Data Mining Final Project[*]

Recommendation System with Collaborative Filtering

Gábor Vitrai
University of Trento
2nd Year, EIT DSC
gabor.vitrai@studenti.unitn.it

ABSTRACT

This document was created for the Data Mining Subject as a final - end of semester - assignment. The project was divided into two segments.
1. Data Generation, where I had to come up with a method to generate a dataset with the given attributes and structure.
2. I had to implement a recommendation system which gets a set of patients, conditions, and therapies to learn from. From this data, the code gets 3 main inputs. The dataset, a patient ID and a condition ID. The application has to return 5 recommended therapies for the given patient for the requested condition based on what therapies and conditions other patients had.

1 INTRODUCTION & MOTIVATION

1.1 Short Task

We were asked to create a system to help doctors suggest the most prominent therapy for a patient to for a specific condition he or she may have.

1.2 Task in Detail

We have a set of patients. A patient is represented as an entity. She/he has an ID, and a set of other attributes that describe the characteristics of the patient. The mandatory attributes are: ID, Name, Conditions and Trials.

Patients have a set of conditions. A condition is also an entity that has an ID and a set of attributes. A condition is a medical problem that a patient has. Example given by the lecturer: For instance, allergy to wool is a condition. Eye dryness is another condition. Blood pressure of more than 20% is a third condition.

When a patient has a condition, doctors prescribe a therapy. A trial is a therapy that has been applied to a specific patient for a specific condition. The trial is a tuple (Therapy, Patient, Dates, Success, ID). Therapies are not always fully successful. They may be even completely unsuccessful. This is why the success field is used to record how successful the trial was.

The medical history of a patient is a list of the trials he/she has gone through and the respective conditions these trials were done.

1.3 Formal Requirements for Running

The task is to create a project where the formal requirements are the following:
Input 1: A set

$$P$$

of patients, their conditions, and the ordered list of trials each patient has done for each of his/her conditions (i.e, his/her medical history)
Input 2: A specific patient

$$P_{[q']}$$

his/her conditions, the ordered list of trials he/she has done for each of these conditions (i.e, his/her medical history).
Input 3: A condition

$$c_{[q]}$$

This means that to run the program we need to provide 3 arguments:

```
code.py dataset.json patientID conditionID
```

output: Ordered list of 5 recommended therapies for the patient with patientID for the condition with conditionID.

2 RELATED WORK

I distributed the task into different segments:

(1) Demo dataset creation
(2) Initial checks - Condition / Patient has condition.
(3) Generating the vectors of therapy success rated.
(4) Finding the most similar k patients using cosine similarity.
(5) Finding the best therapies with predictions.
(6) Validation

1. For the dataset creation, I will start with creating a tiny dataset to see the initial structure of the classes. This set only consisted of a few patients and other classes, but it was necessary to test the patient_condition and patient_therapy classes. Later this structure was used to generate the big dataset with 50.000 patients.
2. Initial checks are needed to check parameters, to parse the data and to check if the algorithm needs to find a cure to a condition which the patient did or didn't have before. These minor checks are required for the fluent execution of the system.
3. For each patient a vector will be required. To generate these, I will use a self made method (see algorithm later) which will make further processing and similarity measures possible. For each patient, for every therapy, the vectors will be built from the success rates.
4. To find the similar patients, I will be using collaborative filtering and to determine which rows are similar, I am using the cosine similarity between two previously generated vectors. These vectors will be normalized and then compared.
5. Finding the best therapies will be based on the similar users found in the previous step. The algorithm will calculate an average of the success rates and make a prediction based on that. After a specific sorting, the conclusions can be withdrawn.
6. Validation will consist of manual proving of the similarity search and of comparing predictions against withheld known ratings.

[*] Recommendation System for Patients

3 PROBLEM STATEMENT

3.1 What is Collaborative Filtering

Here the main idea is to find a set of N patients whose therapies are similar to the other requested patient, and estimate the possible success rate based on the success rate of each therapy in the case of the similar patients.

- It needs data on other users (which we have and we need)
- It is able to recommend new unpopular therapies.
- It is hard to recommend for new users (We do not have new users as each of them has quite a few **conditions** and **therapies**.

3.2 Why I used Collaborative Filtering

In case of sicknesses / conditions and therapies, it is important to take into consideration similar patients.

- Collaborative filtering needs to know the item which we need to rate. (Therapies are already rated by similar users)
- No need for feature selection
- Needs a lot of patients (We have 100.000)
- No "unique taste" is required, as conditions do not develop on "taste".

3.3 User-User / Item-item collaborative filtering

Since content based approach is good if we don't have patient data (it doesn't necessarily need to have the patient data), however in this exercise, that's what we have to work with so the good solution is to use collaborative filtering. This will take into consideration what therapies were used for the given condition in the case of other patients.
In the case Item-item collaborative filtering we estimate success rate of therapy **i** based on the success rate of similar therapies.

4 SOLUTION

For the solution, I used the known success rate of the therapies for the **given condition** to make a prediction on an item-item basis.
To find the **k** similar patients, I used **user to user filtering** as I had to take into consideration multiple aspects of a patient. Later to predict the success rate of a therapy for a given condition, I used cosine similarity to determine the difference between the vectors of each user. (Relative to each therapy). I didn't seem to have the use of clustering for this specific task, as due to the diverse structure of the dataset, and different classes, it would have been very difficult to extract the data we need for clustering. For further improvement however, speed is definitely a big issue, so later Locally Sensitive Hashing or clustering might should be introduced for similarity search.
Throughout the work, I mention a few definitions. Here I listed 5 different mathematical formulas which were used for the project.

Cosine Similarity:

$$\cos(\mathbf{v1}, \mathbf{v2}) = \frac{\mathbf{v1} \cdot \mathbf{v2}}{\|\mathbf{v1}\|\|\mathbf{v2}\|} \tag{1}$$

The similarity was used to determine the similarity of two patient vectors, where each vector consisted of the normalized success rated of the therapies for a given condition.

Success rate Prediction:

$$r_{xi} = 1/k * \sum_{i=1}^{n} {}_{y \in N} r_{yi} \tag{2}$$

To calculate the points which the recommendation system will decide with, I calculated average as we have learnt it on the lecture.
Vector Normalization

$$\lambda\vec{x} : \vec{x} - mean(axis = 1) \tag{3}$$

Vectors got normalized by subtracting the row means, this way the filtering handles missing ratings as "average" and will capture intuition better.

Root-mean-square-error:

$$\sum_{i=1}^{n} (r_{xi} - r^{*}_{xi})^2 \tag{4}$$

For validation I used Root Mean Square Error as it was recommended on lecture and is a good measurement tool.
Rank Correlation:

$$\cos(p) = \sum_{i=1}^{n} \frac{(\mathbf{x_i} - \mathbf{x})(\mathbf{y_i} - \mathbf{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \vec{x})^2 \sum_{i=1}^{n} (y_i - \vec{y})^2}} \tag{5}$$

Spearman's rank correlation was used which assesses how well the relationship between two variables can be described using a monotonic function.

5 IMPLEMENTATION

As mentioned earlier, I distributed the work into separate segments, and I followed these for the implementation. In this segment, I will write about these parts in detail in terms of implementation.

5.1 Demo dataset creation

The implementation of the dataset creation is simple. A loops go further the 3 different data types and based on the source files (txt with sample names, conditions and therapies). Random but reasonable, sequential dates are generated for the trials.
Afterwards, patients are created, trials and conditions are assigned to newly generated patient based on intervals and rules to make the dataset feel like real.

5.2 Initial checks - Condition / Patient has condition

5.2.1 Primary Checks. First, the script checks if the format of the input is correct and tries to find the requested patient and the corresponding condition based on the provided ID and arguments. In case any of these steps fail, the script exits. Since each entity (especially trials and conditions) have multiple foreign key, many additional methods were created including lookup methods like find_condition_by_id(condition_id).

5.2.2 Condition requirements. When a user runs the script, if the condition exists, the program needs to check if the patient has the given condition in its medical history. If this is not the case (like in real life, a newly diagnosed condition), and the condition is new, it has to be added.
First the code generates a new date for the condition, then it creates the new entity and adds it to the list.

```
start_date, end_date = get_new_dates_with_interval()
p_c = Patient_condition("pc" +
str(get_new_patient_condition_id(patients)),
start_date, end_date, condition_id)
patient.__dict__["conditions"] += [p_c]
```

5.2.3 Finding Similar patients based on condition. Optimization needs to be made as, with 100.000 patients, (the dataset provided by the instructor) running the script would take a lot of time, thus we do not need all the patients. As a first step, we only need to get information of patients who had the same requested condition diagnosed.

This method later could be improved with clustering or LSH, however in this case, with only a few values for each patient, matrices make sense.

5.3 Generating the vectors of therapy success rated

The implemented *generate_vectors(patients, conditions_id, patient_id, norm_vectors)* method generates and returns the vectors of success rates of each therapy for the patients in the given set of patients. The vectors show the success rate of each therapy for each patient for the GIVEN condition. I made normalization optional with a default parameter (norm_vectors), to make readable data sheet.

Algorithm 1 Vector Generation

```
procedure GENERATE_VECTORS($patients, cond\_id, pat\_id$)
    $therapy\_ids \leftarrow get\_all\_therapies(patients)$
    $vectors \leftarrow []$
    for each p in patients do
        for each t in therapy_ids do
            if t in trials_of_patient then
                $success\_rate \leftarrow therapy\_success\_rate(p, t)$
                if t was used to cure condition c then
                    $vector.append(success\_rate)$
                else
                    $vector.append(0)$        ▷ If used for other c
                end if
            else
                $vector.append(0)$
            end if
        end for
        $vector.append(p.\_\_dict\_\_["id"]$
        $vectors.append(vector)$
    end for
    $df = pd.DataFrame(vectors)$        ▷ Create DF from array
    if norm_vectors then
        # Normalize scores by subtracting row means
        $df = df.apply(lambdax : x - df.mean(axis = 1))$
    end if
    return df
end procedure
```

The output is what we need, - for further conclusion - can be seen below (extracted). The table consists of all the patients who had the condition (rows) and all the therapies which were ever used to attempt to cure the sickness (columns).

```
Example      Th1       Th10       Th11      Th154
7      -8.352941 -8.352941 -8.352941 -8.352941
41     -1.176471 -1.176471  0.000000 -1.176471
118    -7.039216 52.960784 -7.039216 -7.039216
```

5.4 Finding the K most similar patients using cosine similarity

The next step is to find the most similar **k** patients. On the lecture, we did not define what number **k** should be, so I analyzed the data manually, and I used the top 10% of the most similar patients. Using the generate_vectors(...) function, I generated the vectors of each patient and I removed the selected patient to a separate variable. Afterwards, the function iterates over the patients and removes patients with **zero vectors**. Zero vectors occur, in case a patient was not treated for the requested condition, so we can not make a prediction based on them.

Patients with usable vectors (not null) are selected, and for each of them, a similarity score is calculated. Since the vectors were already normalized we can apply the cosine similarity. With the help of numpy, it is easy to implement.

```
def cosine_sim(v1, v2):
    return np.dot(v1, v2) /
    (np.linalg.norm(v1) * np.linalg.norm(v2))
```

Cosine similarity captures intuition better. It Threats missing scores as "average", and handles outliers better.

As a final step, the script sorts the vectors (patients) based on similarities to the requested patient and gets the upper 10% of them. The output is a set of patient records.

Algorithm 2 Getting Similar Patients

```
procedure GET_SIMILAR_PATIENTS($n\_pat, p, patients, c\_id$)
    $patients\_vectors \leftarrow generate\_vectors(...)$
    $patient\_vector \leftarrow patients\_vectors.loc[patient["id"]]$
    $patient\_similarities \leftarrow []$
    for index, row in patient_vectors.iterrows() do
        if row is zero vector then
            $patient\_vectors.drop(index)$
        else
            $sim \leftarrow cosine\_sim(row, p\_vector)$
            $patient\_similarities.append(sim)$
        end if
    end for
    $ids \leftarrow patients\_vectors.index[$
    $np.argsort(patient\_similarities)[:: -1]][: n\_p]$
    return [find_patient_by_id(id) for id in ids]
end procedure
```

5.5 Finding the best therapies with predictions

The last step is to predict the possible success rate of each therapy for our patient. For this, the script uses the same generate_vectors(...) function without normalization. This generates the table what we need. The output looks the following: (One of the examples)

```
(PatientID: 6 ConditionID: Cond248)
        Th10  Th29  Th32  Th40  Th46  Th48  Th49
ids
52547      0   100     0     0     0     0    51
76231    100     0     0     0     0    33     0
58579      0     0    57    54   100     0     0
```

```
7602     0  100    0    0    0   55    0
47295    0   67    0    0    0    0    0
44552    0    0    0    0  100    0    0
93410    0    0    0    0    0    0  100
9705     0    0    0    0  100    0    0
46901    0    0    0    0    0    0  100
46705    0    0    0    0  100    0    0
55640    0    0    0    0    0    0   43
8250     0    0    0    0  100    0    0
51257    0    0    0    0  100    0    0
30229    0    0    0    0    0    0  100
33729    0    0    0    0    0    0  100
72251    0    0    0    0    0    0  100
31181    0    0    0    0  100    0    0
43879    0    0    0    0    0    0  100
36041    0    0    0    0  100    0    0
41606    0    0    0    0    0    0  100
```

To find the expectable efficiency of a therapy for our condition, I used the formula mentioned above. If we do the lookup, we can see that Patient 6 and 52547 got similar therapies, and for condition Cond248 they were treated with different conditions. (They are similar because they got similar therapies, while to cure the condition they got different ones.)

Algorithm 3 Finding The Best Therapies

```
procedure FINS_BEST_THERAPIES($patient\_vectors, n$)
    $n\_patients \leftarrow patients\_vectors.shape[0]$
    $n\_therapies \leftarrow patients\_vectors.shape[1]$
    $recommendation\_rates \leftarrow []$
    for therapy_id in range(n_therapies) do
        $avg \leftarrow (1/n\_patients) * patient\_vectors.iloc[:, therapy].sum()$
        $recommendation\_rates.append(avg)$
    end for
    return $patient\_vectors.columns[$
    $np.argsort(recommendation\_rates)[:: -1]]$
    $[: n_recommendations].to\_list()$
end procedure
```

5.6 Implementation of Validation

To implement validation, i decided to create an external script (test.py) to run the original recommendation system. (code.py). The script only consists of 3 methods aside the main function. 2 of them are for evaluation measurements and the 3rd one is running the recommendation system.
First, the script randomly selects elements for the test and stores its attributes.
Next, the recommendation system is run on the specific attributes in a specific configuration, and returns the estimated success rate for the given patient, condition, and therapy. So the output is a single floating point number.

Knowing these results, the algorithm calculates the Root-Mean-Square-Error and the Rank correlation. The results can be found in the last section.

Algorithm 4 Testing Success Rate

```
procedure TEST_SUCCESS_RATE($p\_id, c\_id, n\_samples$)
    $selected\_values \leftarrow []$
    $y \leftarrow []$
    for value in range(n_samples) do
        $random\_th \leftarrow df.sample(1).index$
        $random\_cond \leftarrow random.sample(df.columns, 1)$
        $success\_rate \leftarrow df.loc[random\_th][random\_cond]$
        if success_rate != 0 then
            $y.append(success\_rate)$
        end if
    end for
    for configuration in y do
        $run(code.py)$                          ▷ Differs from real code
    end for
end procedure
```

Algorithm 5 Calculating Root-Mean-Square-Error

```
procedure RMSE($y\_pred, y$)
    $rmse \leftarrow 0$
    for i in range(y_pred) do
        $rmse \leftarrow rmse + (y\_pred[i] - y[i] * *2)$
    end for
    return $np.sqrt(rmse/len(y\_pred))$
end procedure
```

6 DATASET

6.1 Data Structure

The Structure of the set is built on 3 main classes: Condition, Therapy, Patient. To support the rest of the requirements with the dates and success rates, two additional classes were created. These are the Patient_Condition and Patient_Therapy. These are similar classes with the requested additional attributes, functioning as a sort of connector class / table.

6.2 Data Generation

The generated dataset contains 100 patients, 321 conditions and 22 therapies. The number of available therapies were reduced to 22 so this way, the random therapy selection will result in some of the available therapies being actually useful for some of the conditions. It was randomly determined how many conditions and trials a patient can have. (1-5 each)
To generate the dataset, first I downloaded and filtered the conditions, therapies, and patient names from the given resources [1][2][3]. To do so, I implemented some fairy simple scripts with a common baseline:

```
import random library;
open resource file;
select X random names;
capitalize names;
sort names;
export output;
exit;
```

I implemented helper functions to generate dates for the day of the diagnosis and ending of the therapy. Date generation pays attention to the beginning of the therapies so they start on the day as the condition was diagnosed. Therapy generation also influences the final outcome of the conditions. If a therapy was more then 75% effective, it is considered successful. This case the condition is cured for the given patient. Otherwise the "cured" attribute remains "null".

6.3 Export / Import

To have the ability to sufficiently transfer the data between python dictionaries and JSON files, I implemented the needed import and export methods using the python JSON library. These methods can handle the embedded connections between patients and conditions/therapies with the connector classes involved. (Later lambda hook methods were removed as the dataset we got contained other not relevant - from the point of the recommendation - attributes but the classes could not support them).

6.4 Patient Observations Regarding Data Generation

In the example below, we can observe how Patient number 70 has 2 conditions. Condition number 229 was attempted to be cured 2x, the first therapy failed with a 15% success rate, while the second therapy (Th2) managed to cure the condition. Note that the first therapy was started when the condition was diagnosed, and the second therapy was started on the day when the first therapy was finished. (Possibly at a visit with the doctor).

```
{
"id": 70,
"name": "Rizzuti",
"conditions": [
    {
        "id": "pc146",
        "diagnosed": "20161003",
        "cured": "20201104",
        "kind": "Cond229"
    },
    {
        "id": "pc147",
        "diagnosed": "20120413",
        "cured": null,
        "kind": "Cond191"
    }
    ],
"trials": [
    {
        "id": "tr180",
        "start": "20161003",
        "end": "20190709",
        "condition": "pc146",
        "therapy": "Th16",
        "successful": 15
    },
    {
        "id": "tr181",
        "start": "20190709",
        "end": "20201104",
        "condition": "pc146",
        "therapy": "Th2",
        "successful": 83
    }
    ]
}
```

7 EXPERIMENTAL EVALUATION

7.1 Results

We received 10 test cases by the lecturer to test the application with. The results can be seen below. Since we did not specify what the K number should be, in the case of most similar users, I did two tests. First I used the top 5 most similar patients, however as the overall amount of patients is significantly larger, so I decided to go with the top 10%.

First I show the evaluation using the top 10% as this is meant as the FINAL SUBMISSION FOR THE PROJECT. The recommendations using the top 5 users are there only, to show the difference between the results depending on the amount of data we evaluate.

7.1.1 Appearance of a recommendation. The output is a list of the 5 most recommended therapies. For the user, it appears in the form shown below, however to report the outcome of the code, I collected them into an array like format in the next subsection.

```
Recommended therapies for patient with id 6
for condition with id Cond248:
1.: (id: Th29) magnetic resonance therapy - Physio
2.: (id: Th46) stem cell therapy - Physio
3.: (id: Th49) systemic therapy - Sugar
4.: (id: Th18) gold standard therapy - Physio
5.: (id: Th10) curative therapy - Acid
```

7.1.2 Top 10% Most Similar Patients.

```
P-ID   P-Cond     Recommendations in Order
6      pc32       [Th29, Th46, Th49, Th18, Th10]
51345 pc277636    [Th22, Th32, Th50, Th47, Th25]
82486 pc445475    [Th14, Th31, Th25, Th26, Th5 ]
51348 pc277652    [Th16, Th8,  Th47, Th43, Th51]
51358 pc277696    [Th4,  Th16, Th31, Th12, Th18]
51362 pc277711    [Th35, Th11, Th50, Th33, Th30]
51366 pc277723    [Th30, Th11, Th1,  Th31, Th33]
51387 pc277825    [Th32, Th14, Th16, Th8,  Th31]
51416 pc277986    [Th10, Th11, Th50, Th12, Th22]
51453 pc278191    [Th38, Th8,  Th40, Th37, Th29]
```

7.1.3 Top 5 Most Similar Patients.

```
P-ID    P-Cond        Recommendations in Order
6      pc32       [Th46, Th49, Th29, Th10, Th48]
51345 pc277636    [Th50, Th22, Th32, Th25, Th47]
82486 pc445475    [Th5,  Th26, Th25, Th14, Th35]
51348 pc277652    [Th16, Th51, Th47, Th43, Th8 ]
51358 pc277696    [Th4,  Th16]
51362 pc277711    [Th50, Th26, Th11, Th35, Th30]
51366 pc277723    [Th31, Th1,  Th30, Th5,  Th12]
51387 pc277825    [Th8,  Th32, Th31, Th14, Th16]
51416 pc277986    [Th12, Th50, Th10, Th26, Th11]
51453 pc278191    [Th38, Th8,  Th40, Th36, Th14]
```

7.1.4 Difference Evaluation. As we can see, the patients are the same, however we have a lot of differences in the order of the recommendations, and even in the selected therapies.

The first thing which hits the eye is that in the case of the 5th test case, with only 5 patients used for computing, the algorithm couldn't come up with 5, but only 2 recommendations.

If we observe content of the lists, most of the therapies are the same, but the order is different. For example, in the case of patient 6, therapies **Th10, Th29, Th46, Th49** are the same, which is 4

Table 1: Patient Similarity Validation (Requested Patient: 58

Patient ID	Conditions	Therapies	Has Condition?
58	113,234,207	**6**,12,120	O
26672	**185**,253,277	**6**	X
11854	**185**	**6**	X
3083	**185**	**6**	X
11577	**185**,296	**6**,18	X
38806	1,50,**185**	**6**,18,2	X

out of 5. These behaviours can be observed across the ten test cases.

7.2 Validation

7.2.1 Validation of similar patients on 100.000 records. Let's see the #1. most similar patient in the example of section **6.7**! Patient 6 was requested for condition 248. If we analyze the input, we can see that the patient had the following trials. (Sets)

- Patient 6: **3**,10,13,**18**,**29**,**32**,40,46,48,**49**
- Patient 52547: **3**,6,**18**,19,20,22,**29**,**32**,34,37,42,43,**49**
- Common ones: **3,18,29,32,49**

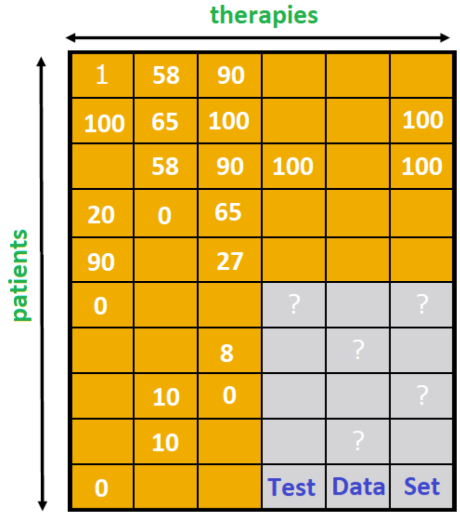
The two patients had 5 common therapies, thus 2 of them were used for the requested condition (248). This is high similarity.

7.2.2 Validation of similar patients on 20.000 records. With less patients in the database and with a reduced amount of available therapies and limited number of conditions and therapies, I tried to show that patients recommended by the system are indeed similar.

7.2.3 Comparing predictions against withheld known ratings. As described earlier in the implementation part, the algorithm selects a subset of results to be the test set. These elements look like following: *(Patient ID, Therapy to test for, true success rate)*

```
[[29102, 'Th37', 29],  [44552,  'Th46', 100],
[78951,  'Th10', 100], [47046,  'Th46', 64],
[68221,  'Th32', 16],  [95582,  'Th49', 100],
[74514,  'Th15', 57],  [6915,   'Th33', 31],
[26402, 'Th34', 100],  [35797, 'Th46', 18]]
```

Figure 1: Evaluation Example



1	58	90			
100	65	100			100
	58	90	100		100
20	0	65			
90		27			
0			?		?
		8		?	
	10	0			?
	10			?	
0			Test	Data	Set

To show concrete examples, here are the predicted success rates compared to the true values. We can see that to known high values, the predicted success rates tend to be a high number (above 70), while to known small numbers, the numbers are usually closer to the real value. An other observation is that sometimes small values are predicted to be high, this might be due to high success rates in the case of other patients.

Table 2: Patient Similarity Validation (Requested Patient: 58

Y Pred	Pred	Diff
64	29	-35
80	100	20
70	100	30
75	64	-11
59	16	-43
83	100	17
80	57	-23
82	31	-51
66	100	34
32	18	-14

The calculated RMSE is **32.3239** which is quite low, however with these values, with this dataset, the differences numerically were quite big, **while the predictions are actually not far at all.** The Rank correlation also turned out to be low (**37.2417**) due to the above mentioned reason.

REFERENCES

[1] Used Name Dataset https://github.com/philipperemy/name-dataset. Last accessed 3. Jan. 2022.

[2] Used Conditions Dataset, https://www.nhsinform.scot/illnesses-and-conditions/a-to-z. La st accessed 3. Jan. 2022

[3] Used Therapies Dataset, https://en.wikipedia.org/wiki/List_of_therapies. Last accessed 3. Jan. 2022

[4] Spearman correlation, https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient. Last accessed 17. Jan. 2022

[5] J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org/. Last accessed 15. Jan. 2022

[6] Cosine Similarity, https://en.wikipedia.org/wiki/Cosine_similarity. Last accessed 4. Jan. 2022

[7] Root Mean Square Error, https://en.wikipedia.org/wiki/Root-mean-square_deviation. Last accessed 10. Jan. 2022