A hospital recommendation system

Recommending medical therapies in a real-life case study

Gianluca Sassetti
University of Trento
gianluca.sassetti@studenti.unitn.com

ABSTRACT

This report describes the procedure to build a recommendation system that, given a dataset containing conditions, therapies and patients' clinical history, suggests the best therapy for a patient when input a specific, non-cured, condition.

1 INTRODUCTION

The aim of the project is to build a recommendation system such that, given a dataset containing conditions, therapies and patients with their own conditions and tried therapies, can recommend the five best therapies for a patient for a new, uncured condition. In the following sections there will be a brief introduction to the theoretical foundations for such systems (section 2, useful to understand the solution. Then, a more thorough statement of the problem (section 3. Following there will be the personal interpretation of the problem, followed by the solution (sections 4 and 5). Section 6 and 7 will provide an overview of the implementation (which is also discussed in the previous sections) and of the dataset. Finally the last two sections (section 8 and 9) will conclude the report with the experimental results and a critical analysis to extend the work of this project.

2 THEORETICAL FOUNDATIONS

There is not only one way to create recommendation systems. Nevertheless project implementations are built to overcome problems specific to the data fed to the algorithm.

In the following sections an introduction to the basic approach to recommendation systems and some ways to overcome its main issues.

2.1 Collaborative Filtering

Collaborative filtering is one of the most widely adopted solutions for recommendation systems. Users and item are represented in a $Users \times Items$ matrix, where each $M(x, i)$ contains the rating of the user x for the item i.

At its core, collaborative filtering tries to predict the rating of a user x for a given, unrated item i using the rating of other items rated by the user x, or the ratings of other users for the same item i. Therefore the name "collaborative".

When the known ratings are taken from other users we talk about *user-based* collaborative filtering, *item-based* otherwise. Usually, the known ratings used for the estimation are weighted by some similarity measure, either between items or users, so to increase the relevance of a rating with the similarity with our user or item.

To conclude the estimate is the result of the following:

$$\hat{r}_{x,i} = \bar{r}_x + \frac{\sum_{x' \in X} sim(x, x')(r_{x',i} - \bar{r}_{x'})}{\sum_{x' \in X} sim(x, x')}$$

Where $\bar{r}_x$ is the average rating user x and $sim(x, x')$ is some similarity measure between the users x and x'.

Nevertheless, this approach is far from perfect. Similarity measures often times can be too much of an approximation, even when using more accurate measures, like the *cosine similarity*. Furthermore, there are a few more challenges for this kind of systems, the reader can find more in the following sections.

2.2 Singular Value Decomposition

The singular value decomposition (SVD) is a factorization of a matrix M $n \times m$ in three matrices $U\Sigma V$, where Σ is a rectangular diagonal matrix, which entries are called *singularvalues* of M. U and V are both orthogonal matrices.

Most of the times computing the exact SVD of M is not important and we'd rather use an approximation. The matrix Σ is thus truncated with rank $k < rank(\Sigma)$. This still allows an accurate reconstruction of $M \approx U\Sigma V$, where the error is proportional to the number of truncated singular values.

The SVD of matrix can have many applications and is often used to reduce the dimensionality of the data. On top of that, a factorization of a matrix M can be used to estimate unknown values in M.

Collaborative filtering systems do not work well with very sparse matrices, but SVD solves perfectly this issue, as the reconstructed matrix has no unknown value.

When using the SVD to estimate unknown values in the input matrix, the objective is to "guess" a factorization of a matrix, or rather to approximate such factorization. To do so, two matrices nk and $k \times m$ are estimated. The second matrix is estimated as the product of Σ and V.

In order to have a good reconstruction of the input matrix, the objective is then to decrease the mean squared error on the known ratings of the input matrix. The best way to optimize the MSE is to use a gradient descent algorithm. The result of the process will be two matrices U, P s.t. $U \cdot P \approx M$ and the $MSE(M, U \cdot P)$ is minimized.

As a result the reconstructed matrix will have no unknown rating, thus avoiding all the problems typical of collaborative filtering when using sparse matrices.

2.3 Locality Sensitive Hashing

The Locality-Sensitive Hashing (LSH) is a commonly used algorithmic procedure to reduce the number of records in an input dataset. Records are mapped, through an hash function, to a given set of buckets in such a way that only similar items end up in the same bucket.

The technique can be also used for data clustering or nearest neighbour search. Despite being imprecise, and for this reason other techniques are usually preferred for clustering, LSH is extremely fast, as hashing requires constant time, and it can be used for initialization purposes.

Besides clustering, LSH is a powerful method to reduce the dimension of the input data, as it allows to select only as subset

of similar records that have been mapped to the same bucket. This is its use when implemented in recommendation systems. Not only the algorithms is sped up by cutting a large portion of the dataset, but also the recommending process is more accurate, as non-similar records are not taken into consideration anymore.

LSH worked wonders for this project: it increased the accuracy of predictions and made the algorithm run in feasible time.

3 PROBLEM STATEMENT

The goal is to build a system such that given in input a dataset containing a list of therapies, conditions and the clinical history of a set of patients, and input a patient identifier along with a condition identifier, outputs the five therapies that are considered to work best for the given patient for the given condition.

The dataset contains for each patient the list of all discovered conditions, and the therapies that have been tried for each condition. Such list of therapies is called "trials". It also associates with each trial a rating (from 1 to 100) of the effectiveness of the therapy for the given condition for the given patient.

The desired final system is therefore a recommendation system that will make use of the therapy ratings to output a ranked list of therapies.

4 PROBLEM INTERPRETATION

Outputting a ranked list of the best therapies, given a specific patient-condition couple, can be solved with a collaborative filtering approach.

In this case, the $Users \times Items$ matrix would have the Patients as the Users and the Therapies as the items. Accessing $M(x, i)$, with patient x and therapy i, would return a rating (*success* field in the dataset) if the therapy is present in the patient's *trials*, 0 otherwise.

Nevertheless, using only the $Patients \times Therapies$ matrix to solve the problem means outputting the best therapies overall for the given patient, without considering the condition. For this reason, a second matrix $Conditions \times Therapies$ is made, where we the entries are the average success rate of a therapy for a specific condition.

Through the rating, we can derive from the two tables the best therapies for our patient and the best therapies for our condition. The final output will be a list of therapies ordered by a score which was assigned them by aggregating the data from the two tables.

More specifically, each therapy t is assigned a score s.t.:

$$score(t) = \frac{sim_{pat}(p) \cdot P \times T_t}{\sum sim_pat(p)} + \frac{sim_{cond}(c) \cdot C \times T_t}{\sum sim_{cond}(c)}$$

where p and c are the input patient and condition; $P \times T_t$ and $C \times T_t$ are the columns of the therapy t from the $Patients \times Therapies$ and $Conditions \times Therapies$ matrices respectively. $sim_{cond}(c)$ is the vector of the pairwise similarity of the condition c and $sim_{pat}(p)$ is the vector of the similarities for the input patient.

5 SOLUTION

The data is initially loaded onto a $Patients \times Therapies$ and a $Conditions \times Therapies$. For the latter only the ratings of therapies for conditions that have been cured are considered, as there is no point in recommending therapies that cannot cure the input condition.

The two matrices, and in particular the one with patients, are extremely big. The best way to reduce the number of rows to a point at which the problem becomes treatable is to implement Locality-sensitive Hashing, as described in section 2.3. Thus for both matrices only a subset of rows that are consider similar by our hashing function is considered.

Estimating missing ratings using only other items' ratings, as in section 2.1, can lead to large inaccuracies. This is mainly because the implemented matrices are very sparse, and the similarity measures approximate too much. For this reason the values in the two matrices are estimated through an approximation of a matrix factorization (SVD, section 2.2).

Two matrices, called p and q, of size $Patients \times k$ and $k \times Therapies$ respectively, are initialized uniformly over $(0, 1]$. The values in p and q are adjusted over time to fit the data in the original matrix. To do so, a Stochastic Gradient Descent algorithm, with the objective of minimizing the mean squared error on the known values from the initial table, is implemented.

The final matrices are thus two smaller and complete $Patients \times Therapies$ and $Conditions \times Therapies$ matrices.

After this, a score, as described in section 4 is carried out for each therapy. The output of the program are the five therapies with the highest score.

5.1 Trial and error: other possible solutions

The development of the project followed a trial and error process: other solutions have been tested before the one shown could be implemented. The interested reader can still find old versions of the program in the files `hashing.py` and `analysis.py`

First, the two matrices used in the SVD were initialized with the values of the actual SVD. Second, the implemented optimization algorithm on the mean squared error was a standard gradient descent, and not Stochastic Gradient Descent. Both o these settings used to lead to a slower convergence of the algorithm. On top of that, it was way harder to optimize bias vectors in the case of standard gradient descent. For this reason, in the final version the matrices P and Q are initialized uniformly and SGD with bias estimation are implemented.

5.1.1 First approach as a baseline evaluation method. The first implementation of the software did not use any factorization of matrices or optimization algorithms. It simply loaded the data onto the matrices and carried out the estimations for missing values as:

$$\hat{r}_{x,i} = \bar{r}_x + \frac{\sum_{x' \in X} sim(x, x')(r_{x',i} - \bar{r}_{x'})}{\sum_{x' \in X} sim(x, x')}$$

where the similarity measure was the cosine similarity between patients. The results for this technique were, as expected, underwhelming.

When tested against known values of the matrices, the aggregation function seen above output results with large errors. Moreover, the estimations for unknown values were not meaningful: every estimated value was approximately in the interval of $[0.45, 0.55]$.

The next step was to try an optimization algorithm that would estimate a set of parameters by reducing the mean squared error between the known values and the estimates. Basically, instead of the cosine distance, a set of parameters w was used. The estimates were thus carried out as:

$$\hat{r}_{x,i} = \bar{r}_x + w_i \cdot r_i^T$$

The parameters were estimated by optimizing through gradient descent the mean squared error between the estimated and the known values of the matrix.

In this case, despite having lowered significantly the MSE (>0.1), the estimates on unknown values kept being meaningless, as their values were all in rather small interval, that was either $[0.95, 1]$ or $[0.45, 0.55]$, therefore the estimates where all quite the same.

Finally, also another approach was tested, but with very poor results. In this case, only the row of our current patient was estimated, as in the following:

$$\hat{r}_{x,i} = \bar{r}_x + w \cdot r_x$$

The results were extremely inaccurate: the MSE against known ratings was always very high (>0.2). This implementation was therefore dropped and not used again. Although, I needed to known if the approach was viable.

Nevertheless, these approaches can still be used as baseline evaluations to show how the new approach gave better results.

6 IMPLEMENTATION

In this section a brief discussion of the tools, parameters and settings used for the algorithm.

The project focused mainly on making a good estimation of the unknown ratings in the matrix, thus using mainly a gradient descent optimization algorithm. For this purpose, Pytorch, a very well known machinie leraning Python library was used. The high compatibility between Pytorch and NumPy allowed great speed and flexibility in the phases in which the data, after being loaded from JSON file, was to be transformed into a Pytorch compatible tensor enabled for the optimization algorithm.

The number of latent factor for the SVD approximation was 40, the learning rate for the algorith 0.001 and the number of epochs roughly 20000 in the case of the standard gradient descent, and 100 in the case of the stochastic gradient descent (SGD allows a much faster convergence). These are all numbers with which I came up during the testing phase: they are all manually adjusted to try to trike balance between a good estimation accuracy and fast enough estimation time.

7 DATASET

The dataset in composed of three main elements: Patients, Conditions and Therapies. Each patient, additionally, has a clinical history composed of a list of Conditions, each of which refers to a condition present in the Conditions list, and a list of Trials. Trials are a list of therapies that a patient has tried to cure a diagnosed condition with. Each trial is thus specific to a triple (patient, condition, therapy) and has a success field, a value between 1 and 100, associated with it. The success values have been used as ratings to populate the matrices of the recommendation systems.

In the delivered dataset conditions and therapies were scraped from the two websites provided in the project presentation document.

Patients were assigned a random clinical history. Namely, the list of personal conditions and trials were chosen at random. The length of the lists was randomly generated with an upper bound of a hundred conditions per patient. Then, a number of conditions were selected randomly, with uniform probability, over the set of all scraped conditions. For each selected condition, a number of therapies were selected, always with uniform probability, over the list of all conditions.

This kind of synthetic dataset surely cannot be used in a real world scenario, but it is the best way to develop a recommendation system that has to be semantically abstracted from the content of the dataset. As the aim of the project is not to find secret correlation between patients and therapies, but to just recommend therapies in a good way when being fed any kind of data, the dataset as presented before, was thought to be suitable for testing purposes.

8 EXPERIMENTAL RESULTS

Testing and experimentation over time has shown that the solution that reconstructed the matrices through an estimation of a factorization them was the best solution overall in terms of minimization of the reconstruction error. On top of that, the reconstructed matrices were populated with meaningful values: values that had variance. I also extensively talk about this in section 5.1.

When compared with the "standard/basic" collaborative filtering, where unknown values are estimated just through an aggregation function of other ratings (either in the same column or row), the values reconstructed as in the presented solution have greater variance, thus outputting meaningful estimates. With the "basic" collaborative filtering the values would all be in a rather small interval, like (0.45, 0.55), which meant that any therapy would be half effective. The very meaning of recommendation system is to output suggestions for things that would likely work, and this was clearly not possible with every suggestion being always around 0.5.

On top of that, the presented solution minimized the reconstruction error on the known values. Namely the error was made infinitesimal, which is a great achievement, stating the efficacy of the technique. Other estimation methods that I tried, as in section 5.1, cannot bear the comparison, as their MSE on the known values was as low as 0.1 at best.

Furthermore, the presented solution implements LSH. LSH showed during tests that it could increase the accuracy of the estimations, as only a subset of records was selected. For the same reason, LSH makes the entire procedure scalable. Outside of the testing environment, when fed the real dataset, the algorithm can output the recommendations in a matter of minutes, depending on the length of the training for the matrices.

The solution is overall satisfying even though it lacks and implementation of time, section 9 for more.

9 FURTHER WORK

Further work can try to take time into consideration in the estimation process.

In one of the last version of the software, I tried to implement time by adding a bias to each estimation $Condition \times Therapies$ matrix. This bias would have been negative if rating of a therapy has a negative trend over time, and positive for positive trend.

In the implementation, the derivative of the ratings of the therapy as a function of time. Practically, the idea works only if the trends of the ratings are monotone, either increasing or decreasing, but not for the actual data. Such implementation lead to carry out misleading biases: the trend of therapy might be increasing, but the bias negative even if the last rating slightly decreased from the previous one.

After a first testing phase the idea was completely discarded: the ratings, that uniformly distributed, are more likely to have a lot of "up and downs", thus making the biases not only misleading, but also incorrect. The simple average of the rating over time was a much better bias.

Future versions of the software can try to implement time and should avoid at all costs to do it in the way I did.