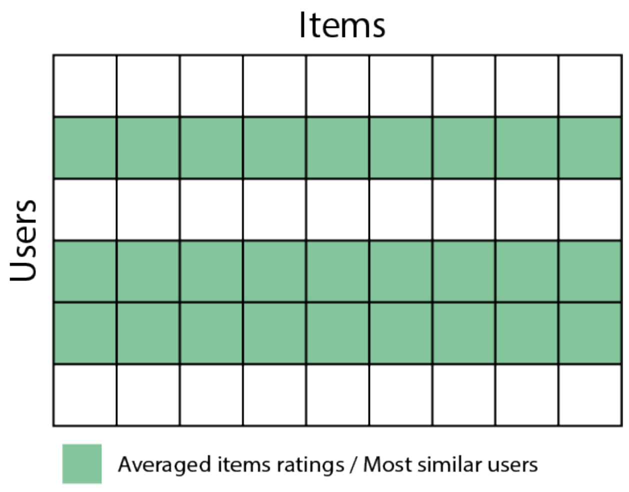

Figure 1: The standard user-2-user CF algorithm

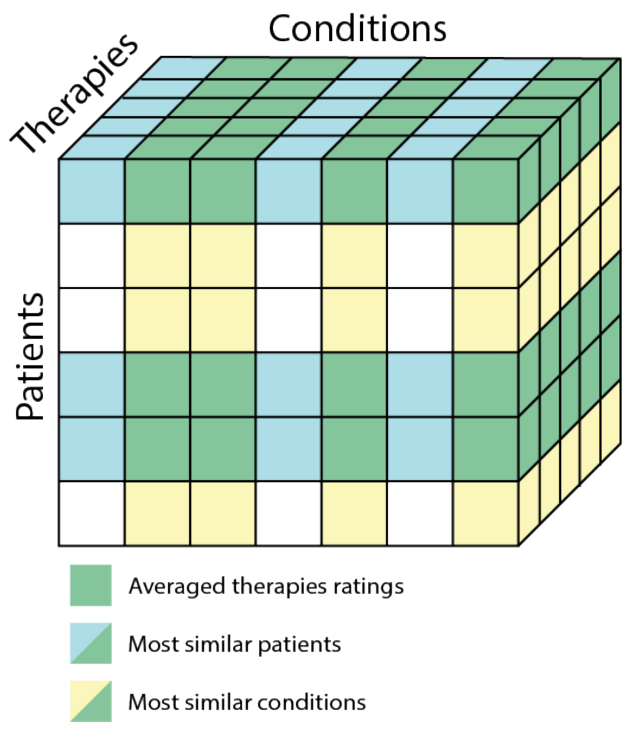

Figure 2: The modified multidimensional CF algorithm

The Global Baseline method can improve CF predictions. The baseline ratings b_{pt} are calculated by adding the patient's ratings deviation from the mean and the therapies' ratings deviation from the mean to the mean rating.

$$b_{pt} = \mu + \Delta_p + \Delta_t$$

By computing the baseline ratings ahead of time, you can easily incorporate them into the CF computation at query time.

$$\hat{r}_{cpt} = b_{pt} + \frac{\Sigma_{p' \in N(p)} \Sigma_{c' \in N(c)} s(p,p') \cdot s(c,c') \cdot (r_{c'p't} - b_{p't})}{\Sigma_{p' \in N(p)} \Sigma_{c' \in N(c)} s(p,p') \cdot s(c,c')}$$

If the query patient hasn't made any trial, and therefore its similarity with other patients is not meaningful, the System just averages equally for all patients and weights for the nearest conditions.

$$\hat{r}_{cpt} = b_{pt} + \frac{\Sigma_{p' \in P} \Sigma_{c' \in N(c)} s(c, c') \cdot (r_{c'p't} - b_{p't})}{|P| \cdot \Sigma_{c' \in N(c)} s(c, c')}$$

From $\hat{r}_{cpt}$, the 5 therapies with the highest predicted rating are selected as responses. Furthermore, the recommender keeps track of the therapies that have already been prescribed for the patient's condition and excludes them from the candidate therapies.

5 IMPLEMENTATION

The therapies recommender has been implemented with Python3. The project is structured into layers: the `recommender.py` script parses the input command, and instantiates the `Recommender`, a class in the `routines.py` file. Such class defines the basic algorithm stages, that can be called in sequence to generate the structures necessary for the predictions. The `algorithms.py` file defines the main algorithms as functions that are called by the `Recommender` to generate the structures.

5.1 Instantiate the system

`r = Recommender(data)` instantiate the recommender class, taking the dataset as parameter. The dataset is already converted from the JSON string to a nested dictionary of lists. In the `init` method some basic information, like the number of patients, therapies, and conditions are saved as class attributes. Furthermore, dictionaries later used to convert the dataset's string IDs into numerical indexes are saved. An additional list saves the conversion from therapies indexes to therapies IDs. Another dictionary saves instead the conversion from patient condition IDs to condition IDs. Patients without any trial are ignored, because they would just occupy memory without bringing any meaningful information. Then, in preparation for the next phases, the trials lists are sorted in ascending order for the condition ID, the patient condition ID, and the start date.

5.2 Building the ratings tensor

The fundamental library providing multidimensional array objects is NumPy, the most used package for scientific computing in Python. Calling `r.build_ratings_tensor()`, passing as parameters the success for untested therapies and the discount factor. The method creates a masked NumPy `ratings` tensor with fill_value=0 and datatype `int8`, and saves it as class attribute. The mask can be used during the evaluation to hide efficiently the test values, without having to recompute the tensor. Given the small interval of the ratings, 1 Byte is sufficient to save them. For each patient, his trials are then analyzed. Since the trials were already ordered, they can be grouped by condition ID and patient condition ID. The sequence of trials for a patient conditions is converted to a dictionary having for each therapy in the trials the corresponding discounted score. In the case of therapies duplicates, their ratings are averaged after being discounted. The dictionary values can then be added to an array long as the number of therapies, where the therapies untried for the patient condition are kept to zero. The therapies indexes are also added to a dictionary having the corresponding patient condition IDs as keys: this registry can be used to recover the therapies that have already been tried for a certain condition. These arrays are grouped by condition ID and averaged together as described in Section 4.1. The result is an array of therapies scores for combination of patient and condition. The scores are finally normalized by subtracting the success for untested therapies and multiplied by 1.27, occupying the greatest interval allowed by the memory Byte. Then, they are added to the `ratings` tensor in the indices corresponding to the patient ID and condition ID.

The boolean `data_mask` of shape (Condition, Patients) is then saved as class attribute, having True values for combination with at least one rated therapy.

5.3 Building the global baseline

`r.build_global_baseline()` computes the baseline estimates matrix of all patients and therapies combinations. The patients averages are computed by summing along the conditions and the therapies, and dividing by the number of non-zero values contributing to each sum. The therapies averages are computed similarly, summing along patients and conditions. With the same procedure, the global average is obtained by summing all non-zero values and dividing by their number. The deviations are obtained by subtracting the average value. The final global baseline matrix, saved in the class attributes, is the sum of the global average with all the patients and therapies deviations pairs.

5.4 Building the nearest neighbors

`r.build_nearest_neighbors()` computes the nearest neighbors matrices for the patients $|P| \times K_{patients}$ and the conditions $|C| \times K_{conditions}$. It also computes the corresponding matrices with the cosine similarity values.

First, it computes the matrices R_{PT} and R_{CT} by averaging, respectively, the conditions and the patients in the `ratings` tensor. To reduce the amount of computations during the evaluation, when some elements are hidden, the previously computed `data_mask` matrix is reduced to two `good_mask` vectors thanks to a boolean OR operation on the patients and the conditions. These vector masks indicate the presence of any data for each of the patients and the conditions, considered as a whole. A private method `r._nn()` can be used to generalize separately the following operations on the matrices R_{PT} and R_{CT}, passing as parameters also their corresponding `good_mask`, the SVD threshold, LSH number of rows and bands, and the K number of neighbors.

`r._nn()` first restricts the matrix to the non-zero rows according to the `good_mask`, and passes it to the `svd_embeddings` algorithm.

5.4.1 SVD.

5.4.1 SVD. `numpy.linalg.svd()` is the function from the NumPy library implementing the SVD algorithm. In order to get matrices of correct size, the function should be called with `full_matrices=False`.

For the low-rank approximation, the columns corresponding to the lower eigenvalues are removed from the `u` matrix. To get the upper bound on the meaningful column indices, it is first computed the cumulative sum `cs` of the `sigma` diagonal vector. Then `np.argmax(cs >= cs[-1] * thresh) + 1` gets the number of higher value eigenvalues carrying the percentage of information defined by the threshold parameter.

5.4.2 LSH. denser embeddings produced by SVD are then passed to the `lsh()` algorithm. Hyperplanes are generated by creating a matrix $|embeddings| \times B \cdot R$ with random values sampled from a Gaussian distribution. The matrix product `@` between embeddings and hyperplanes, followed by the `> 0` binarization, produces the `signatures` matrix.

The NumPy `split` function along `axis=1` divides the signatures into bands. The boolean signature features in the same band are hashed by interpreting them as binary representations of decimal numbers. The matrix product of the bands with the binary column $[2^0, 2^1, 2^2, \ldots]$ produces the decimal representations for the band, which can be stacked together again.

`defaultdict(set)` is the structure used for the bucketization. The pairs (band index, band decimal hashing) are used as keys in the dictionary to compose the different buckets. The decimal signatures matrix is iterated and the embeddings indices are added to the correct bucket.

By iterating on the buckets with more than one element, `itertools.combinations()` is used to generate all the pairs of embedding indices in the same bucket, which are added to the `candidate_pairs` set.

5.4.3 TopNN. Lastly, the candidate pairs are analyzed by `top_k_similar()`. The neighbors are initialized as a list of one empty priority queue (an empty list) for each embedding. For every candidate pair, the embeddings indices are first translated back to the original ratings tensor indices using the `good_mask`, since the embeddings were initially restricted to the non-zero rows. Each element of the pair is then added to the priority queue of the other element using the `heapq.heappush()` function, which implements the priority queues methods. The priority value is the explicitly computed cosine similarly. This data structure permits to efficiently retrieve just the K top nearest neighbors, and stack them in a neighbors matrix. The corresponding similarities are saved in a matrix of the same size. If not enough neighbors appear among the candidates, zeroes are used to fill the neighbors matrix. By setting their similarity to 0, they are not considered for the CF average.

The top nearest neighbors and their similarities, computed for the patients and the conditions, are then saved as class attributes. This concludes the procedures to initialize the system.

5.5 Querying the system

The predictions are computed by the `r.predict()` method. It first translates the patient and the patient condition IDs into the patient and condition indexes using the dictionaries saved as class attributes. If the IDs are not present or the `data_mask` indicates they have been discarded, the indexes are set to `None`.

The modified CF algorithm slices the `ratings` tensor with the vector of the nearest patients and conditions neighbors got from the TopNN matrices. If the query patient or condition are `None`, the corresponding axis is not sliced. The similarity score vectors are also sliced accordingly, and their outer multiplication produces the `weights` matrix. The global baseline matrix is also sliced with the nearest patients. The sliced global baseline is subtracted from the sliced ratings, multiplied by the `weights`, summed on the patients and conditions dimensions, and divided by the sum of the weights. The unsliced dimensions are evenly averaged. The result is then added to the global baseline of the query patient to produce the predicted therapies ratings.

If there are already tried therapies for the patient condition, they are removed from the candidates in the predicted therapies ratings. Then, the 5 therapies with the highest rating are converted back to the string IDs and returned by the system.

6 DATASET

The JSON dataset have been created with a Python script, heavily reliant on the BeautifulSoup library for the web crawling.

6.1 Conditions

The list of conditions is obtained from NHS Inform [2], a website which organizes illnesses by category. When inserted in the dataset, the category is saved as type attribute. An additional risk attribute, a random integer between 1 and 100, is added as an extra attribute.

6.2 Therapies

The list of therapies is created similarly, merging the list of therapies from Wikipedia [4] with the list of drugs from Drugs.com [1]. A list of words is downloaded from the MIT website [3]. Fifty of them are chosen as possible types, given to the therapies added to the dataset. The categorical "class" property is also added, having as value a random letter from A to F. The "strength" property is a random value from 0 to 1, with 4 decimal digits.

6.3 Patients

A parameter controls the number of patients randomly generated. A random sex, male or female, is randomly chosen for a patient. The `names` library exposes a function to get a random name and surname given a certain gender, which is used to generate the patient name. The patient birth date is chosen by selecting a random date in the interval going from the date of generation to 110 years back.

A sequence of patient conditions is generated at random and assigned to the patient. The maximum number is set at about four per year of life. The generator ensures that the time spans of two patient condition instances of the same condition do not overlap. With a 10% chance, patient conditions are marked as unresolved. Unresolved patient and patient conditions are also saved as candidates for the test set.

The list of trials is obtained by generating a random number of trials for each patient condition, with a minimum of 0 and a maximum that depends on the duration of the patient condition. The start and end dates of the trial are extracted from the condition time span without taking overlapping into account, assuming that their order is determined solely by the start date. Each trial also has a random numerical dosage property and a categorical frequency property that indicates how frequently the patient should use the therapy. If the corresponding patient condition was generated as cured, the success of the last therapy of the sequence is set to 100%, with the end of the trial equal to the patient condition end. For the other cases, the success is a random integer in the $[0, 99]$ range. A few therapies for uncured conditions are also generated with an `None` end date, to indicate an ongoing therapy.

7 EXPERIMENTAL EVALUATION

Several experiments were designed to evaluate the proposed solution. The first experiment assesses the system's performance under unseen patient conditions. The second one, on the other hand, evaluates the recommender just on 100% success trials, which are the last therapies in cured sequences. The third experiment compares the system's performance to that of a baseline method, a simpler condition-2-condition CF algorithm that is not based on the 3D ratings tensor. Finally, the fourth experiment assesses the system's scalability.

7.1 Evaluating on unseen patient conditions

In this experiment, the system should predict the first therapy to be administered to a patient suffering from a completely unseen

Table 1: Experiment on unseen patient conditions

Precision@5	32.85%
N° of folds	1000
N° of iterations	4
Success of untried therapies	5
Discount factor	0.9
SVD threshold	0.9
LSH Bands	400
LSH Rows	32
Patient Neighbors K	3
Condition Neighbors K	3
Memory	1.47 GB
Instantiation time	201 s
Case prediction time	11 ms

patient condition. This experiment's runs use an A/B N-fold testing strategy with a limited number of iterations (smaller than the number of folds, to speed up the testing). At first, each patient and condition combination is assigned a random fold number. Then, for each iteration, a different fold is chosen as test data and is masked out of the data used to initialize the model. The recommender then predicts the five best therapies for the patient and condition combinations in the test set that have at least one tried therapy. Because the ground truth therapies are the ones with the highest ratings in the unmasked ratings tensor, the correct therapies to predict are determined by the tradeoff between order and success of therapies set with the discount hyperparameter. The run results are obtained by averaging the results of each iteration.

Precision @ 5 is the metric used, and the relevant results are the top five ground truth therapies. One limitation of this metric is that for fewer than five relevant therapies, even a perfect system has a score of less than 1.

To save time, the hyperparameter combinations for the runs were chosen by hand. A grid search would be a more methodical approach to fine-tune the system before the distribution.

The best results obtained on P@5 are 32.85%. Table 1 displays all of the run's hyperparameters.

7.2 Evaluating on unseen successful trials

In this experiment, the system should predict the final therapy, the one that will cure the condition, to be administered to a patient who is suffering from a patient condition for which the previously tried therapies are available. Unlike the first experiment, the ground truth therapies are the 100% successful ones, so they do not depend on the tradeoff between success and order of therapies decided by the discount hyperparameter. This experiment's runs use an A/B N-fold testing strategy with a limited number of iterations. At first, the trials with a 100% success therapy are assigned a random fold number. For each iteration, a different test fold is masked out of the data. The recommender then predicts the five best therapies for the cases associated with the successful trials in the test fold. The run results are obtained by averaging the results of each iteration.

The metric used is the HitRatio@5, which is the percentage of predictions in which one of the suggested therapies is the therapy used in the successful trial.

The hyperparameters are the best values from the previous experiment's best run. Since this experiment was focused on the last trials only, a run of the experiment has also been tested with a discount factor of 1, which does not penalize later therapies.

Table 2: Experiment on unseen successful trials

HitRate@5	10.52%
N° of folds	1000
N° of iterations	4
Success of untried therapies	5
Discount factor	0.9
SVD threshold	0.9
LSH Bands	400
LSH Rows	32
Patient Neighbors K	3
Condition Neighbors K	3

Table 3: Baseline on unseen patient conditions

Precision@5	9.23%
N° of folds	1000
N° of iterations	15
Condition Neighbors K	3
Memory	14.7 KB
Instantiation time	31 s
Case prediction time	7 ms

Table 4: Baseline on unseen successful trials

HitRate@5	10.22%
N° of folds	1000
N° of iterations	15
Condition Neighbors K	3

As it can be seen in Table 2, the obtained HR@5 is compatible with the first experiment. Interestingly, the run with a discount factor of 1 obtained a slightly worse HR@5 of 9.82%.

7.3 Comparison with baseline method

In these experiments, it has been tested a baseline method to evaluate the improvements of the modified collaborative filtering algorithm. The baseline algorithm that has been used is a condition-2-condition CF, where the ratings matrix is obtained by averaging the ratings tensor along the patients.

The baseline method has been tested on the same modalities of the experiments explained in Section 7.1 and Section 7.2. The selected hyperparameters were also the same, with the exception of the number of iterations, which was increased because the baseline CF is significantly faster.

Table 3 displays the results of the experiment with completely unseen patient conditions. The baseline result is inferior to the system result. Table 4 displays the results of the experiment with unseen successful trials. The baseline result comparable to the system result.

In terms of resource consumption, the baseline method requires significantly less memory and is quicker to initialize. The case prediction time, on the other hand, is comparable.

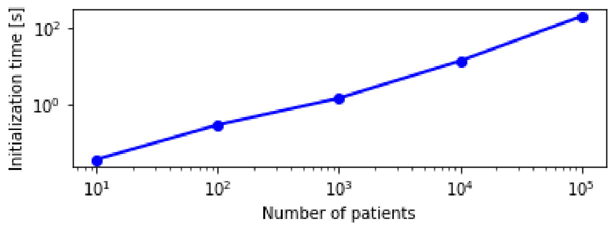

Figure 3: Initialization time scalability

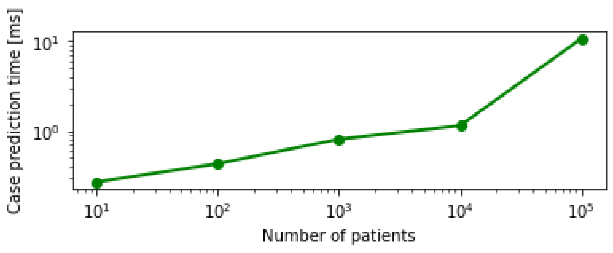

Figure 4: Case prediction time scalability

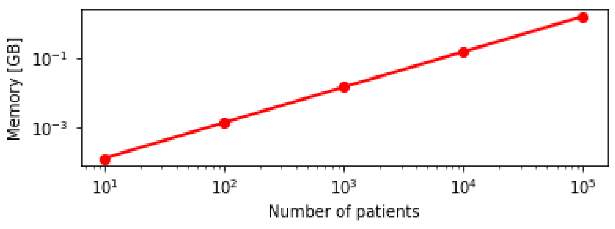

Figure 5: Memory scalability

7.4 Scalability

In this experiment, the system was tested with a growing number of patients to observe the relationship with the initialization time (Figure 3), the case prediction time (Figure 4), and the memory (Figure 5). The number of patients was chosen as the scalability parameter because it is the factor that is most likely to grow as the system collects more data. Increasing the number of trials only slows the building of the ratings tensor and has no effect on the remaining parts of the algorithm. The testing machine has a RAM of 24 GB, and an Intel Core i5-6500 CPU @ 3.2 GHz.

It can be seen that the number of factors considered grows linearly with the number of patients. Every patient occupies 14.7 KB of the allocated memory.

8 CONLUSIONS

Because the dataset on which the recommendation system was tested was generated at random, interpreting the performance achieved in absolute terms is misleading. In fact, data mining techniques are unlikely to produce good results in the absence of patterns in the data. However, when compared to a baseline Collaborative Filtering on a 2D matrix, the results obtained are at least similar in terms of Hit Ratio for conditions that have already been partially treated, if not superior in terms of accuracy for completely new cases. A comparison on a realistic dataset with real-world patterns would allow us to see if the higher complexity of the proposed algorithm can better model the problem.

As previously stated, aspects such as treatment order and additional dimensionality can be addressed by much more complex solutions than the proposed algorithm, which has the advantage of remaining relatively simple in its operation.

The algorithm is slightly intensive in terms of memory required and requires a significant amount of initialization time. However, it has been demonstrated that it scales linearly, which is a positive result. Furthermore, the response time to queries, which has been shown to be quite low, is more important than the initialization time, which is only relevant for the periodic updating of the available dataset.

REFERENCES

[1] Drugs.com. [n.d.]. Drug Index A to Z. https://www.drugs.com/drug_information.html.

[2] NHS inform. [n.d.]. Illnesses and conditions. https://www.nhsinform.scot/illnesses-and-conditions.

[3] MIT. [n.d.]. Word list. https://www.mit.edu/~ecprice/wordlist.10000.

[4] Wikipedia. [n.d.]. List of therapies. https://en.wikipedia.org/wiki/List_of_therapies.