

**Polytechnic Institute of Coimbra**  
Institute of Accounting and Administration of Coimbra

Liliana Carina Pereira Brandão

## Wavelet-Based Cancer Drug Recommender System

Wavelet-Based Cancer Drug Recommender System

Liliana Carina Pereira Brandão

ISCAC | 2020

Coimbra, July 2020





## **Polytechnic Institute of Coimbra**

Higher Institute of Accounting and Administration of Coimbra

Liliana Carina Pereira Brandão

### **Wavelet-Based Cancer Drug Recommender System**

Project submitted to the Higher Institute of Accounting and Administration of Coimbra in fulfillment of the requirements for the Degree of Master in Data Analytics and Decision Support Systems under the supervision of Doctor Fernando Paulo Belfo and co-supervision of Doctor Alexandre Gomes da Silva.

Coimbra, July 2020

## **RESPONSABILITY TERM**

I declare to be the author of this project, which is an original and unpublished work that has never been submitted to another institution of higher education to obtain an academic degree or other qualification. I also certify that all quotations are properly identified and that I am aware that plagiarism constitutes a serious lack of ethics, which could lead to the cancellation of this project.



To my Parents –  
Brandão and Palmira.

## **ACKNOWLEDGMENTS**

This research project would not had been possible without the enriching academic and professional events that I was fortunate to experience from the moment I enrolled this master. That moment was a milestone and a turning point in my life.

I want to use this opportunity to acknowledge my supervisor – Professor Fernando Paulo Belfo – whose attention to detail guided this work in such a valuable way, and my co-supervisor – Professor Alexandre Gomes da Silva – for his always pragmatic point of view.

I would also like to thank my Erasmus supervisor – Luigi Roggia – who offered me a unique working experience, in an amazing project, within a high-skilled team. That wonderful period of time shaped me in brand-new ways and, therefore, positively impacted this work.

Last but not least, I have been blessed with a supportive family and outstanding friends. I will not mention them by name (I do not want to risk leaving anyone out), but I must state that without them this journey would not had been so pleasant and colorful.

## ABSTRACT

Molecular nature of cancer is the foundation of systematic studies of cancer genomes, providing exceptional insights and allowing treatments advancement in clinic. Moreover, they are motivating the clinical use of genomic information to make otherwise unexpected treatment decisions for patients with a wide range of cancer types, rendering precision medicine possible.

Having this in mind, we combine techniques of image processing, for feature enhancement, and recommender systems for proposing a personalized ranking of cancer drugs. The system is implemented in Python and tested using a database containing drug sensitivity data for more than 310.000  $IC_{50}$ , describing response of more than 300 anticancer drugs across 987 cancer cell lines.

After several preprocessing tasks, regarding drug sensitivity data, two experiments are performed. First experiment uses original DNA microarray images and second uses wavelet transforms to preprocess those images. Experiments confirm that wavelet transformed DNA microarray images enhance recommender system performance by improving the search of cancer cell lines with similar profile to a target cell line.

In addition, we conclude that properly chosen wavelet transformed DNA microarray images, not only uncover richer information for the users' similarity search, but also efficiently compress these images, optimizing computational resources.

To the best of our knowledge, this project is innovative in its use of wavelet transformed DNA microarray images, to profile cell lines in a cancer drug recommender system.

Keywords: recommender system, wavelet transform, cancer genome, cancer disease, cell line, DNA, Google Colaboratory, Python

## RESUMO

A natureza molecular do cancro serve de base para estudos sistemáticos de genomas cancerígenos, fornecendo valiosos *insights* e permitindo o desenvolvimento de tratamentos clínicos. Acima de tudo, estes estudos estão a impulsionar o uso clínico de informação genómica na escolha de tratamentos, de outro modo não expectáveis, em pacientes com diversos tipos de cancro, possibilitando a medicina de precisão.

Com isso em mente, neste projeto combinamos técnicas de processamento de imagem, para aprimoramento de dados, e sistemas de recomendação para propor um *ranking* personalizado de drogas anticancerígenas. O sistema é implementado em *Python* e testado usando uma base de dados que contém registos de sensibilidade a drogas, com mais de 310.000  $IC_{50}$  que, por sua vez, descrevem a resposta de mais de 300 drogas anticancerígenas em 987 linhas celulares cancerígenas.

Após várias tarefas de pré-processamento, são realizadas duas experiências. A primeira experiência usa as imagens originais de *microarrays* de DNA e a segunda usa as mesmas imagens, mas submetidas a uma transformada *wavelet*. As experiências confirmam que as imagens de *microarrays* de DNA submetidas a transformadas *wavelet* melhoram o desempenho do sistema de recomendação, otimizando a pesquisa de linhas celulares cancerígenas com perfil semelhante ao da nova linha celular.

Além disso, concluímos que as imagens de *microarrays* de DNA com transformadas de *wavelet* apropriadas, não apenas fornecem informações mais ricas para a pesquisa de utilizadores similares, mas também comprimem essas imagens com eficiência, otimizando os recursos computacionais.

Tanto quanto é do nosso conhecimento, este projeto é inovador no que diz respeito ao uso de imagens de *microarrays* de DNA submetidas a transformadas *wavelet*, para perfilar linhas celulares num sistema de recomendação personalizado de drogas anticancerígenas.

Palavras-chave: sistema de recomendação, transformada *wavelet*, genoma cancerígeno, cancro, linha celular, ADN, *Google Colaboratory*, *Python*



# CONTENTS

1	INTRODUCTION .....	1
1.1	Statement of the problem .....	1
1.2	Purpose and importance of the study .....	1
1.3	Research question.....	2
1.4	Report outline.....	2
2	THEORETICAL FRAMEWORK.....	5
2.1	Genomics background.....	5
2.1.1	Introduction.....	5
2.1.2	Cell lines .....	5
2.1.3	Gene expression .....	5
2.1.4	DNA microarrays.....	7
2.1.5	Half-maximal inhibitory concentration - $IC_{50}$ .....	8
2.2	Image processing using wavelet transforms .....	8
2.2.1	Introduction.....	8
2.2.2	Wavelet transforms .....	9
2.2.3	Previous work .....	14
2.3	Recommender systems.....	15
2.3.1	Introduction.....	15
2.3.2	Taxonomy .....	17
2.3.3	Techniques overview .....	18
2.3.4	The collaborative filtering technique .....	21
2.3.5	Evaluation metrics .....	24
2.3.6	Practical issues – data sparsity and cold start .....	25
2.4	Personalized recommender systems of cancer drugs .....	26

2.4.1	Domain specificities and application fields .....	26
2.4.2	Previous work .....	27
3	RESEARCH METHODOLOGY .....	30
3.1	The CRISP-DM model.....	30
3.2	The dataset .....	31
3.2.1	Introduction.....	31
3.2.2	Cell lines .....	34
3.2.3	Compounds and IC <sub>50</sub> .....	38
3.2.4	Preprocessing .....	40
3.3	Proposed framework .....	42
3.3.1	Introduction.....	42
3.3.2	Stage 1 – users similarity measurement.....	42
3.3.3	Stage 2 – cancer drug recommendation .....	45
3.4	Used tools.....	46
3.5	Ethical considerations and social responsibility .....	48
4	FINDINGS .....	51
4.1	Introduction .....	51
4.2	First experiment and its results – without wavelet transform .....	51
4.3	Second experiment and its results – with wavelet transform .....	53
4.4	Recommendation example - cancer cell line “HH” .....	54
5	DISCUSSION OF THE RESULTS.....	58
6	CONCLUSIONS .....	65
6.1	Contributions and implications .....	65
6.2	Limitations .....	66
6.3	Recommendations .....	66
6.4	Final considerations .....	67

REFERENCES .....	68
APPENDIXES .....	73
APPENDIX 1. EXPERIMENT 1 / PART 1 – PYTHON CODE.....	74
APPENDIX 2. EXPERIMENT 1 / PART 2 – PYTHON CODE.....	84
APPENDIX 3. EXPERIMENT 2 – PYTHON CODE. ....	132



## LIST OF FIGURES

Figure 2.1 Gene expression .....	6
Figure 2.2 Example of a DNA microarray image.....	7
Figure 2.3 Dose-response curve and $IC_{50}$ .....	8
Figure 2.4 Wavelet translations (location).....	9
Figure 2.5 Wavelet dilations (scale) .....	9
Figure 2.6 Original signal and wavelet transform .....	10
Figure 2.7 Some wavelet types .....	11
Figure 2.8 Wavelet decomposition of a signal for 3 levels .....	12
Figure 2.9 Example of a wavelet decomposition of an image for 3 levels.....	13
Figure 2.10 - Example of a non-personalized summary statistic recommendation.....	18
Figure 2.11 Example of a content-based filtering recommendation.....	19
Figure 2.12 SVD applied to the rating matrix .....	19
Figure 2.13 Predicting ratings using the factors .....	20
Figure 2.14 Example of a hybrid recommender system .....	21
Figure 2.15 User-based collaborative filtering .....	21
Figure 2.16 Item-based collaborative filtering .....	22
Figure 2.17 - Item scoring in item-based collaborative filtering .....	23
Figure 2.18 Rating matrix and corresponding user-item graph.....	25
Figure 3.1 CRISP-DM process diagram.....	30
Figure 3.2 Body parts origins (1018 cancer cell lines).....	33
Figure 3.3 - Body parts origins (927 cancer cell lines).....	34
Figure 3.4 Gene expression data.....	35
Figure 3.5 E-MTAB-3610.raw.1.zip archive file (sample view showing 8 files of the 41 available).....	36
Figure 3.6 Example of a CEL file (storing the gene expression of a cancer cell line) ...	36

Figure 3.7 Example of a DNA microarray image.....	37
Figure 3.8 Sample data description.....	38
Figure 3.9 Cell lines, assays and filenames correspondence file (sample view) .....	38
Figure 3.10 Original GDSC1 dataset (sample view) .....	40
Figure 3.11 Retrieved variables from the GDSC1 dataset (sample view).....	40
Figure 3.12 Drug-response matrix – initial version (sample view) .....	41
Figure 3.13 Drug-response matrix – final version (sample view) .....	41
Figure 3.14 Experiment 1: without wavelet transforms .....	42
Figure 3.15 Experiment 2: with wavelet transform .....	43
Figure 3.16 Wavelet transformed images .....	44
Figure 3.17 Example of a recommendation score .....	45
Figure 3.18 Cancer drug recommendation pipeline.....	46
Figure 3.19 Example of a Google Colab Notebook.....	47
Figure 4.1 Dataset division for evaluation purposes.....	52
Figure 4.2 Daubechies 7 wavelet.....	53
Figure 5.1 Hit-rate box plot .....	59
Figure 5.2 Average reciprocal hit-rate box plot.....	59
Figure 5.3 Similarity box plot.....	60
Figure 5.4 Experiments' final results according to target body part .....	61
Figure 5.5 Final results .....	62
Figure 5.6 Experiments' execution time .....	63

## LIST OF TABLES

Table 3.1 Cell line drug sensitivity data .....	39
Table 4.1 Target cancer cell line "HH" - ground truth ranking .....	55
Table 4.2 Experiment 1 – recommendation list .....	55
Table 4.3 Experiment 2 – recommendation list.....	55

## **List of abbreviations, acronyms, and initials**

CI50 – Concentração Inibitória Média

Colab – Google Colaboratory

CRISP-DM - Cross Industry Standard Process for Data Mining

DNA - Deoxyribonucleic Acid

EDA – Exploratory Data Analysis

GDSC – Genomics of Drug Sensitivity in Cancer

GPU – Graphics Processing Unit

IC<sub>50</sub> - Half-Maximal Inhibitory Concentration

IDLE – Integrated Development and Learning Environment

KPI – Key Performance Indicator

MAE – Mean Absolute Error

ML – Machine Learning

MSE – Mean Squared Error

NCBI - National Center for Biotechnology Information

NR – Not Recommended

RMSE – Root Mean Squared Error

SGD – Stochastic Gradient Descent

SSIM - Structural Similarity

SVD – Singular Value Decomposition

VM – Virtual Machine

WHO – World Health Organization

## **1 INTRODUCTION**

### **1.1 Statement of the problem**

Recommender systems are becoming part of our daily life. Most of their practical applications are web-centric, namely for e-commerce where they engage users by presenting personalized recommendations that best suit their preferences. Nevertheless, recommender systems' potential is much wider.

They can be defined as “software tools and techniques providing suggestions for items to be of use to a user” (F. Ricci, L. Rokach, B. Shapira, 2011). The terms “suggestions” (or recommendations), “items” and “user” can be understood in a broad sense. The underlying logic behind these systems is anchored on machine learning algorithms that are very versatile and can be applied to many fields.

Machine learning has been increasingly used in most diverse domains, either at public sector, such as in fiscal area (Seiça, Trigo, & Belfo, 2019), in education area (Pimenta, Ribeiro, Sá, & Belfo, 2018), in the medical field (Cios & Moore, 2002) or, at private sector, such as in marketing (Cui, Wong, & Lui, 2006), in media and entertainment industry (Sereday & Cui, 2017), in events industry (Loureiro, Lourenço, Costa, & Belfo, 2014) and in many other areas, contributing to create new knowledge and helping organizations to define strategies that allow them increase their performance.

This project explores the application of recommender systems in a specific field of medicine. The problem under analysis is related to cancer disease. Many research laboratories are testing numerous compounds on cancer cell lines, in order to find the most effective drugs. Cancer biology is complex and, as being closely related with the physiognomy of each patient, it means that some drugs are more effective than others in each situation (Iorio et al., 2016). From this perspective, the problem may be solved with a recommender system supported by machine learning, for which the system aims, given a target cancer cell line (i.e., a new patient), to propose a ranking (i.e., a recommendation) of the most effective drugs (i.e., items).

### **1.2 Purpose and importance of the study**

Cancer drug recommender systems allow the research and development of drugs tailored specifically to an individual based on his/her personal genetic information. It is a step towards precision medicine, a new paradigm that benefits all stakeholders. Patients can

have a personalized treatment, increasing the chances of a successful and faster recovery. Healthcare providers (e.g. hospitals) can decrease costs. Pharmaceutical industry can offer new personalized treatments.

In order to contribute to the research community, by stimulating new ideas and drawing attention to the topic, this study led to the homonym paper “Wavelet-based cancer drug recommender system” (Brandão, Belfo, & Silva, 2020) which gathers its main findings.

### 1.3 Research question

Here, a user-based collaborative filtering approach is followed. As a result, users’ profiles are central to the proposed framework. In the specific context, cancer cell lines are profiled through their corresponding gene expression profile, represented by a DNA microarray. As recognized by Serra (2003), there are several microarray systems and methods which differ in several details but produce the same result, an image of spots. On the other hand, images are 2-D spatial signals.

Hence, we intent to assess if the prior pre-processing of DNA microarray images, using wavelet transforms, can improve the recommender system performance. To the best of our knowledge, this is the first work that attempts to do it.

In practice, such preprocessing represents a shift for the data stored in the DNA microarray image from spatial domain (pixels intensities) to wavelet domain (frequencies).

We hypothesize that the representation of the users’ profile in a wavelet domain uncovers distinct and discriminating features that improve the search of similar users (a step of vital importance in a user-based recommender system). This is due to the fact that, as stated by Pittner & Kamarthi (1999), wavelet transforms “allow the extraction of richer problem-specific information”.

### 1.4 Report outline

The document is organized into six chapters.

The current chapter – *Introduction* – contextualizes the general problem under analysis and identifies the specific research question addressed by the project. At the forefront, it states the hypothesis that is expected to be confirmed by the experiments. Furthermore, the real-world impact and contemporaneity of this work are outlined.

The second chapter – *Theoretical Framework* – enlightens the main theoretical concepts applied in the project. It starts by providing the genomic background of the problem, which is fundamental for a better understanding of the dataset that is explored. Afterwards, it exposes the main theory regarding wavelet transforms, focusing on its capacity to perform premium feature extraction for machine learning algorithms. Next, an overview of recommender systems theory is drawn with a special attention in what concerns collaborative filtering. Related work with respect to image processing using wavelet transforms and personalized recommender systems of cancer drugs is also pointed out.

The third chapter – *Research Methodology* – starts by explaining the workflow of the different CRISP-DM phases of the project. Additional details regarding cancer disease worldwide and the used sample are given under the section “introduction”. A full explanation of the database is followed along with the preprocessing steps taken towards the final dataset. This chapter also gives a comprehensive description of the proposed wavelet-based cancer drug recommender system. In fact, details of its two main stages (users similarity measurement and cancer drug recommendation) are explained as well as the identification of the tools used to implement them. The chapter closing remarks go to the ethical and social questions that might arise in the problem domain.

The fourth chapter – *Findings* – and the fifth chapter – *Discussion of the Results* – state the experimental results and provide their critical evaluation, respectively. In the chapter *Findings*, it can also be found a practical example of a cancer drug recommendation for a target cancer cell line.

Finally, the sixth chapter – *Conclusions* – states our main findings and contributions besides the limitations and potential future improvements of this work. As it explains, while pursuing the main research question, not only our initial hypothesis is confirmed but other interesting outcomes emerge. Moreover, it shows how this project contributes to the research towards efficient and effective real-world cancer drug recommender system.





## **2 THEORETICAL FRAMEWORK**

### **2.1 Genomics background**

#### **2.1.1 Introduction**

According to WHO (2020), Genomics can be defined as “the study of genes and their functions, and related techniques”. Contrarily to Genetics (which focus on the study of single genes), Genomics “addresses all genes and their inter relationships in order to identify their combined influence on the growth and development of the organism” (World Health Organization, 2020).

The analysis of “all genes and their inter relationships” allows the identification of distinct patterns which are central for establishing similarities and differences between individual patients.

#### **2.1.2 Cell lines**

In Genomics, cell lines refer to the cells capable of renewing themselves in an artificial culture (i.e. under certain laboratory conditions) indefinitely, which makes them ideal for testing new drugs.

Particularly in what concerns cancer, cell lines are extracted from biopsies (tissues removed from a living body) of patients with different types of tumors (from several body parts depending on the cancer location like lungs or breast).

#### **2.1.3 Gene expression**

There are several ways to characterize cell lines, namely using: gene expression, whole-exome sequencing, copy number variation and DNA methylation. However, as shown by Costello et al. (2014), gene expression provides “the best predictive power”.

Gene expression is the process by which the instructions in our DNA are converted into a functional product, such as a protein. When genes are expressed, the genetic information (base sequence) on DNA is first copied to a molecule of mRNA (transcription). The mRNA molecules then leave the cell nucleus and enter the cytoplasm, where they participate in protein synthesis by specifying the particular amino acids that make up individual proteins (translation).

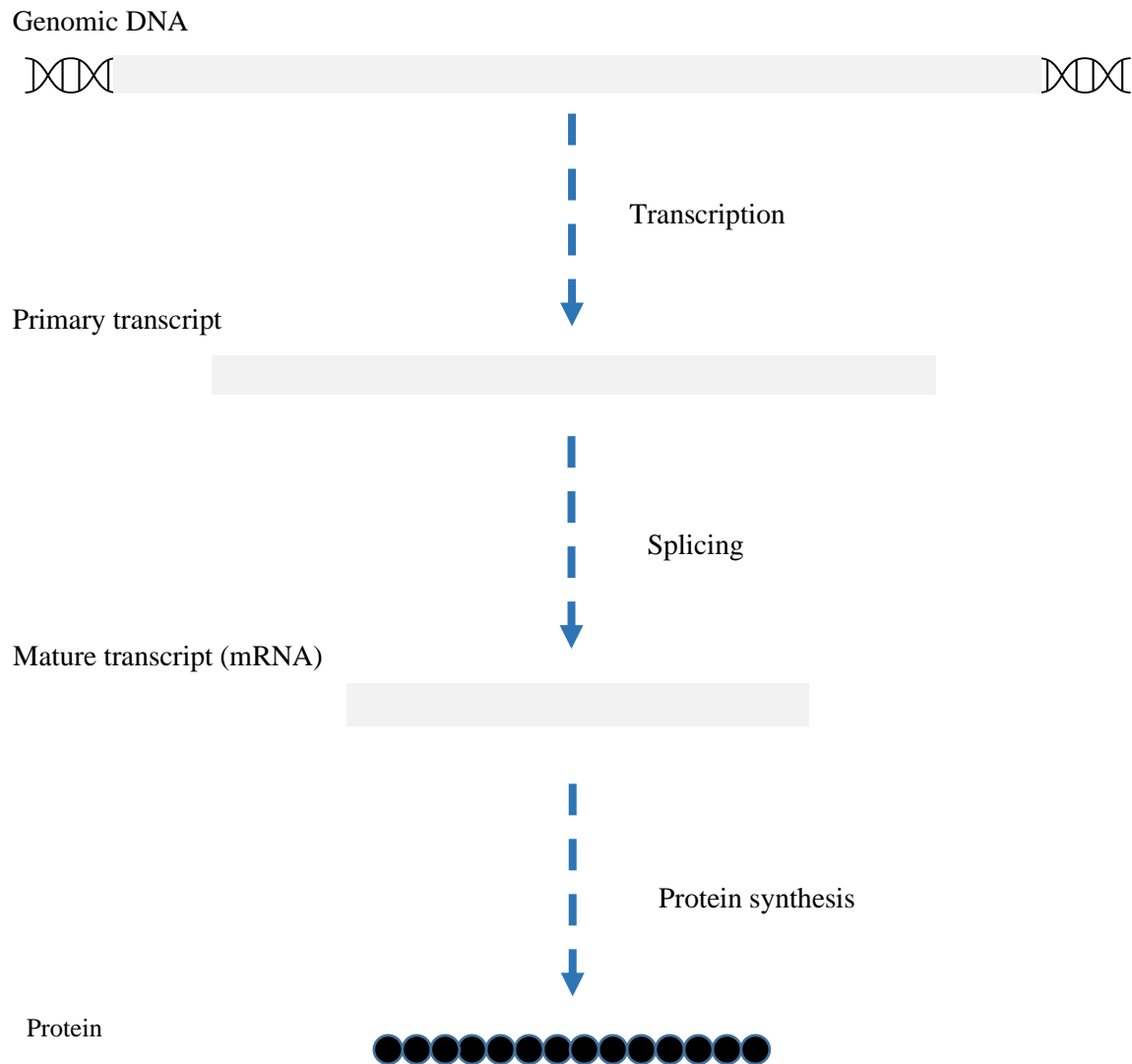


Figure 2.1 Gene expression

Adapted from (NCBI, 2020)

Consequently, one way to measure gene expression is by measuring RNA levels because, as previously mentioned, in order to activate a gene, a cell must first copy the DNA sequence of that gene into a piece of mRNA. Thus, by determining which mRNA transcripts are present in a cell, it is possible to determine which genes are expressed at different stages of development and under different environmental conditions.

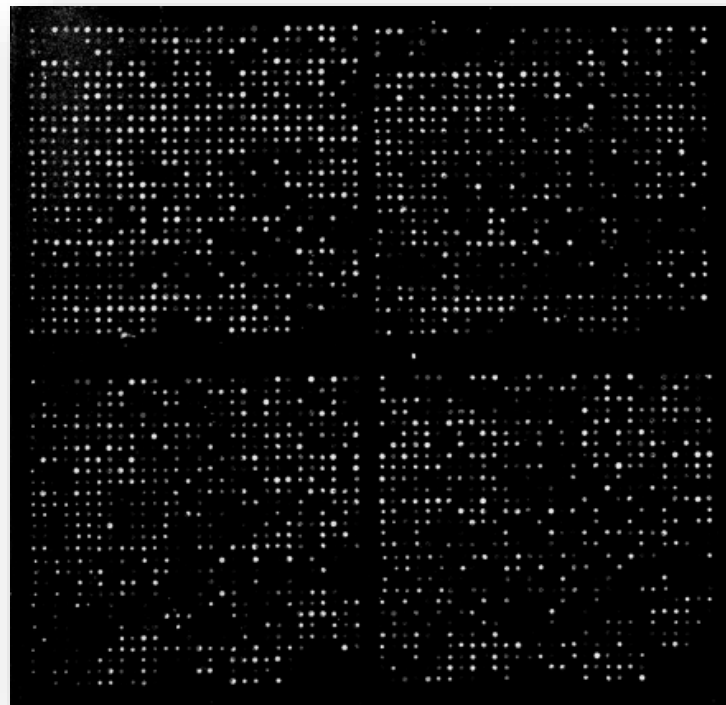
The quantity of mRNA transcript for a single gene directly reflects how much transcription of that gene has occurred. Tracking of that quantity will therefore indicate how vigorously a gene is transcribed or expressed.

Hence, overall, the gene expression of a cell line is a portrait of the genes' activity contained in that cell and the corresponding gene expression analysis is “the determination of the pattern of genes expressed at the level of genetic transcription” in that specific cell (National Center for Biotechnology Information, 2020).

#### **2.1.4 DNA microarrays**

The most used technique to measure mRNA levels is through DNA microarrays (Information Resources Management Association, 2019).

The DNA microarray slide contains up to tens of thousands of microscopic spots. Each individual spot will be used to measure the activity of a specific gene. This happens during the microarray scanning when the fluorescent intensity of all individual gene spots is stored in an image. A spot with high fluorescence intensity represents a hyperactive gene whereas the absence of fluorescence represents a silent one. Therefore, the DNA microarray image provides a “fingerprint” of the cell line.



*Figure 2.2 Example of a DNA microarray image*

*Source (Light et al., 2001)*

### 2.1.5 Half-maximal inhibitory concentration - $IC_{50}$

In pharmacological experiments, it is usual to construct a dose-response curve to represent a drug's effect on a receptor. This curve describes the relationship between increasing the dose (or concentration) of the drug and the change in response that results from this increase in concentration. Typically, it comprises a wide concentration range. (Tulane University - School of Medicine, 2020)

Regarding cancer drugs, the efficacy of a compound is usually assessed by the corresponding  $IC_{50}$  ("I" for inhibition and "C" for concentration), i.e., the half-maximal inhibitory concentration. The  $IC_{50}$  is the concentration of the compound required to inhibit the cell growth at 50%. Hence, the lower the  $IC_{50}$  value is, the more efficient the compound is.

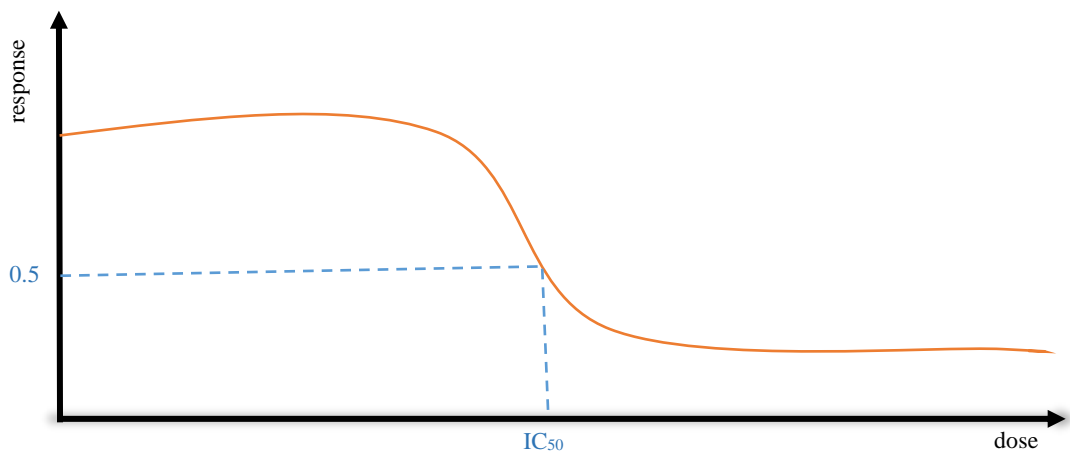


Figure 2.3 Dose-response curve and  $IC_{50}$

## 2.2 Image processing using wavelet transforms

### 2.2.1 Introduction

In the process of microarray scanning, an image of the genes' activity induced from the fluorescence dye is captured by the scanner. However, "due to the weak fluorescence response, complex biochemical reaction, imperfections in glass slide and photoelectric sensor conversion distortion, etc., the signal of fluorescence probe is inevitably degraded, which leads to serious noise interference in the microarray image" (Gan et al., 2019).

Wavelet transform is a signal processing technique (also applied to images since they are 2-D spatial signals) that, simultaneously, allows to filter noise and extract more informative features that allow better discriminability of the original signal.

### 2.2.2 Wavelet transforms

The purpose of the wavelet transform is to “transform the signal under investigation into another representation which presents the signal information in a more useful form” (Addison, 2017).

A wavelet is a little wavelike function. During the wavelet transform, a convolution of the signal with a wavelet function happens. This convolution is computed at various locations of the signal (wavelet translations) and for various scales (wavelet dilations). The signal regions where the wavelet overlaps the signal result in large transform values, called wavelet coefficients.

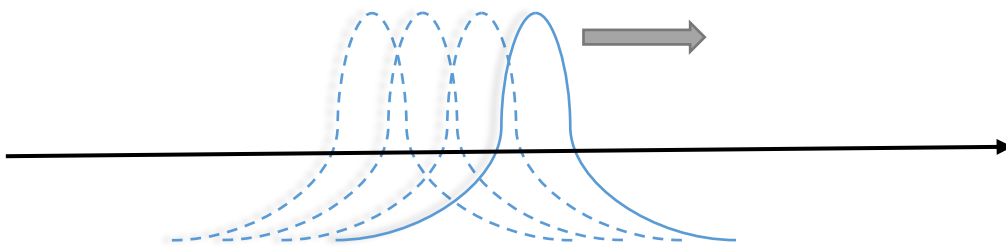


Figure 2.4 Wavelet translations (location)

Adapted from (Addison, 2017)

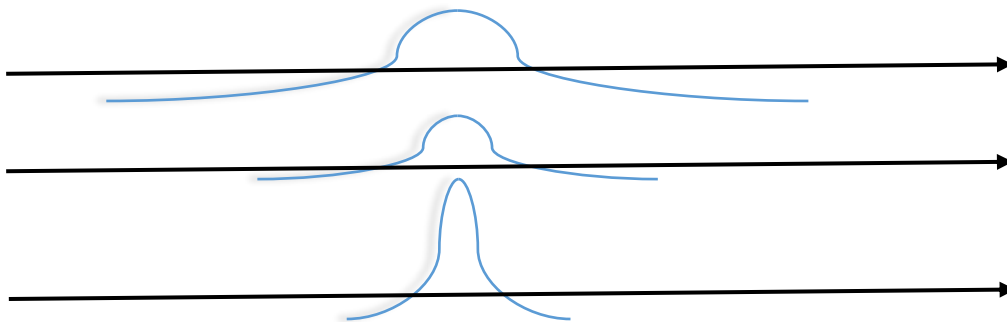


Figure 2.5 Wavelet dilations (scale)

Adapted from (Addison, 2017)

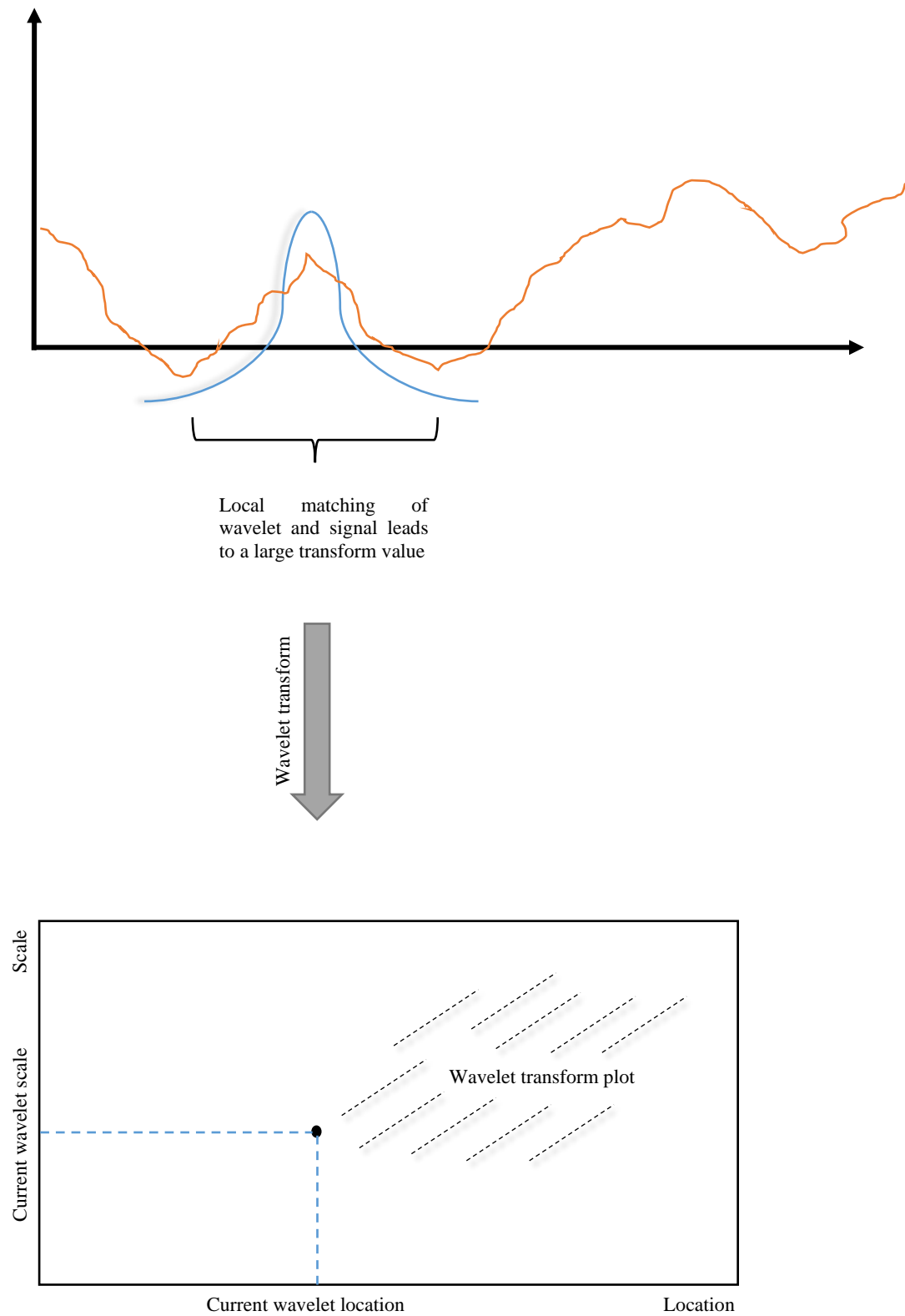


Figure 2.6 Original signal and wavelet transform

Adapted from (Addison, 2017)

Therefore, several clear structures relating to a specific scale in the wave are detected by shifting the wavelet along the signal (Addison, 2017). For this reason, wavelet transform has been called a ‘mathematical microscope’.

There are several types of wavelets, such as those from Haar, Daubechies, coiflet, and symlet, and it is important to choose the one that best suits our signal and the scope of the analysis.



*Figure 2.7 Some wavelet types*

*Source (Addison, 2017)*

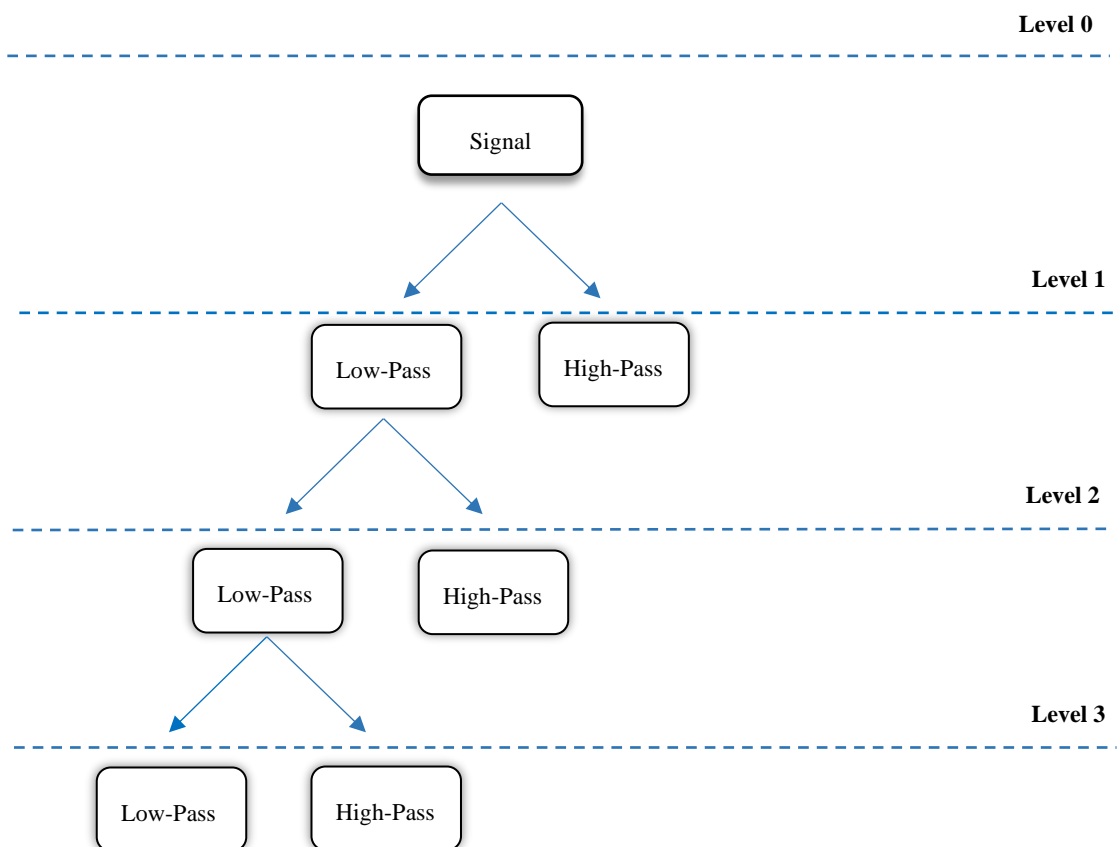
Moreover, wavelet transforms come in two distributions: continuous and discrete. The major difference relies in the way how they discretize the scale parameter. The continuous wavelet transform uses exponential scales with a base smaller than 2 (e.g.  $2^{1/5}$ ) while the discrete wavelet transform uses exponential scales with the base equal to 2 (i.e., the scales are powers of 2). Hence, continue wavelet transform discretizes scale more finely than the discrete wavelet transform.

However, for image processing, the discrete wavelet transform is the type of distribution usually used, allowing a sparse representation of the signal. Here, coefficients whose value is close to zero may be ignored, remaining only those that have captured important features.

The wavelet transform can also comprise several levels of decomposition. In the first level, the signal is decomposed in low and high frequencies regions. The convolution of the wavelet with the low frequency regions results in the so-called approximation coefficients. On the other hand, the convolution of the wavelet with the high frequency

regions results in the so-called detail coefficients. In the next level, the approximation coefficients (of the previous level) are again divided into low and high frequency regions. This goes on until it is reached the level of detail that it is needed or until there is no more low and high frequency regions.

The wavelet packet transform is similar to the discrete wavelet transform, however, at each decomposition level, it decomposes not only the approximation coefficients but also the detail coefficients, yielding a higher frequency resolution even in higher frequencies.



*Figure 2.8 Wavelet decomposition of a signal for 3 levels*

It is possible to apply wavelet transforms to any signal such as time and spatial signals. Spatial signals comprise, for example, 2-D images.



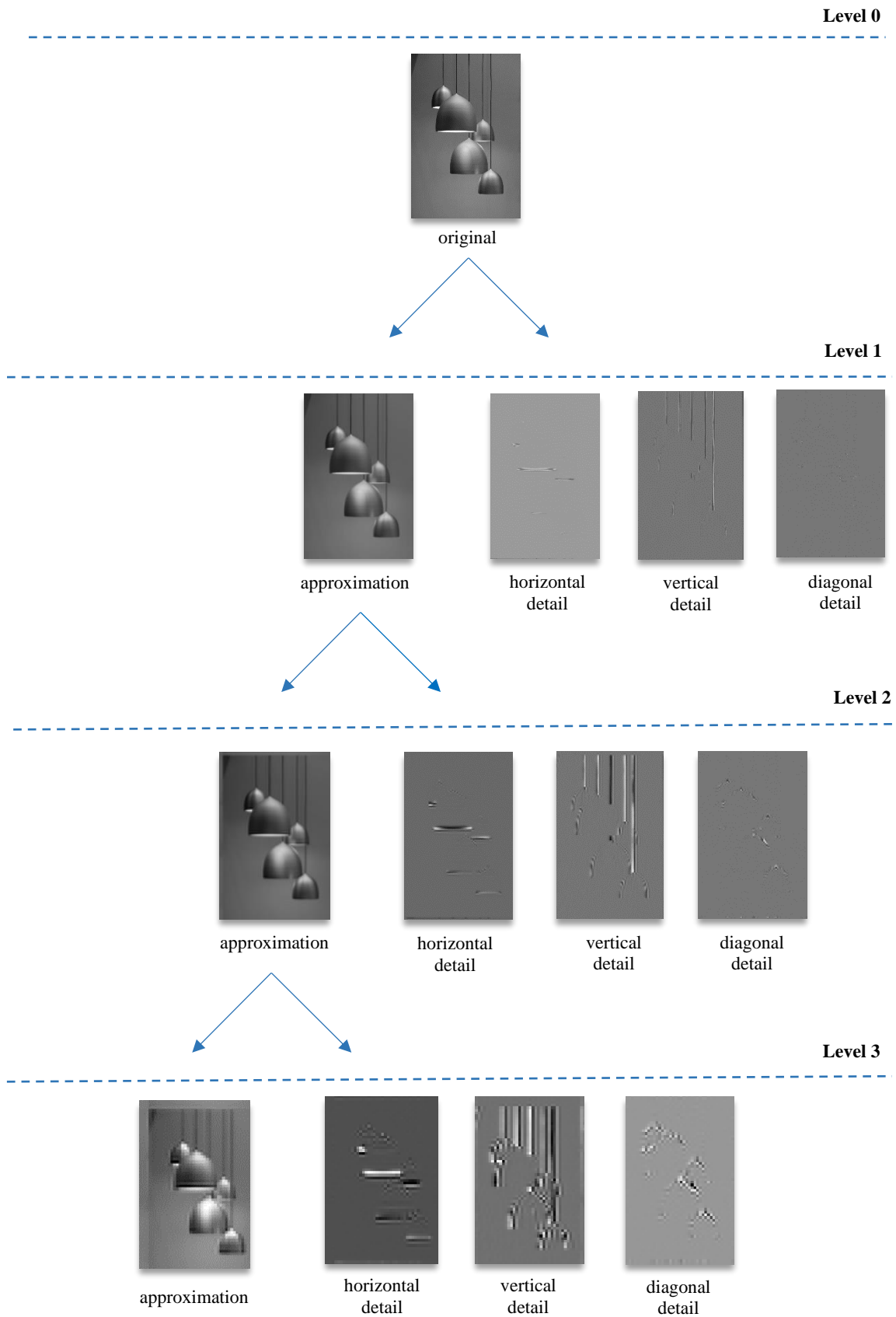


Figure 2.9 Example of a wavelet decomposition of an image for 3 levels

### **2.2.3 Previous work**

In what concerns machine learning classification tasks using images as inputs, the use of wavelets as a preprocessing technique is already in use.

For example, Wan & Zhou (2010) made use of the Haar wavelet to extract features of ultrasound liver images for a support vector machine classifier. The global scope was to find “an effective method of feature extraction for revealing texture details and the succeeding classifications”.

For each sub-image of the transform, they quantified mean and energy values and stored them in a feature vector later on used to feed the classifier. They came to the conclusion that wavelet packet transform allowed better feature vectors, regarding wavelet transform, since they had shown “excellence in differentiating normal and cirrhosis classes”.

More recently, Rasheed, Younis, & Bilal (2020), working in a classification task using deep neural networks, applied the wavelet transform as a preprocessing technique in order to “help the network by enhancing the features in the images”. They reasoned that wavelet transforms can unravel important features “within the image before feeding it to the classifying network”.

Therefore, instead of the original images, they used two wavelet transformed images to train the network. Those images were obtained by convoluting the wavelet along the vertical and horizontal directions of the original image.

They found that the wavelet transform of X-Ray scans improves considerably the performance of the classification network.

We can also find research regarding the application of wavelet transforms to DNA microarray images. For instance, Li, Liao, & Kwok (2006) proposed a gene selection method using the discrete wavelet transform on microarray data for cancer classification. As they expressed, “microarray data typically have thousands of genes, and thus feature extraction is a critical problem for accurate cancer classification”.

They began using the wavelet transform to decompose the microarray data. Afterwards, they applied “the maximum modulus method to select some high-frequency coefficients”. Those coefficients and the approximation ones were then “combined together to form a new gene subset with a much lower dimensionality than the original one”.

The experiments showed that this wavelet-based feature extraction method for microarray data was able to “outperform the other methods in terms of classification accuracy”.

Also, for a cancer classification task, Nanni & Lumini (2011) adopted wavelet features extracted from DNA microarrays. According to them, the high dimensional data present in the microarrays contains irrelevant information that jeopardizes the classifier accuracy. Therefore, “a feature reduction should be performed before the classification step”.

In their proposed system, the DNA microarrays were submitted to a wavelet decomposition and the resulting detail coefficients selected through the sequential forward floating selection method. The final subset of detail coefficients was then used as an input to the classifier.

Since “different sub-bands of different wavelet families bring different information”, they studied the impact of eighteen different wavelets at different decomposition levels.

Overall, the literature shows that wavelet decompositions can be used to overcome the curse of the dimensionality problem by highlighting relevant information for the machine learning algorithm. Nevertheless, it is important to keep in mind that, since there is a wide choice of wavelet families and distinct decomposition levels can be applied, each problem will have its one optimal wavelet transform, requiring knowledge and art to find it.

## **2.3 Recommender systems**

### **2.3.1 Introduction**

The global scope of recommender systems comprises the identification of the need and preferences of users, filtering the huge collection of data accordingly and displaying the best fitted option by using some well-defined mechanism (F. Ricci, L. Rokach, B. Shapira, 2011). Thus, they allow information filtering (in opposition to information retrieval like the one performed by search engines), providing personalized information that is relevant to the user.

It is important to notice that, even before the appearance of recommender systems, humans have been making use of personalization as a way to tailor services and products to specific individuals. However, such personalization is rooted in personal intuition and experience whereas the personalization made by recommender systems is automated and supported by data.

There are several important historic milestones that one could point out regarding recommender systems. Their early days go back to 1994 when GroupLens (a research lab in the Department of Computer Science and Engineering at the University of Minnesota) built a system able to produce personal predictions regarding news articles. At that time, the internet use was rapidly spreading, and recommender systems relying on collaborative filtering were conceived to help users handle information overload. Thus, the GroupLens recommendation system, using the reading opinions (ratings) of like-minded users about Usenet news articles, produced personalized recommendations that were displayed in the article header. (Resnick, Bergstrom, & Riedl, 1994)

Quickly, business applications began to be explored, and in 2009 Netflix (a web-based commercial company) launches the first recommender system challenge with the grand prize of US\$1,000,000.

In fact, recommendation engines expanded rapidly among online retailers and online content providers. Even in nowadays, recommender systems are still very web-centric.

We can identify at least two reasons for that. On one hand, personalized recommendations engage people and engaged users are loyal (or profitable) users – exactly the type of customers that companies seek.

On the other hand, we must not forget that, while humans rely on intuition to make personalization, recommender systems rely on data and it is fairly simple for a web-based company to collect a considerable amount of data produced by its online users. That data is raw information and, therefore, the data trail that an online user leaves behind is processed by the recommender engine to learn preferences and produce meaningful recommendations.

Furthermore, in an online context it is possible to collect, not only explicit data (for example, when a user rates an item), but also implicit data. This is the type of data that a user produces non-intentionally, without being directly asked to, but that still has very rich information about him/her. Examples of implicit data include click data (e.g. page views), purchase data, consumption data (e.g. time spent in a page), and so on.

One might be led to think that explicit data is the best data to know a user. That it is not always the case, explicit data has some pitfalls. It requires an extra effort from the user (the direct action of rating) so, usually, this data is very sparse (not every user is willing

to rate). Also, when using it to produce recommendations, it might be necessary to overcome some rating subjectiveness.

### **2.3.2 Taxonomy**

There are several dimensions that can be used to characterize and classify recommender systems.

The domain regards the recommendation item itself which can range from products to services. As previously mentioned, (see Section 1.1), the definition of item within a recommender system background can be very broad.

The purpose identifies the overall goal underneath the recommendation; sales, information, education, ...

Another important characteristic is the personalization level. In fact, recommender systems can have different levels of personalization: from universal (non-personalized recommender systems in which everyone receives the same recommendations) to tailor-made (personalized recommender systems that match the user personal preferences). In between, we can have different personalization degrees. For example, demographic personalization.

Another differentiating aspect is the approach followed to solve the recommendation problem. A possible solution is, given a set of unknown items to the user, predict the ratings each item will have and then present the most N (predicted) rated items as recommendations for that user. This is the “prediction version” of the problem (Aggarwal, 2016).

However, it is not mandatory to predict ratings in order to make recommendations. One can simply recommend the top-N most likely relevant items to the user. This is the “ranking version” of the problem and “in many cases it is easier and more natural to design methods for solving the ranking version of the problem directly” (Aggarwal, 2016). In fact, this is the version commonly adopted in real world problems.

Finally, recommender systems can be classified regarding the model technique. Section 2.4.3. provides an overview of the main techniques available while Section 2.4.4 takes a deeper look to the one employed in this project – collaborative-filtering.

### 2.3.3 Techniques overview

The basic task of a recommender system is to suggest items to users. In order to perform it, several techniques are available, but most of them fall under one of the following categories: non-personalized summary statistics, content-based filtering, collaborative filtering (described in the next Section), matrix factorization methods and hybrid approaches.

Recommender systems relying on non-personalized summary statistics use aggregated indicators in order to propose items. “Best-seller”, “most-popular”, ... are examples of such indicators regarding e-commerce recommender engines. This is one of the most simple and easy to implement techniques, although its inexistent level of personalization.



IMDb Charts  
Most Popular Movies  
As determined by IMDb Users

Showing 100 Titles      Sort by: Release Date

Rank & Title	IMDb Rating	Your Rating
 The Matrix 4 (2022) 61 ( ⭐ 66)		☆ +
 Zack Snyder's Justice League (2021) 100 ( ⭐ 48)		☆ +
 The Batman (2021) 63 ( ⭐ 1)		☆ +

Figure 2.10 - Example of a non-personalized summary statistic recommendation

Adapted from (IMDb, 2020)

Content-based filtering recommender systems, on the other hand, belong to the group of recommender systems that enable personalization. To model the user profile, the engine exploits the items towards which the user’s preferences are already known. These known preferences, along with the content (or characteristics/properties) of the items, are then used to infer the preferences of the user towards a new item. In other words, the algorithm recommends items that are similar to the ones that received favorable preference of the user in the past.

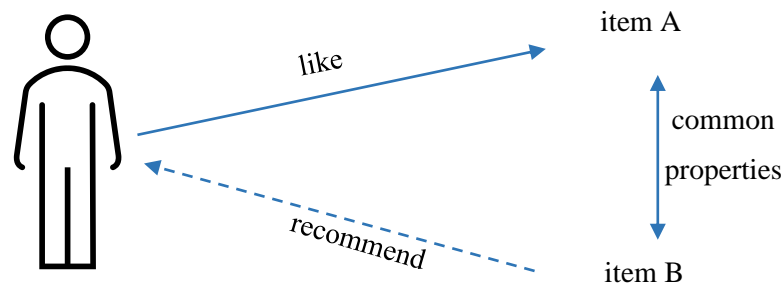


Figure 2.11 Example of a content-based filtering recommendation

Matrix factorization, another important recommender system technique, finds items to recommend using extracted factors from the so-called rating matrix. These factors, called latent factors, express trends in the data that explain the user's behavior and, although they make sense data-wise, it might be hard to interpret their meaning.

The first step is to factorize the rating matrix into smaller matrices. One of the most commonly used methods for that is SVD that will decompose the rating matrix into three matrices:

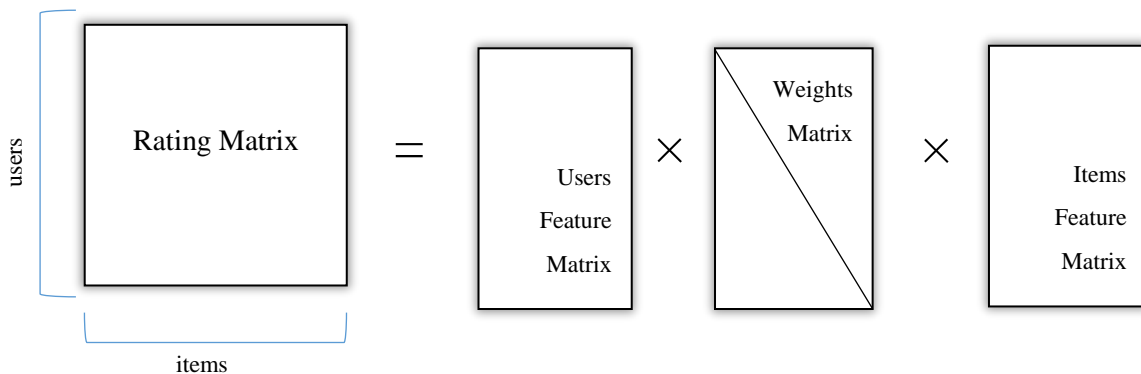


Figure 2.12 SVD applied to the rating matrix

Adapted from (Falk, 2019)

The weights matrix is a diagonal matrix whose elements are sorted from the largest to the smallest. The values of these elements, called singular values, indicate how much information a feature (both a column in the users feature matrix and a row in the items feature matrix) produces for the dataset. Therefore, we can select a  $k$  number of features

(for example, the ones that retain 90% of the information) and set the rest of the diagonal to zero. As a practical consequence, we will be reducing the number of columns in the users feature matrix and the number of rows in the items feature matrix, keeping only the most meaningful dimensions. These reduced matrices are called the rank-k approximation of the rating matrix. According to linear algebra, this is the best possible rank-k approximation.

To predict a rating for a given user, we multiply his/her reduced factors.

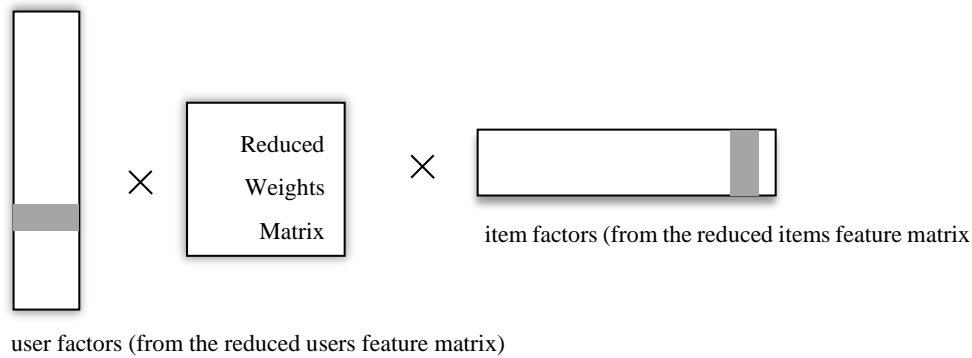


Figure 2.13 Predicting ratings using the factors

Adapted from (Falk, 2019)

Since rating matrices are usually sparse, their unknown values must be filled prior to the SVD factorization, and for that several methods are available like imputation (for example, replace the unknown values by a mean value).

Alternatively, there are approaches that, also under the matrix factorization mindset, use SGD to directly search for the best rank-k approximation without the need to deal with the missing values. In fact, SGD allows efficient approximation of the SVD from known data.

Obviously, each recommender system algorithm has its own strengths and weaknesses. Hybrid approaches, combining different algorithms, can be used as a way to overcome an algorithm's weaknesses by another algorithm's strengths. Once more there are several ways to perform such an ensemble. For example, in a weighted hybrid recommender, the predicted rating is computed from the results of all of the available recommendation techniques present in the system.



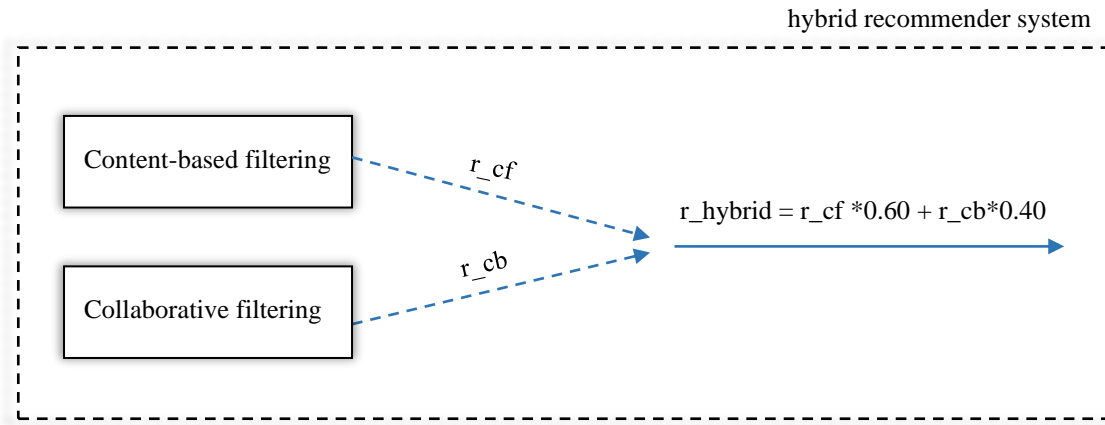


Figure 2.14 Example of a hybrid recommender system

Adapted from (Falk, 2019)

### 2.3.4 The collaborative filtering technique

Collaborative filtering models use the “collaborative power” (Aggarwal, 2016) of the ratings provided by multiple users. The assumption on which they are based is that similar users share similar behaviors and similar items receive similar ratings. Therefore, the items that were relevant to existing users and the items that are similar to the ones that were relevant to the target user will most likely be good recommendations.

This technique relies on the concept of neighborhood to infer the relevance of an item to the target user. Two types of neighborhood can be defined – user-based and item-based.

In the user-based collaborative filtering, the ratings provided by a neighborhood of similar users to a target user are exploit to make recommendations.

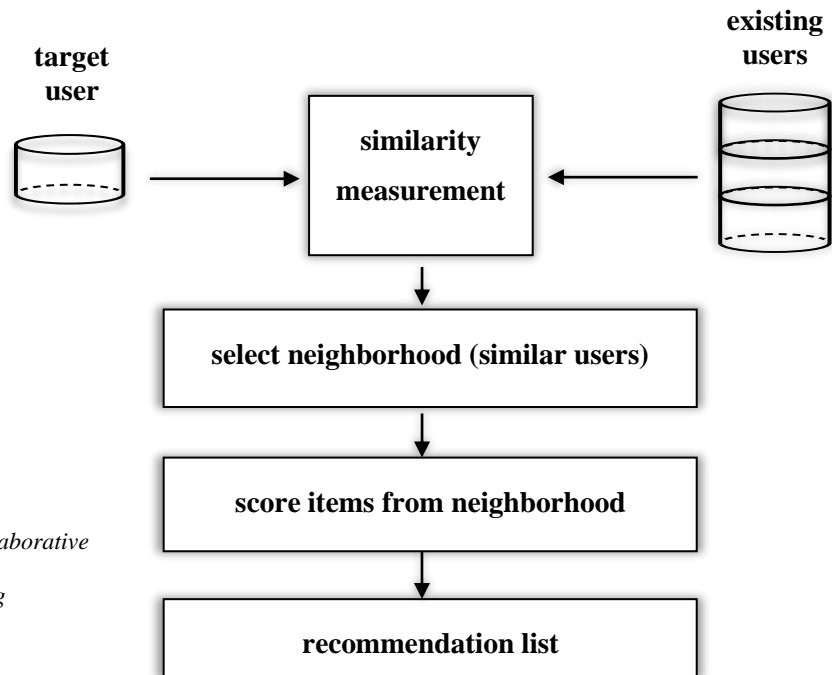


Figure 2.15 User-based collaborative filtering pipeline for creating a recommendation list

According to Figure 2.15, the first step is “similarity measurement” which requires calculating the target user’s similarity regarding all the existing users. Such similarity is usually computed through the users’ rating vectors. Several similarity metrics can be applied, such as Pearson correlation (a standard correlation metric), and the background of the problem should guide to the final choice (for example, some metrics are more well suited for certain types of data than others).

When the similarity measurement is completed, is time to “select a neighborhood”. The scope is to remove the noise introduced by users that are not alike the target user. Once more, there are several ways to determine a neighborhood. For example, limit the neighborhood to the top-k neighbors or to a similarity threshold.

The “recommendation list” is then created using the items of the neighborhood. Firstly, each item receives a “score” - the rating given by the neighbor weighted by the similarity between the neighbor and the target user. It is important to notice that ratings may need to be previously normalized. For example, when dealing with a rating scale, human users might have different perceptions of it and, therefore, for instance, a rating “5” may mean different things to different users. Normalization compensates this user bias. Secondly, the items are sorted by “scores” and the top-N items are recommended to the target user. Also, it might be necessary to filter some items (for example, items that for some reason are not suitable for the target user).

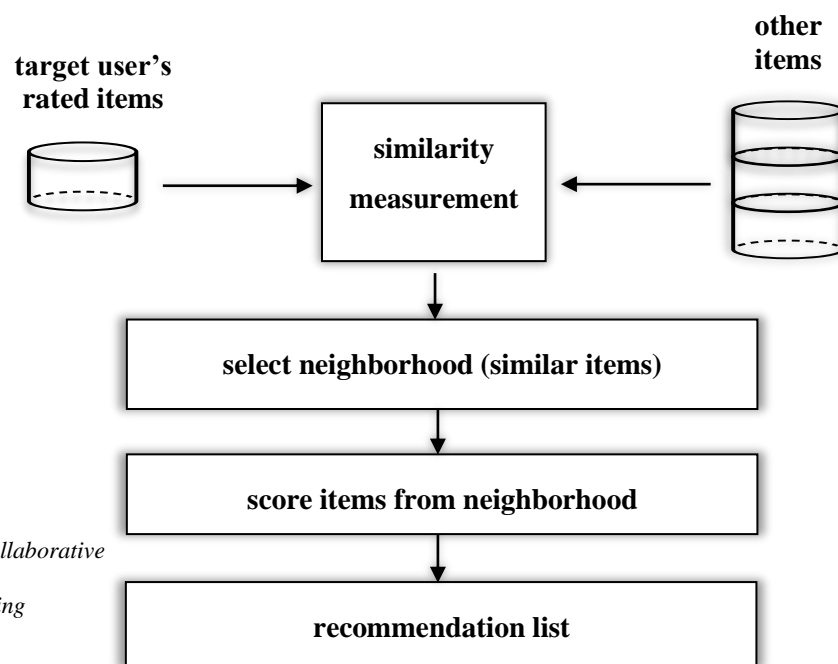


Figure 2.16 Item-based collaborative filtering pipeline for creating a recommendation list

The second type of neighborhood is item-based, and it also relies on similarity, namely between items according to their vector ratings (and not according to items' properties/characteristics as in content-based filtering). Hence, the first step is to compute the similarity between each of the items rated by the target user and the other items of the database. The “score” of each neighbor item is the rating given by the target user weighted by the similarity between the neighbor item and the target user's rated item, as illustrated in figure 2.17.

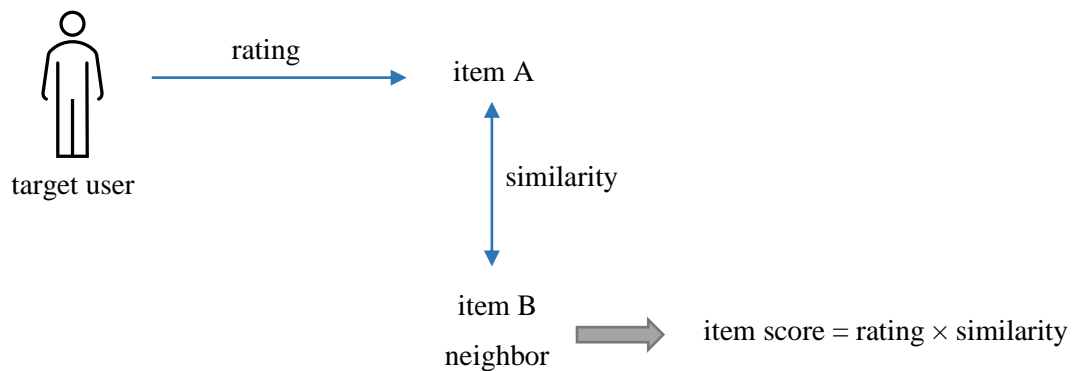


Figure 2.17 - Item scoring in item-based collaborative filtering

In short, in user-based collaborative filtering, neighborhoods are defined by similarities among users (rating matrix rows), whereas in item-based collaborative filtering, neighborhoods are defined by similarities among items (rating matrix columns).

Usually, item-based collaborative filtering is a technique with better performance and stability than user-based. However, under certain circumstances it is likely to fail to outperform user-based. For example, in an application where there is a relatively small number of users, and many more items since the benefits of item-based collaborative filtering depend on having more users than items.

It is also important to notice item-based low serendipity. In order to be effective, item-item relationships need to be stable and, as consequence, it is very difficult for the algorithm to discover highly different items to recommend and, hence, more of the items recommended are expected. (On the other hand, user-based collaborative filtering by default will elevate items that a close neighbor rates highly.) These “conservative” recommendations can be good for shopping or consumption tasks, but they might be frustrating, for example, for browsing/entertainment in which users might enjoy being surprised with bold recommendations.

### **2.3.5 Evaluation metrics**

Like in any other machine learning algorithm, historical data can be used to evaluate a recommender system performance. In this regard, three types of basic metrics can be distinguished: prediction accuracy metrics, decision support metrics and ranking metrics.

Prediction accuracy metrics evaluate how good the recommender system is in what concerns predicting users' ratings. Therefore, they measure the error between the prediction and the actual rating. Such metrics include MAE, MSE and RMSE. RMSE, in particular, gives significantly greater weight to larger errors (unlike MAE) which often reflects the fact that big errors are much worse for user experience. Also, RMSE is easier to interpret than MSE because the scale of errors is matched more closely to the scale of ratings.

On the other hand, decision support metrics measure how well a recommender system helps users make good decisions, i.e., choosing relevant items. Under such mindset, for example, a prediction of 4 stars versus 2.5 stars (true rating) is worse than a prediction of 2.5 stars versus 1 star (although the MAE – 1.5 – is the same on both predictions). Precision – the percentage of selected items that are actually relevant – and recall – the percentage of relevant items that are selected – are perhaps the most widely used decision support metrics. It is also common to restrict such measurements to the N top items. For example, top-N hit rate (or precision at N) is the fraction of relevant items that are in the top-N recommendation list in relation to the N recommended items (the recommended items that are actually relevant items are called hits).

Finally, ranking metrics assess how good the recommender system is at suggesting relevant drugs on top positions. In other words, these metrics trace the position within the recommendation list in which the item appears. As stated by Aggarwal (2016), “the disadvantage of the hit-rate is that it gives equal importance to a hit, irrespective of its position in the recommended list”. Average reciprocal hit-rate is like top-N hit-rate, but, contrarily to it, it takes into account for where in the top-N recommendation list the hits appear. Its scope is to reward recommended items that match top relevant items.

Other types of metrics, that relate more closely to business goals and/or user experience, can be equally important. For example, coverage measures the percentage of items for which a recommender system can make a prediction (in a commercial domain, for instance, it is paramount to ensure that the algorithm will recommend everything in the

sales catalogue). Another example is serendipity which attempts to measure unexpected recommendations that lead to unexpected beneficial results.

All the previous metrics are measured in a retrospective mindset, i.e., dealing with items already rated. However, especially in what concerns recommender systems designed for the web, it is very common to also perform online evaluations (live experiments), under a prospective mindset (looking at how recommendations are actually received). One of the most popular mechanisms to perform such field experiments are the so-called A/B tests which aim to see if a system change makes a positive improvement in user activity. For example, if we are trying to decide between a user-based collaborative filtering (version A) or an item-based algorithm (version B), we might give half of the users each of those and follow them over a period of time, tracking a selected number of KPI (e.g. conversion rate, i.e., recommendations that result in sales).

### 2.3.6 Practical issues – data sparsity and cold start

The two main problems of the recommender systems are: data sparsity and cold start.

Regarding the first problem, rating matrices might be sparse and, consequently, provide little information, making the recommendation task difficult especially for collaborative filtering algorithms (which, as shown in Section 2.4.4, rely on rating matrices to compute similarities between users and items). One way to address such problem is through the use of graph models that provide a “structural representation of the relationships among various users and/or items” (Aggarwal, 2016) and allow the identification of neighborhoods using random-walk or shortest-path methods.

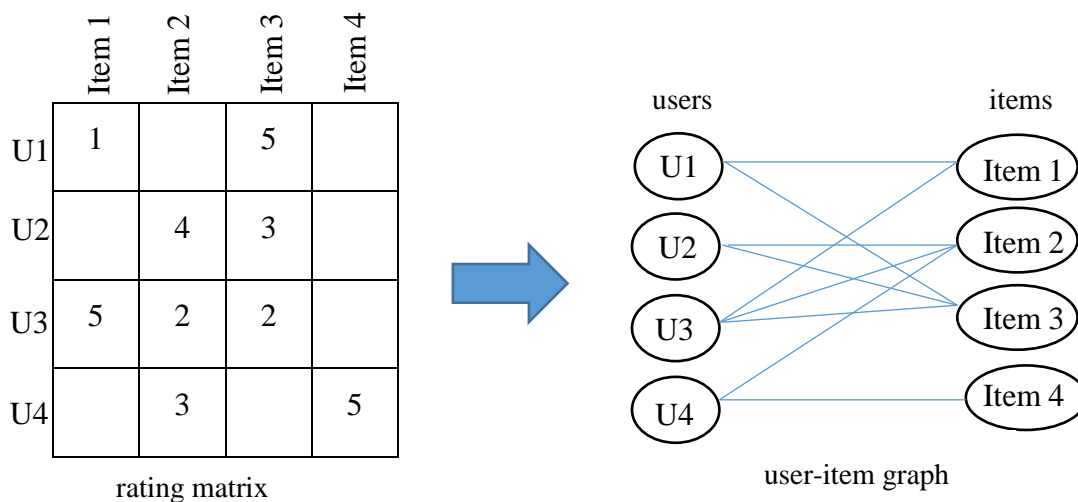


Figure 2.18 Rating matrix and corresponding user-item graph

Adapted from (Aggarwal, 2016)

Another potential issue is the cold start problem – what to recommend to a new user (that has no profile of preferences) and how to recommend new items (that have not been rated yet)?

For a new user, some solutions might include providing useful default personalization options (e.g. recommend popular items) or make use of their implicit data (e.g. recommend items similar to the item they are looking). For a new item, content-based approaches (including similarity to other items) might be useful to overcome the problem.

## **2.4 Personalized recommender systems of cancer drugs**

### **2.4.1 Domain specificities and application fields**

Although the classic expression “the human body”, large response variability among individuals due to genetic differences is observed. This fact has been increasing the rise of precision medicine (also called personalized medicine). Precision medicine aims to develop tailored diagnostic, treatment, and prevention based on a patient’s genes and genetic modifications of these genes.

Under this mindset, personalized recommender systems of cancer drugs target the use of a patient’s genomic profile to generate tailored cancer drugs recommendations, i.e., one-to-one recommendations (instead of one-to-many).

The underlying concepts behind such systems can be found in the general recommender system theory but, as expected, with the necessary adaptations imposed by their specific domain and problem background.

For example, the typical rating matrix (inspired by online backgrounds and storing users’ preferences regarding items), in a cancer drugs recommender system context gives place to a drug-response matrix. The drug-response matrix stores, as the name implies, the drug responses (or reactions) of each cancer cell line (rows) with respect to each drug (columns).

Moreover, personalized recommender systems of cancer drugs with a collaborative filtering approach (like the one we propose here), may not rely on drug response vectors to compute similarities between users. Instead, they may use other type of rich information about the users such as gene expression. Among other benefits (for example, enhancement of the similarity measurement), this option allows to overcome both data sparsity and cold start problems on the users’ side.

Also, in what regards evaluation metrics, the focus relies on the recommender system ability to suggest the most efficient drugs to the target patient (or cancer cell line). Therefore, certain metrics may not make sense in such context. For example, metrics reflecting business goals such as coverage. On the other hand, the skill to suggest the most likely relevant drugs in the right rank is highly important to ensure that patients receive the top treatments as soon as possible.

Finally, it is important to notice that personalized recommender systems of cancer drugs can be useful not only in a clinical scenario (assisting doctors on the search of personalized treatments for each individual patient) but also in a clinical laboratorial set up, with a drug development scope, aiding researchers to identify new active pharmaceutical ingredients. In fact, these recommendation algorithms may well represent an outstanding opportunity for the pharma industry to offer precision medicine.

#### **2.4.2 Previous work**

Most of the research on cancer drug recommender systems has focused on the “prediction version” of the problem, i.e., predicting the exact sensitivity values for the potential drugs. To achieve this goal, a common technique applied is matrix factorization. Matrix factorization solves the recommendation problem by finding latent features that determine the relationship between users and items.

For example, Suphavitai, Bertrand, & Nagarajan (2018) used this technique to project both drugs and cell lines into a latent space, named as “pharmacogenomic space“, such that “the dot product between a cell line vector and a drug vector provides the cell line specific drug response”.

With that aim, they factorized the drug response matrix into three matrices - biases, cell lines and drugs – and then treated the problem as a minimization problem, where they try to find the missing values in these matrices that best minimize the errors in the known drug response.

With a similar approach, L. Wang, Li, Zhang, & Gao (2017) also focused on the drug response prediction. Exploring similarities of drugs and cell lines simultaneously, they proposed a similarity-regularized matrix factorization framework.

The major difference, regarding the previous research work, is that the latent space is built using a drug similarity matrix and a cell line similarity matrix.

Nevertheless, a few researchers have also focused on the “ranking version” of the problem.

That is the case of He, Folkman, & Borgwardt (2018) whose framework gains were “maximized when the most effective  $k$  drugs are the top  $k$  recommended drugs”, even though, without any drug response being predicted.

In order to achieve that goal, the drug recommendation was framed into a problem of learning a weight matrix  $W$  such that, given a new molecular profile  $x$ , the predicted ranking vector (containing the predicted ranking scores for each of the  $m$  distinct drugs) would be  $f=xW$ .

Then, using a loss function to evaluate how well the order of the top  $k$  recommendations in  $f$  matched with the order of the most effective drugs in the drug response matrix, learning  $W$  was settled as a minimization problem.

To sum up, despite the fact that most of the research effort is made under a “drug response prediction” mindset, the “ranking version” approach (which does not rely on predicting drug responses to make recommendations) is equally interesting. In fact, in a real-life clinical scenario, it is more reasonable that a patient receives a recommendation of a few, most effective, drugs rather than predicting the exact response to all drugs.





### 3 RESEARCH METHODOLOGY

#### 3.1 The CRISP-DM model

The CRISP-DM model is a data mining methodology and process model that provides a standard blueprint for managing a data mining project. Launched in 1999, it is still used worldwide by major players due to its timeless advantages such as effective and efficient project planning and management.

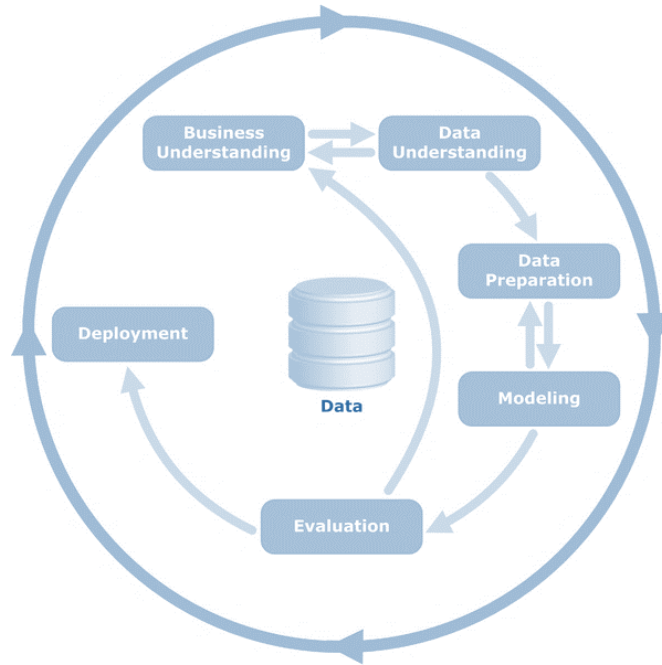


Figure 3.1 CRISP-DM process diagram

Adapted from (Jensen, 2012)

Despite initially created for data mining, this methodology provides powerful guidelines for even nowadays the most advanced data science activities and, therefore, it was the approach followed by this project. In fact, throughout this project report, the six CRISP-DM main phases can be easily identified:

1. *Business understanding*

Focused on identifying the project objectives, setting a clear research question, and gathering the necessary Genomics background for a full understanding of the necessary domain knowledge. Hence, we started with a generic problem – cancer disease – and gradually narrowed it into a specific research question. Later on, the research question was translated into a data science problem and a framework to

answer it using mathematical instruments (wavelet transforms) and computer science algorithms (recommender systems) was planned.

2. *Data understanding*

The data was collected from the GDSC database (Yang et al., 2013) (Genomics of Drug Sensitivity in Cancer, 2020) and imported into a Python Notebook where several EDA tasks were conducted. This allowed us to get familiar with the data and discover first insights. For example, the detection of several cancer cell lines without DNA microarrays available.

3. *Data preparation*

In order to get the final dataset, several preprocessing tasks were necessary to transform the initial raw data. These tasks ranged from missing values imputation to features engineering (design of wavelet transformed images).

4. *Modeling*

Recommender systems comprise several techniques, therefore, guided by the data available, a user-based collaborative filtering approach was selected and applied. To assess the performance of the models (with and without wavelet transformed images) two evaluation metrics were chosen: top-N hit rate and average reciprocal hit-rate.

5. *Evaluation*

The results of the first round of experiments allowed us to fine-tune the models, motivating adjustments in the wavelet transformed images design.

6. *Deployment*

This project aims to be a preliminary study, hence, no deployment of the model into an operating system was planned.

## 3.2 The dataset

### 3.2.1 Introduction

According to WHO, cancer killed 9.6 million people in 2018, being the second leading cause of death globally (about 1 in 6 deaths were due to cancer). Tobacco use was the most important risk factor, causing approximately 22% of cancer deaths. Besides the human dimensions, there is also a significant and increasing economic impact. The total annual economic cost of cancer in 2010 was estimated at approximately US\$ 1.16 trillion (World Health Organization, 2018).

Cancer can affect any part of the body and, also in 2018, WHO identified the most common cancers:

- Lung (2.09 million cases)
- Breast (2.09 million cases)
- Colorectal (1.80 million cases)
- Prostate (1.28 million cases)
- Skin cancer (non-melanoma) (1.04 million cases)
- Stomach (1.03 million cases)

And, the most lethal cancers:

- Lung (1.76 million deaths)
- Colorectal (862 000 deaths)
- Stomach (783 000 deaths)
- Liver (782 000 deaths)
- Breast (627 000 deaths)

The previous numbers help us to understand why cancer has received increasing worldwide attention, from governments to the research community. The dimension of the problem is so significant that it has motivated several cooperation projects such as the GDSC database (Yang et al., 2013) (Genomics of Drug Sensitivity in Cancer, 2020).

The GDSC is one of the largest public resource for information on drug sensitivity in cancer cells. It is based on three types of datasets: (1) genomic datasets for cell lines, (2) cell line drug sensitivity data and (3) analysis of genomic features of drug sensitivity.

In this project, we use datasets (1) to obtain the cancer cell lines profiles and dataset (2) to obtain a drug-response matrix. Hence, the main variables are cell lines, compounds and  $IC_{50}$ . From a recommender system perspective: cell lines represent users (whose profile is characterized by a gene expression profile), compounds represent items and  $IC_{50}$  represent ratings given by users to each item (in other words, the drugs 'efficacy').

Regarding the project's unit of analysis - cancer cell lines - GDSC offers a wide collection: 1018 cancer cell lines from 50 different body parts.

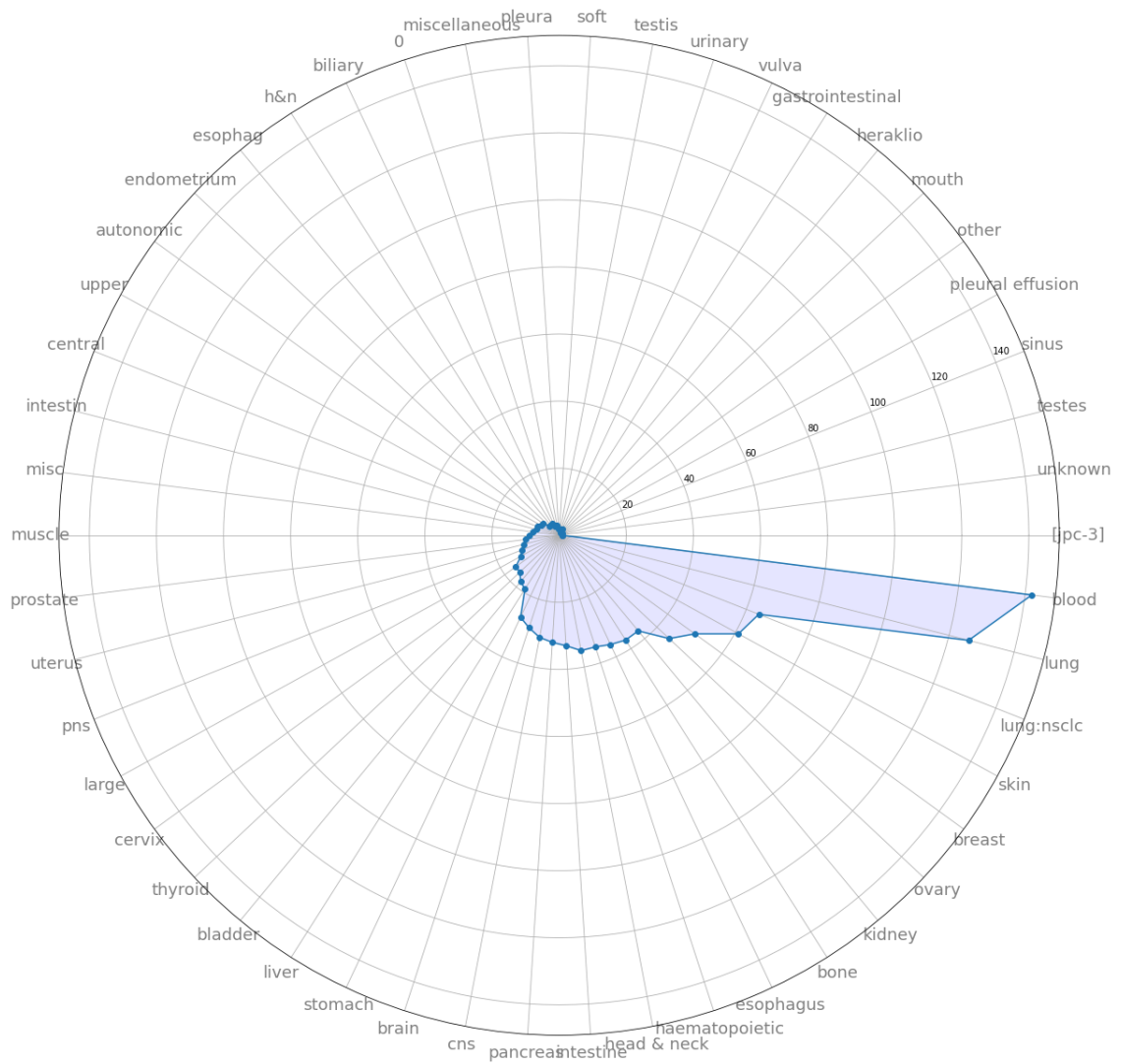


Figure 3.2 Body parts origins (1018 cancer cell lines)

Of the total number of cancer cell lines, this project makes use of 927, representing 45 different body parts, as explained in the next section.

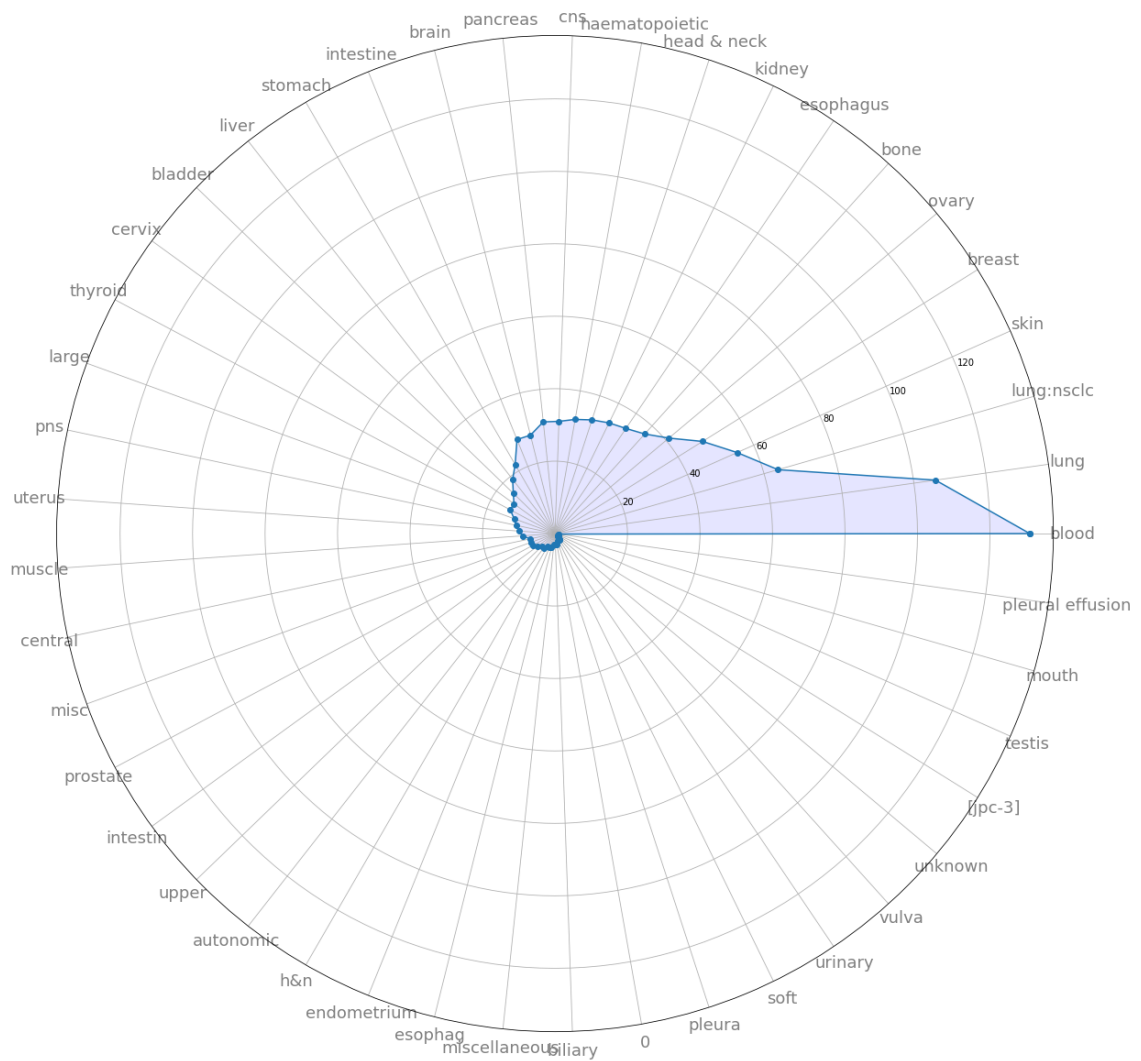


Figure 3.3 - Body parts origins (927 cancer cell lines)

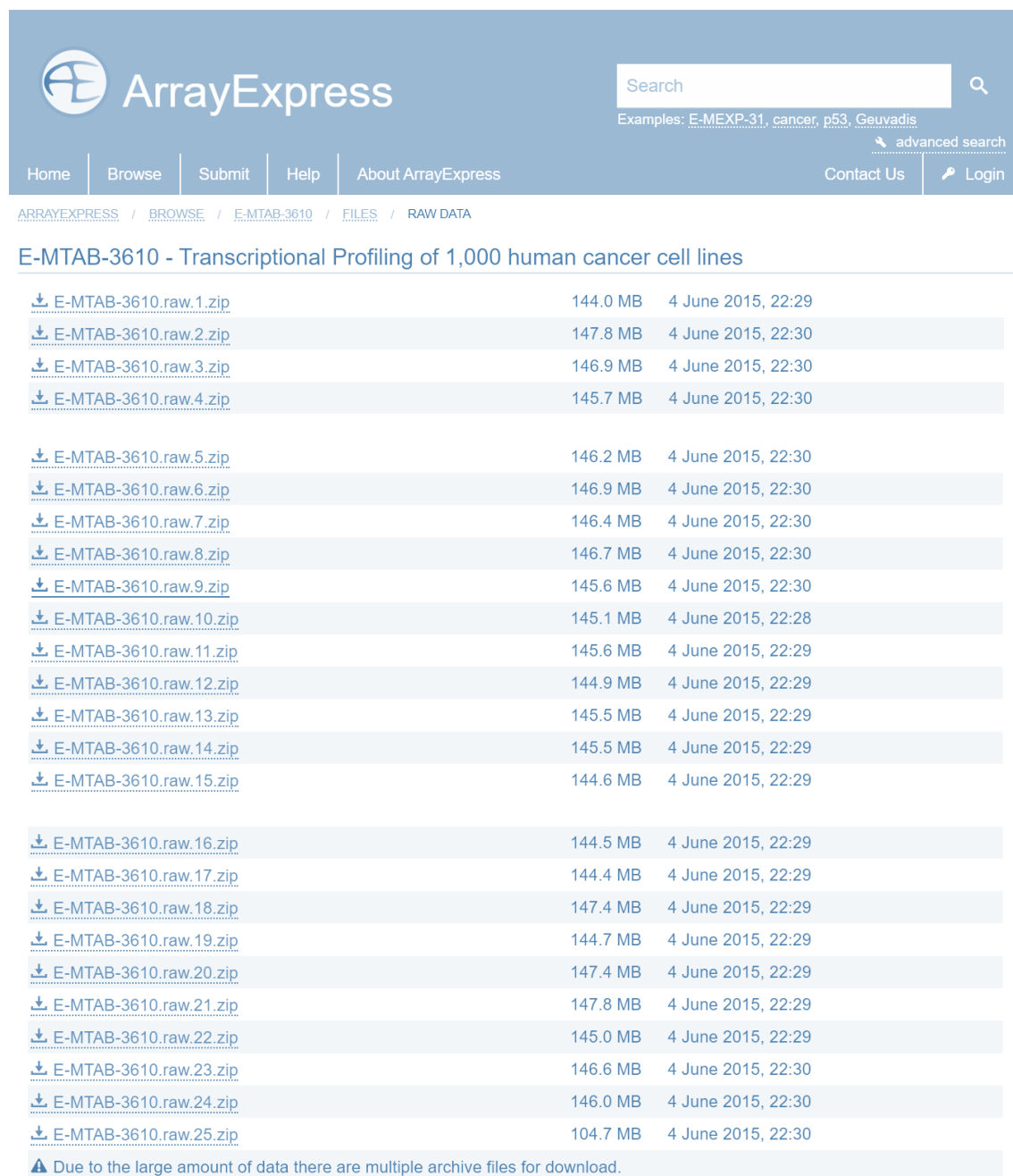
### 3.2.2 Cell lines

Datasets (1) contain a collection of > 1000 different cell lines which “represent the spectrum of common and rare types of adult and childhood cancers of epithelial, mesenchymal and haematopoietic origin” (Yang et al., 2013).

A wide genomic characterization of these cell lines has been made available by the Wellcome Trust Sanger Institute: mutation, copy number, methylation, and expression. Nevertheless, it is commonly recognized that gene expression data provides “the best predictive power” (Costello et al., 2014). In fact, gene expression is the common choice

regarding the development of cancer drug recommender systems that use GDSC1. Therefore, we choose gene expression data to profile the cancer cell lines.

The gene expression data can be found at the ArrayExpress website:



ArrayExpress

Search

Examples: E-MEXP-31, cancer, p53, Geuvadis

advanced search

Home Browse Submit Help About ArrayExpress Contact Us Login

ARRAYEXPRESS / BROWSE / E-MTAB-3610 / FILES / RAW DATA

### E-MTAB-3610 - Transcriptional Profiling of 1,000 human cancer cell lines

<a href="#">E-MTAB-3610.raw.1.zip</a>	144.0 MB	4 June 2015, 22:29
<a href="#">E-MTAB-3610.raw.2.zip</a>	147.8 MB	4 June 2015, 22:30
<a href="#">E-MTAB-3610.raw.3.zip</a>	146.9 MB	4 June 2015, 22:30
<a href="#">E-MTAB-3610.raw.4.zip</a>	145.7 MB	4 June 2015, 22:30
<a href="#">E-MTAB-3610.raw.5.zip</a>	146.2 MB	4 June 2015, 22:30
<a href="#">E-MTAB-3610.raw.6.zip</a>	146.9 MB	4 June 2015, 22:30
<a href="#">E-MTAB-3610.raw.7.zip</a>	146.4 MB	4 June 2015, 22:30
<a href="#">E-MTAB-3610.raw.8.zip</a>	146.7 MB	4 June 2015, 22:30
<a href="#">E-MTAB-3610.raw.9.zip</a>	145.6 MB	4 June 2015, 22:30
<a href="#">E-MTAB-3610.raw.10.zip</a>	145.1 MB	4 June 2015, 22:28
<a href="#">E-MTAB-3610.raw.11.zip</a>	145.6 MB	4 June 2015, 22:29
<a href="#">E-MTAB-3610.raw.12.zip</a>	144.9 MB	4 June 2015, 22:29
<a href="#">E-MTAB-3610.raw.13.zip</a>	145.5 MB	4 June 2015, 22:29
<a href="#">E-MTAB-3610.raw.14.zip</a>	145.5 MB	4 June 2015, 22:29
<a href="#">E-MTAB-3610.raw.15.zip</a>	144.6 MB	4 June 2015, 22:29
<a href="#">E-MTAB-3610.raw.16.zip</a>	144.5 MB	4 June 2015, 22:29
<a href="#">E-MTAB-3610.raw.17.zip</a>	144.4 MB	4 June 2015, 22:29
<a href="#">E-MTAB-3610.raw.18.zip</a>	147.4 MB	4 June 2015, 22:29
<a href="#">E-MTAB-3610.raw.19.zip</a>	144.7 MB	4 June 2015, 22:29
<a href="#">E-MTAB-3610.raw.20.zip</a>	147.4 MB	4 June 2015, 22:29
<a href="#">E-MTAB-3610.raw.21.zip</a>	147.8 MB	4 June 2015, 22:29
<a href="#">E-MTAB-3610.raw.22.zip</a>	145.0 MB	4 June 2015, 22:29
<a href="#">E-MTAB-3610.raw.23.zip</a>	146.6 MB	4 June 2015, 22:30
<a href="#">E-MTAB-3610.raw.24.zip</a>	146.0 MB	4 June 2015, 22:30
<a href="#">E-MTAB-3610.raw.25.zip</a>	104.7 MB	4 June 2015, 22:30

Due to the large amount of data there are multiple archive files for download.

Figure 3.4 Gene expression data

Adapted from (ArrayExpress, 2020)

It is important to notice that, although ArrayExpress website provides the gene expression profiles of > 1000 cell lines, only 987 cell lines were drug screened for the dataset GDSC1 (release 8.2). Also, exploratory data analysis unfolds that only 928 of those cell lines have

their DNA microarray available at the website. Later, while conducting the experiments, one of them was found to have its CEL file corrupted. Therefore, the final number of cancer cell lines used in this project is 927.

As shown in Figure 6.2, due to the large amount of data, there are multiple archive files for download. For example, *E-MTAB-3610.raw.1.zip* archive file comprises the first 41 gene expressions:

meu disco > ... > GeneExpressionProfiles > raw1









Nome ↓	Proprietário	Última modificação	Tamanho do ficheiro
 5500994173212120213068_A05.cel	eu	25/02/2020 eu	12 MB
 5500994173212120213068_A04.cel	eu	25/02/2020 eu	12 MB
 5500994173212120213068_A03.cel	eu	25/02/2020 eu	12 MB
 5500994173212120213068_A02.cel	eu	25/02/2020 eu	12 MB
 5500994173212120213068_A01.cel	eu	25/02/2020 eu	13 MB
 5500994172948120113978_A04.cel	eu	25/02/2020 eu	13 MB
 5500994172948120113978_A03.cel	eu	25/02/2020 eu	13 MB
 5500994172948120113978_A02.cel	eu	25/02/2020 eu	13 MB

Figure 3.5 *E-MTAB-3610.raw.1.zip* archive file (sample view showing 8 files of the 41 available)

Each gene expression is stored in a specific format file – a CEL file (created by an Affymetrix DNA microarray image analysis software) – containing a 2-D matrix (744×744):

```
[[ 242.8 4950.4 806. ... 8229.7 264.3 8619.9]
 [5117.4 1263.3 5163.6 ... 420. 8223.5 368.3]
 [1100.5 5229.1 1231.3 ... 8121.9 430. 8327.4]
 ...
 [7417.5 667.5 7322.9 ... 432.8 8585.3 330. ]
 [ 625.3 7490.7 660.8 ... 8604.5 416.3 8880.1]
 [7082.7 472.5 7517.6 ... 331.5 8635.3 245.8]]
```

Figure 3.6 Example of a CEL file (storing the gene expression of a cancer cell line)

This matrix represents an image – a DNA microarray image which is, as previously explained (see Chapter 2 Genomics Background), the most common way to represent the gene expression profile of a cell line.



The microscopic DNA spots mark the positions of specific genes of the cell line (thus, each cell line will be characterized by  $744 \times 744 = 553\,536$  different genes). Thus, each number of the matrix represents the expression (or, in other words, the intensity of the manifestation) of the corresponding gene after the artificial stimulation of the cell line.

Consequently, each cell line will have its characteristic gene expression profile that will be like a “fingerprint” of the cell line.

The DNA microarray image stored in the previous matrix is:

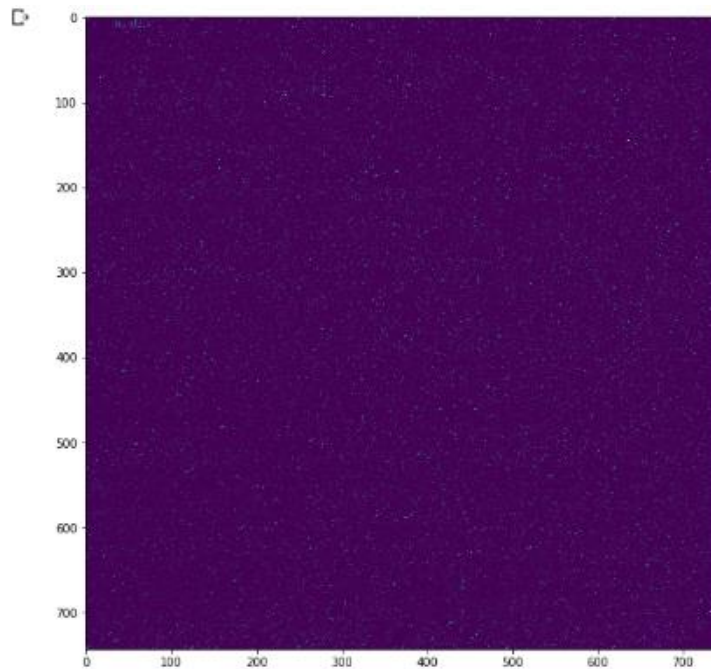


Figure 3.7 Example of a DNA microarray image

Although it is difficult for the human eye to visualize the 553 536 gene spots, this is an easy task for a computer.

In order to allow a cell line correspondence, the CEL filename contains the identification of the corresponding assay (used to obtain that DNA microarray). This identification, matched with additional information available at the ArrayExpress website, allows us to identify the tested cell line.

For example, the DNA microarray stored in the file *5500994173212120213068\_A01.cel* belongs to the cell line UACC-812.

## E-MTAB-3610 - Transcriptional Profiling of 1,000 human cancer cell lines

Display full sample data table [Export table in Tab-delimited format](#)

Page 1 2 3 4 5 6 .. 41 Showing 1-25 of 1018 rows Page size 25 50 100 250 500

Source Name ▲	Sample Attributes		Variables	Assay	Links to Data
	organism	cell line	cell_line	Assay Name	Raw
J132_EPA01P10_UACC-812_breast_910910	Homo sapiens	UACC-812	UACC-812	5500994173212120213068_A01	<a href="#">↓</a>
J132_EPA01P1_CAL-120_Breast_906826	Homo sapiens	CAL-120	CAL-120	5500994157493061613625_A01	<a href="#">↓</a>
J132_EPA01P2_201T_Lung:NSCLC_1287381	Homo sapiens	201T	201T	5500994158987071513209_A01	<a href="#">↓</a>
J132_EPA01P3_EVSA-T_Breast_906862	Homo sapiens	EVSA-T	EVSA-T	5500994172383112813929_A01	<a href="#">↓</a>

Figure 3.8 Sample data description

Source (ArrayExpress, 2020)

For future use, the previous correspondences (cell line – assay – filename) are stored in a customized file:

## Cell lines &amp; Assays

```
[ ] cell_assay = pd.read_csv("/content/drive/My Drive/RecSys_Code/dataset/CellLines/AssaysDescription/E-MT")
```

cell\_assay.head(2)

	Characteristics[cell line]	Assay Name	Array Data File
0	UACC-812	5500994173212120213068_A01	5500994173212120213068_A01.cel
1	201T	5500994158987071513209_A01	5500994158987071513209_A01.cel

Figure 3.9 Cell lines, assays and filenames correspondence file (sample view)

Finally, and for evaluation purposes, the DNA microarray images belonging to the selected 927 cancer cell lines are divided into 4 folders (mutually exclusive but of different sizes): one folder with 852 cancer cell lines and three folders with 25 cancer cell lines each.

The three folders of minor size, representing sets of target users in the context of the proposed framework, are evaluated, one at a time. The resulting folder values, for hit-rate and average reciprocal hit-rate, are then averaged and taken as the final result.

### 3.2.3 Compounds and IC<sub>50</sub>

Regarding dataset (2), featuring the cell line drug sensitivity data, the stored values are “generated from ongoing high-throughput screening performed by the Cancer Genome Project at the Wellcome Trust Sanger Institute and the Center for Molecular Therapeutics

at Massachusetts General Hospital using a collection of >1000 cell lines” (Yang et al., 2013).

The selected compounds are anticancer therapeutics “comprised of approved drugs used in the clinic, drugs undergoing clinical development and in clinical trials and tool compounds in early phase development” (Yang et al., 2013).

Cell lines are submitted to fluorescence-based cell viability assays following 72 hours of drug treatment and the results available “include the half maximal inhibitory concentration ( $IC_{50}$ ), the slope of the dose–response curve and the area under the curve for each experiment” (Yang et al., 2013).

The cell line drug sensitivity data is divided into two datasets - GDSC1 and GDSC2 – which are periodically updated and freely available without restriction. For the moment, the most recent release, and the one that we will use in this project, is Release 8.2 (Genomics of Drug Sensitivity in Cancer, 2020).

Release 8.2 (Feb. 2020)	
GDSC1	GDSC2
<b>Age</b> From 2010 to 2015	New
<b>Size</b> 987 Cell Lines 367 Compounds 310904 $IC_{50}$	809 Cell Lines 198 Compounds 135242 $IC_{50}$
<b>Assay</b> Resazurin or Syto60 72 hours	CellTitreGlo 72 hours

Table 3.1 Cell line drug sensitivity data

Source (Genomics of Drug Sensitivity in Cancer, 2020)

Compared with GDSC2, dataset GDSC1 provides access to more data and, hence, it is the chosen dataset to perform the experiments. The corresponding file can be found at [https://www.cancerrxgene.org/downloads/bulk\\_download](https://www.cancerrxgene.org/downloads/bulk_download). Each row of it corresponds to a specific drug tested in a specific cell line.

	DATASET	NLME_RESULT_ID	NLME_CURVE_ID	COSMIC_ID	CELL_LINE_NAME	SANGER_MODEL_ID	TCGA_DESC	DRUG_ID	DRUG_NAME
0	GDSC1	289	14442001	683665	MC-CAR	SIDM00636	MM	1	Erlotinib
1	GDSC1	289	14442813	684055	ES3	SIDM00265	UNCLASSIFIED	1	Erlotinib

Figure 3.10 Original GDSC1 dataset (sample view)

The file provides 19 variables from which we retrieve the ones of interest for this project, namely, “CELL\_LINE\_NAME”, “DRUG\_NAME” and “LN\_IC50” (which represents the natural log of the fitted IC<sub>50</sub>).

	CELL_LINE_NAME	DRUG_NAME	LN_IC50
0	MC-CAR	Erlotinib	2.395685
1	ES3	Erlotinib	3.140923

Figure 3.11 Retrieved variables from the GDSC1 dataset (sample view)

An exploratory data analysis shows that, although some cell lines appear 367 times (meaning that they were drug screened 367 times), only 345 different drugs were used. This happens because some drugs were tested more than once on the same cell line.

Also, for some cell lines, not all the 345 different drugs were tested (for example, cell line NCI-H250 has only 1 drug screening).

It is also important to notice that the IC<sub>50</sub> are not comparable between different drugs (each drug had a different dosage). Hence, it is necessary to normalize the IC<sub>50</sub> before further analysis.

### 3.2.4 Preprocessing

Prior to the experiments, several tasks of preprocessing are conducted regarding the retrieved cell line drug sensitivity data.

First, all rows with cancer cell lines whose gene expression profiles are not available are removed. Consequently, only the IC<sub>50</sub> of 927 cancer cell lines are kept.

Also, the cancer cell lines with repeated drugs (drugs tested more than once on the same cell line) are grouped by their mean  $IC_{50}$  value.

The variables are then rearranged into a new matrix (927×346):

cell line		Erlotinib	Rapamycin	Sunitinib	PHA-665752	MG-132	Paclitaxel	Cyclopamine	AZ628	Sorafenib	Tozasertib	Imatinib
0	MC-CAR	2.395685	-0.658244	2.161095	2.613997	0.530615	-3.647573	4.296779	2.055377	3.440293	0.701282	2.804463
1	ES3	3.140923	-0.067752	3.043634	2.935678	1.296998	-0.356496	4.887534	2.070470	2.047231	2.686398	1.528610
2	ES5	3.968757	0.586881	3.774145	2.696152	0.930800	0.680364	5.695762	2.666071	3.438754	3.723778	3.142681
3	ES7	2.692768	-0.025451	2.991234	2.491081	0.550690	0.075821	4.998695	2.392732	3.287412	2.983166	2.343982
4	EW-11	2.478678	-2.123159	3.600942	2.775492	0.232020	0.920329	5.716290	1.087535	4.148989	3.892344	2.907550

Figure 3.12 Drug-response matrix – initial version (sample view)

Next, the values are converted from LN  $IC_{50}$  to  $IC_{50}$  ( $IC_{50} = \exp(\text{LN } IC_{50})$ ).

To normalize the  $IC_{50}$  values into a  $[0, 1]$  interval, we follow the method used by Menden, Iorio, Garnett, Mcdermott, & Benes (2013) by applying a logistic-like function:

$$\text{normalized } (IC_{50}) = \frac{1}{1 + (IC_{50})^{-0.1}} \quad \text{with } IC_{50} > 0$$

Therefore, the closer a normalized  $IC_{50}$  value is to zero, the more sensitive the cancer cell line is to the drug whereas the closer the normalized  $IC_{50}$  value is to 1, the more resistant the cancer cell line is.

The final drugs response matrix has 13,3% of missing  $IC_{50}$  values which are set equal to 0.5 (the neutral point in the chosen scale, i.e., the previous interval  $[0, 1]$ ).

Finally, a column named “array data file” is added.

	cell line	array data file	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG-132	Paclitaxel	Cyclopamine
0	UACC-812	5500994173212120213068_A01.cel	0.589740	0.448857	0.500000	0.563572	0.535375	0.500000	0.638925
1	201T	5500994158987071513209_A01.cel	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000
2	EVSA-T	5500994172383112813929_A01.cel	0.569937	0.430087	0.548258	0.570864	0.465061	0.413074	0.565090
3	KYSE-520	5500994158987071513202_A01.cel	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000
4	MS751	5500994158987071513207_A01.cel	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000

Figure 3.13 Drug-response matrix – final version (sample view)

### 3.3 Proposed framework

#### 3.3.1 Introduction

The framework proposed here has two main stages: user similarity measurement and cancer drug recommendation. Given a target cancer line (profiled by a DNA microarray image representing its gene expression profile), firstly, a search for the top-N most similar users is conducted and, secondly, using the retrieved information, a personalized cancer drug recommendation is presented, ranking the top-N most effective drugs for the target user.

#### 3.3.2 Stage 1 – users similarity measurement

The goal of this stage is to find the top-N most similar cancer cell lines regarding the target cancer cell line. This is done under two experiments: without and with wavelet transforms.

Figure 3.16 presents the framework used at experiment 1, which is done without wavelet transforms. Accordingly, given a target cancer cell line, whose drugs' response are unknown, its DNA microarray image (individual gene expression profile) will be compared against each already existing DNA microarray image belonging to cancer cell lines whose drugs' response, in turn, are known. This comparison will be performed using a similarity metric. The final results (similarity score) will allow us to sort the database in descending order, i.e., from the most to the least similar cell line.

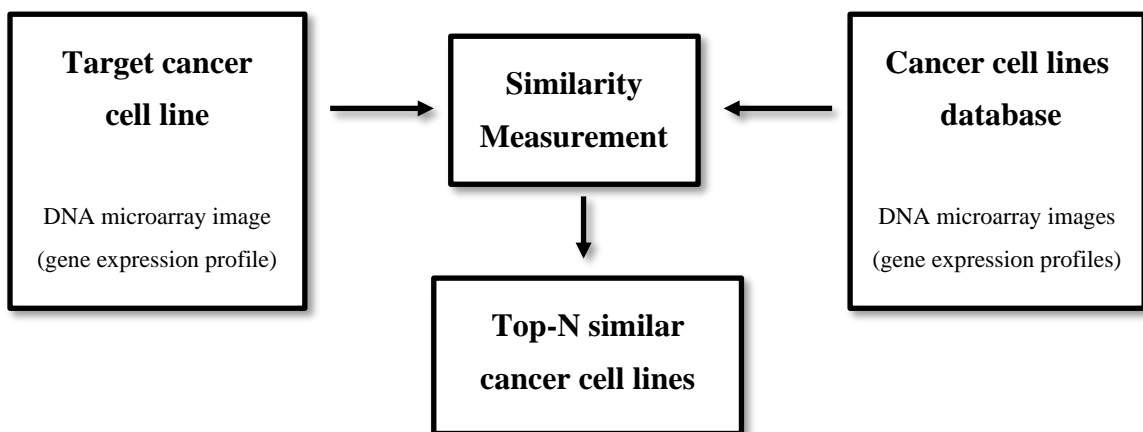


Figure 3.14 Experiment 1: without wavelet transforms

Figure 3.17 presents the framework at experiment 2, made with wavelet transforms.

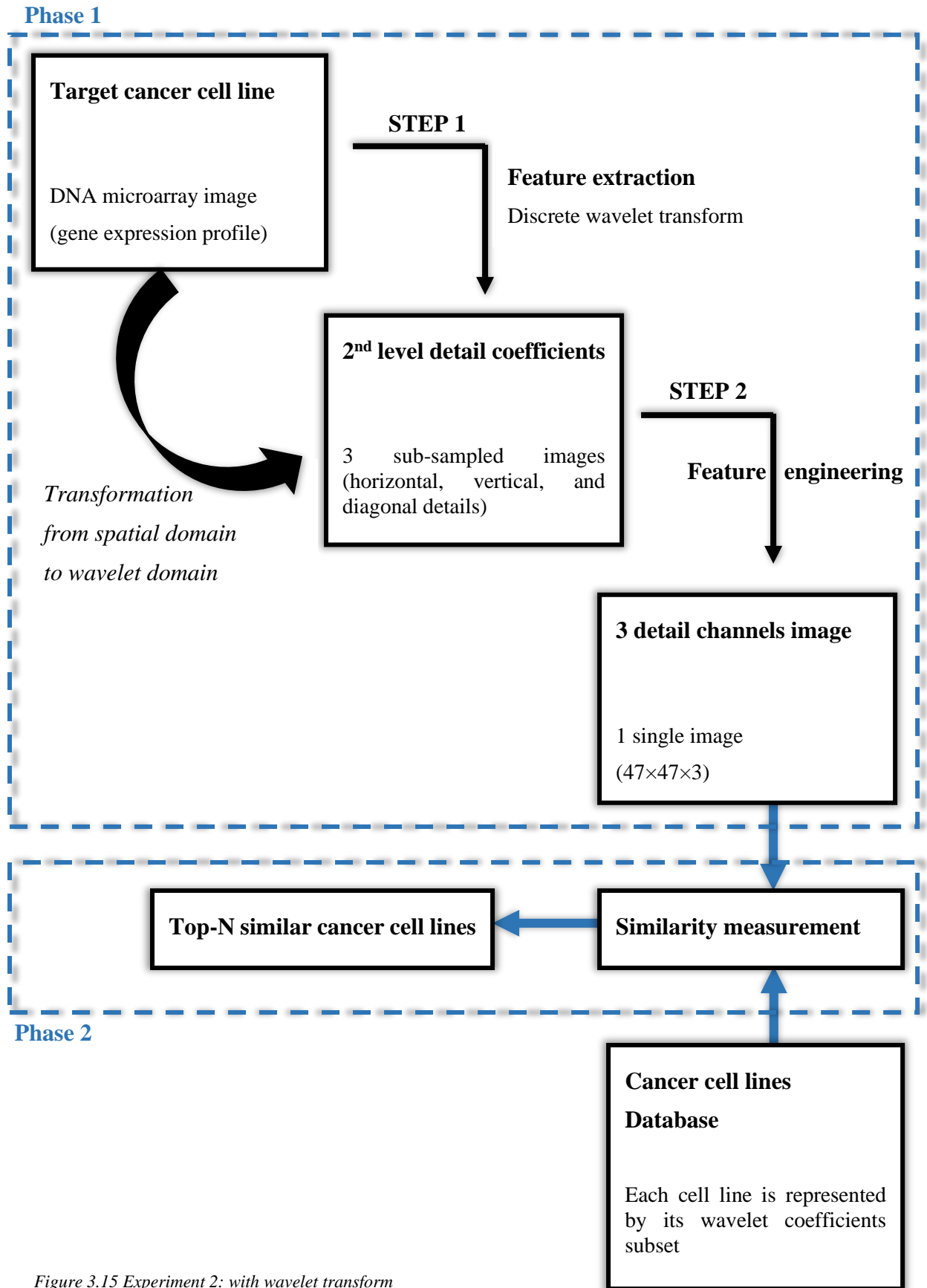


Figure 3.15 Experiment 2: with wavelet transform

The difference regarding the previous framework is that, this time, we will not use the original DNA microarray images to compute the similarity score. Instead, each of the original images (of both target and existing cell lines) will be transformed into new images, i.e., wavelet transformed images.

Firstly, we will apply a discrete wavelet transform to the original images. Secondly, the resulting three sub-sampled images (representing the horizontal, vertical, and diagonal wavelet detail coefficients) will be rearranged in order to form a new 3 detail channels image. Therefore, while the original images have only one channel, the new images will have three channels.

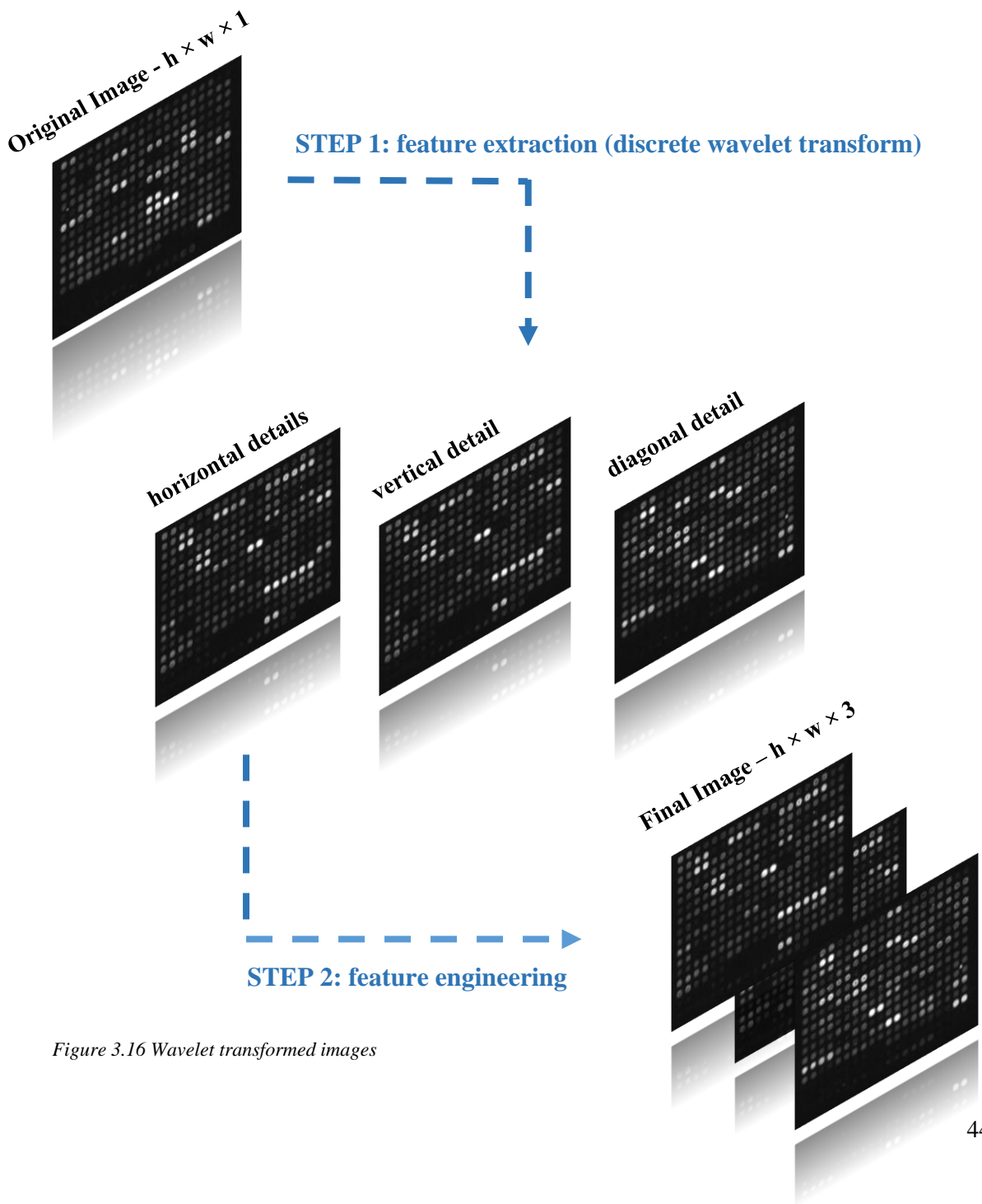


Figure 3.16 Wavelet transformed images



### 3.3.3 Stage 2 – cancer drug recommendation

The recommendation task is anchored in the assumption that similar cell lines have similar drugs' responses. Consequently, after finding the most similar users, their drugs' responses are retrieved.

Next, in order to discover which of the retrieved drugs are the best ones to recommend to the target cell line, the recommendation candidates (i.e., the retrieved drugs) are scored. Such score should reflect, not only how efficient a certain drug is to a cell line similar to the target one, but also how similar that existing cell is regarding the target cell.

Therefore, we establish a score that corresponds to the drug's rating (measured by its  $IC_{50}$ ) but weighted by the similarity score between the retrieved cell line and the target cell line. Also, if one drug appears more than once (a common situation that can occur since some drugs were tested in several distinct cell lines), its scores are added in order to strengthen that fact.

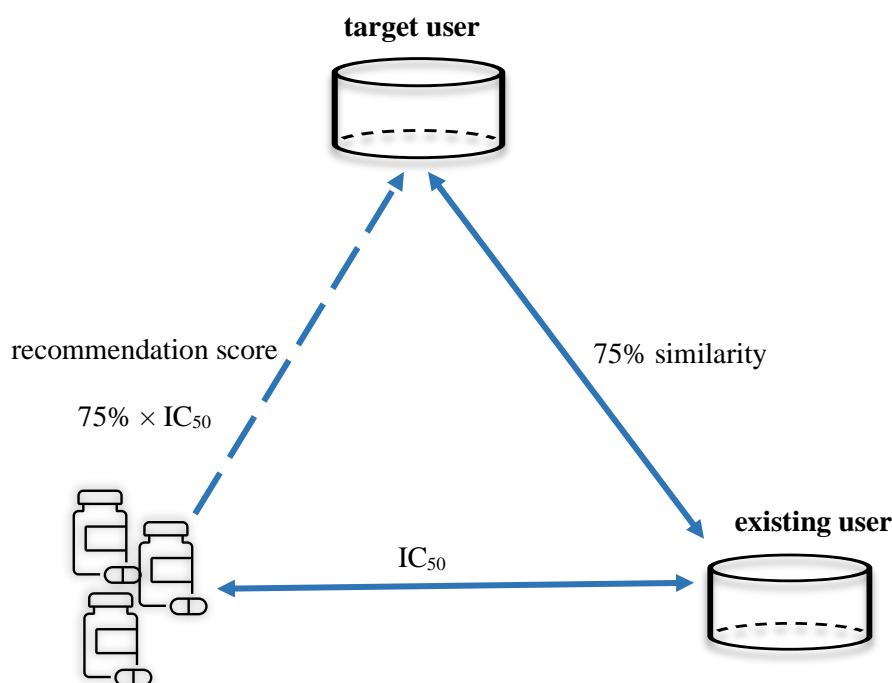


Figure 3.17 Example of a recommendation score

The final scores are sorted in ascending order (the lower the  $IC_{50}$  value is, the more efficient the compound is) and the top-N drugs are then presented as the most likely efficient ones for the target cell line.

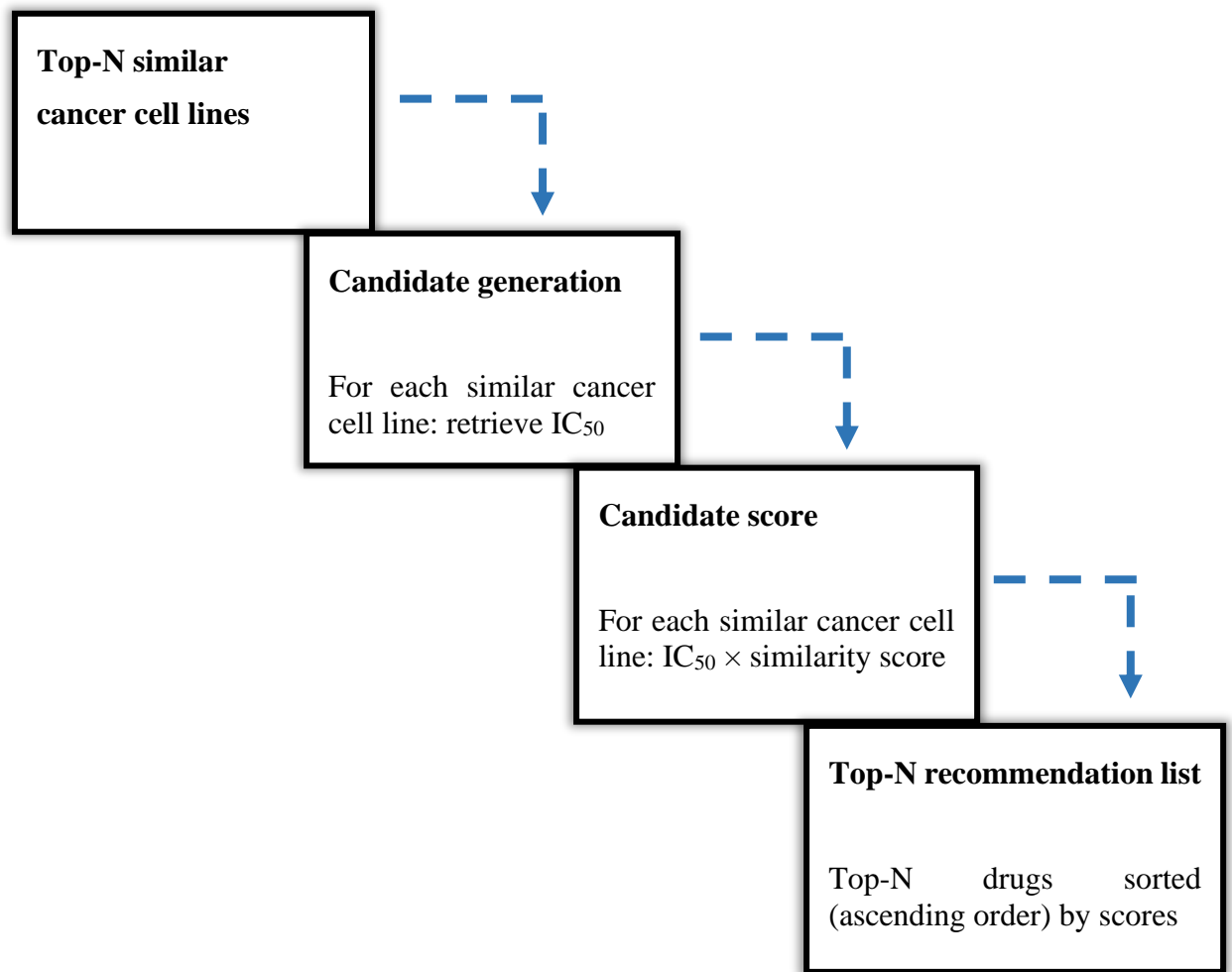


Figure 3.18 Cancer drug recommendation pipeline

### 3.4 Used tools

This project uses an environment provided by Google called Colaboratory (Google, 2020), or just Colab for short. Colab platform requires no setup to use and runs entirely in the cloud, allowing the implementation of machine learning models. Technically speaking, Colab is a hosted Jupyter notebook service available through a Google Drive. The code is executed in a virtual machine private to the user's Google account.

This infrastructure allows to write and execute Python code in a browser with zero configuration required and to freely access GPUs (Graphics Processing Unit) and TPUs (Tensor Processing Unit), accelerating the performance of linear algebra computation, which is used heavily in machine learning applications.

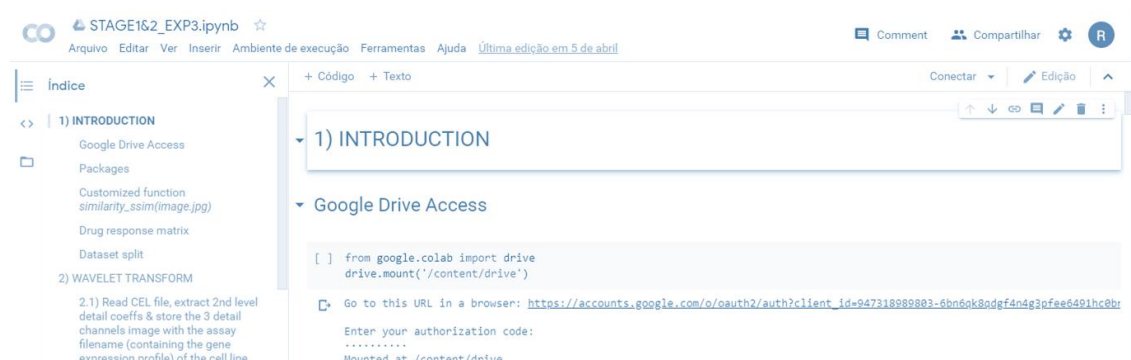


Figure 3.19 Example of a Google Colab Notebook

The benefits of using this cloud service over using our own local machines are several, for example:

- It comes with important Python packages pre-installed and ready to use (Numpy, Pandas, TensorFlow...) and, in addition, it allows the installation of further packages or upgrade of current ones.
- It provides interactive browser-based Jupyter Notebooks that can be easily viewed, edited, and executed by others (including non-technical audience), through a shareable link.
- Free GPU access. The GPU available in Colab vary over time, and often include Nvidia K80s, T4s, P4s and P100s.
- Free RAM of 12 GB with maximum extension of 25 GB.
- Storage of Notebooks on Google Drive.
- Document code with Markdown, making the Notebook layout well-organized and user-friendly.
- Load data from the Google drive.

Despite Colab being free, there are overall usage limits as well as IDLE timeout periods, maximum VM lifetime, GPU types available, and other factors that vary over time. For example, VM have maximum lifetimes that can be as much as 12 hours. Nevertheless, users interested in having resources beyond the limits of the free version may find useful Colab Pro (paid version).

In what concerns this project, there was no need to use GPU (especially suitable for deep learning tasks) and a 12 GB RAM was used to perform the computational tasks. Due to

the VM maximum lifetime constraint, several checkpoints were scheduled to save the results as they were becoming available.

Finally, regarding the Python packages/modules, the main ones used during the course of the project were:

- Numpy, for array computing.
- Pandas, for dataframes analysis and manipulation.
- OS, for interacting with the operating system (directories and files).
- Matplotlib and Seaborn, for data visualizations.
- Biopython, a specialized package for computational biology and bioinformatics.
- PyWavelets, for wavelet transform calculations.
- OpenCV, a computer vision library.
- Scikit-image, for computing the SSIM Index.

### **3.5 Ethical considerations and social responsibility**

The development of data-based solutions for cancer disease goes beyond the technical challenge and also raises ethical and social issues.

Machine learning systems are greatly shaped by the data they are fed. Consequently, they are prone to data bias. For example, if the algorithm was trained and evaluated with data over-representing a certain group, then the system will naturally become biased against under-represented groups. In a clinical context, a practical consequence of this bias might be the system performing better for certain social or ethnical groups.

Also, the development of these algorithms implies the use of large quantities of data and, consequently, issues like ownership and consent are relevant. Besides, in order to benefit from a personalized medicine algorithm, the patient needs to share his/her personal data. If the patient refuses to give such consent, this may lead to a “tension between consent and quality of care”. (Carter et al., 2020)

Another interesting issue regards responsibility. If a doctor relies on a ML algorithm to support a clinical decision, and a negative outcome happens, it might not be clear who should take the responsibility since, currently, there is a regulatory vacuum. This is especially relevant for the so-called “black box algorithms” or non-explainable AI.

It is urgent to bring these (and other related) issues to public discussion and find standard solutions if we truly want to benefit and incorporate AI healthcare solutions in the real world. Such transparency produces a vital feeling in the stakeholders (doctors, patients, healthcare providers, ...) – trustworthiness.



## **4 FINDINGS**

### **4.1 Introduction**

Experiments are performed on the benchmark dataset, called GDSC1 (release 8.2) (Yang et al., 2013). The scope is to assess if the use of wavelet transforms on the DNA microarray images contributes positively, or not, to the recommender system's performance. Therefore, the focus lies, not in achieving state-of-the-art results in terms of evaluation metrics (i.e., hit-rate and average reciprocal hit-rate), but on judging the impact of using wavelet transformed DNA microarray images (versus original images) on the proposed framework. To that aim, two experiments are conducted: one using the original DNA microarray images and another one using wavelet transforms to preprocess the images before feeding them to the recommender system.

To measure the similarity between images (i.e., original and wavelet transformed images), SSIM Index (Z. Wang, Bovik, Sheikh, & Simoncelli, 2004) is used. This metric compares two images using their structural information which suits well one of the main tasks of the proposed framework – the search of similar users based on structural patterns available, but concealed, on their DNA microarray images.

### **4.2 First experiment and its results – without wavelet transform**

Overall, in this experiment, the similarity between images is analyzed in a spatial domain (the original domain of the image).

In practical terms, this means that the similarity between the target cancer cell line (whose drugs' response are considered to be unknown) and the cancer cell lines of the database (whose drugs' response are known) is measured by applying the SSIM Index between the original DNA microarray image of the target cancer cell line and the DNA microarray images of the cell lines in the database.

The similarity measurements show that, in general, the similar cell lines that belong to the same top-5 have very close SSIM Indexes regarding the target one. Furthermore, after performing a pairwise similarity measurement among the cells of these top-5s, we conclude that they are also closely similar to each other. Therefore, we decide to use only the most similar cell line.

The drugs' response of this cancer cell line is then retrieved of the database. The retrieved normalized  $IC_{50}$  values are then weighted by the corresponding SSIM and sorted in

descending order (i.e., from the most to the least effective drug). The top-20 drugs of the previous ranking are taken as part of the top-20 recommendation list for the target cancer cell line.

Since the similarity measurement (stage 1), in this experiment, is very time consuming (using Colab's computational resources each image pair takes approximately 25 minutes), an adapted four-fold cross-validation strategy is used to evaluate the proposed framework. With that aim, the cancer cell lines are divided into four folders, mutually exclusive but of different sizes.

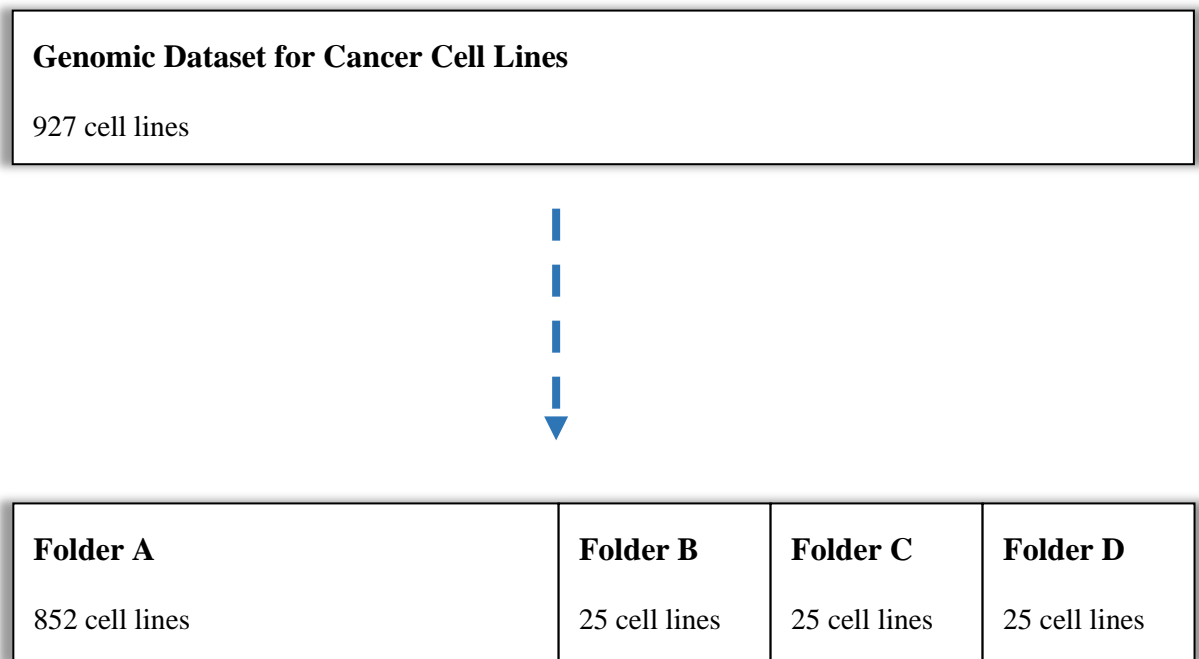


Figure 4.1 Dataset division for evaluation purposes

The framework is evaluated using folders B, C and D, one at a time, and taking the average result as the final result.

The final top-20 hit-rate is 11.31 and the average reciprocal top-20 hit-rate is 2.39.

Equally remarkable is the execution time of the experiment, approximately 30 hours.

These results are set as a baseline for the next experiment.



### 4.3 Second experiment and its results – with wavelet transform

Overall, in this experiment, the similarity between images is analyzed in a frequency domain, i.e., between wavelet transformed images.

For most applications, the chosen wavelet type is Haar or Daubechies. Daubechies, although conceptually and computationally more complex than Haar, can pick up details that are missed by Haar. Thus, we choose a Daubechies approach to perform the wavelet transform, as the option that may confirm the hypothesis of improving the recommender system performance through the transformation of DNA microarray images.

Also, when computing wavelet decomposition, it is possible to use different resolutions (decomposition levels) to convolve the wavelet with the image. Classification using Daubechies 7 with four or five levels of decomposition reported good performance (Nanni & Lumini, 2011). Therefore, we initially decide to carry out a 4-level decomposition. However, the results suggest that the wavelet transformed image at level 4 is too much compressed resulting in a loss of information. Hence, we proceed instead with a 2-level decomposition.

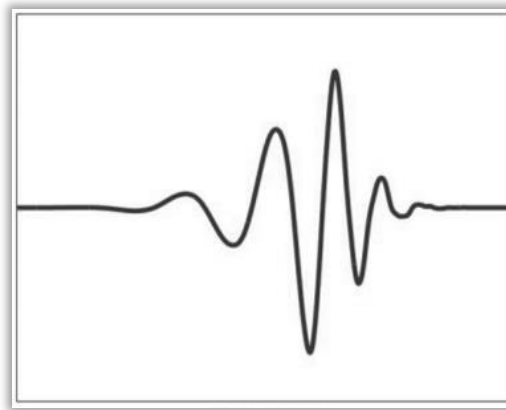


Figure 4.2 Daubechies 7 wavelet

Although approximation coefficients characterize the major trends contained in the gene expression profiles (i.e., the essential information of the microarray data), we only use the detail coefficients because, as observed by Wan & Zhou (2010), “they have better discriminating capacities and make the classification of two classes of subtle differences possible”. In the same line of thought, Liu & Bai (2009) state that “the purpose of detail coefficients is to detect localized features in one of the gene expression profile”.

At the 2nd level of decomposition, and considering only the detail coefficients, three output images are obtained: one in horizontal, other in vertical and another in diagonal directions of the image. These images are then combined, forming a unique 3-detail channels image (similar to a 3-color channels image, using for example, the RGB - Red-Green-Blue color model, but, instead of color channels, the image has the detail channels Horizontal-Vertical-Diagonal). The resulting image is the one used to assess the similarity.

One important advantage of the previous approach is that allows us to preserve spatial patterns (since the coefficients' positions are kept in the image).

Following the same evaluation method as in experiment 1, the final top-20 hit-rate is 12.21 and the average reciprocal top-20 hit-rate is 2.53.

The experiment takes, approximately, 1.5 hours to execute.

#### **4.4 Recommendation example - cancer cell line “HH”**

Using cancer cell line “HH” as the target cancer cell line, we will now provide a recommendation example. Cancer cell line “HH” has its origin in the body part identified as “blood”.

After running experiment 1, cancer cell line “JVM-3” is retrieved as the most similar with a SSIM index of 69.12 %. Like the target “HH”, this cell line belongs to the body part “blood”.

On its turn, experiment 2 returns cancer cell line “QIMR-WIL”, with a SSIM index of 81.41 %, as the most similar. This cell line also belongs to the body part “blood” but, this time, the SSIM Index is higher regarding the previous one.

The following tables provide an overview of the recommendation lists (top-20 drugs) generated by each of the experiments.

Target Cell "HH"	
Drug	Rank
Daporinad	1
Bortezomib	2
SN-38	3
Sepantronium bromide	4
Temsirolimus	5
THZ-2-102-1	6
Omipalisib	7
Vinblastine	8
Vinorelbine	9
ARRY-520	10
Dacinostat	11
Rapamycin	12
Panobinostat	13
Epothilone B	14
AZD4877	15
Ispinesib Mesylate	16
Dactolisib	17
PLK_6522	18
NSC319726	19
Luminespib	20

Table 4.1 Target cancer cell line "HH" - ground truth ranking

Experiment 1 – Similar Cell "JVM-3"	
Drug	True Rank
1 <sup>st</sup> ) SN-38	3
2 <sup>nd</sup> ) Vinblastine	8
3 <sup>rd</sup> ) Docetaxel	23
4 <sup>th</sup> ) AZD4877	15
5 <sup>th</sup> ) Daporinad	1
6 <sup>th</sup> ) Temsirolimus	5
7 <sup>th</sup> ) Sepantronium bromide	4
8 <sup>th</sup> ) Thapsigargin	36
9 <sup>th</sup> ) Methotrexate	24
10 <sup>th</sup> ) PLK_6522	18
11 <sup>th</sup> ) THZ-2-102-1	6
12 <sup>th</sup> ) Bortezomib	2
13 <sup>th</sup> ) Panobinostat	13
14 <sup>th</sup> ) Dactolisib	17
15 <sup>th</sup> ) PARP_9482	123
16 <sup>th</sup> ) Lestaurtinib	60
17 <sup>th</sup> ) Dacinostat	11
18 <sup>th</sup> ) Elesclomol	42
19 <sup>th</sup> ) SNX-2112	21
20 <sup>th</sup> ) NSC319726	19

Table 4.2 Experiment 1 – recommendation list

Experiment 2 – Similar Cell "QIMR-WIL"	
Drug	True Rank
1 <sup>st</sup> ) SN-38	3
2 <sup>nd</sup> ) NSC319726	19
3 <sup>rd</sup> ) Epothilone B	14
4 <sup>th</sup> ) Rapamycin	12
5 <sup>th</sup> ) Omipalisib	7
6 <sup>th</sup> ) Bortezomib	2
7 <sup>th</sup> ) ARRY-520	10
8 <sup>th</sup> ) Gemcitabine	22
9 <sup>th</sup> ) Vinorelbine	9
10 <sup>th</sup> ) Daporinad	1
11 <sup>th</sup> ) Sepantronium bromide	4
12 <sup>th</sup> ) Panobinostat	13
13 <sup>th</sup> ) Vinblastine	8
14 <sup>th</sup> ) Docetaxel	23
15 <sup>th</sup> ) Paclitaxel	25
16 <sup>th</sup> ) Luminespib	20
17 <sup>th</sup> ) Dacinostat	11
18 <sup>th</sup> ) Ispinesib Mesylate	16
19 <sup>th</sup> ) Temsirolimus	5
20 <sup>th</sup> ) THZ-2-102-1	6

Table 4.3 Experiment 2 – recommendation list

As we can observe, experiment 1 outputs a recommendation list with 13 hits whereas experiment 2 recommendation list successfully identifies 17 hits. Moreover, experiment 1 has an average reciprocal hit-rate of 2.79 whereas for experiment 2 its value is 3.33.



## **5 DISCUSSION OF THE RESULTS**

In this project, a recommender system engine, having DNA microarrays (patients' profiles) as inputs and drug recommendation lists as outputs, was implemented. It was expected from the algorithm the ability to identify the top-20 most relevant compounds from 345 possible drugs. In a random scenario, i.e. without any data-based approach, the probability of being successful on finding those 20 drugs is less than 0.0000000000002%.

Therefore, although in this preliminary study our research efforts did not focus on implementing a cutting-edge recommender system in terms of evaluation metrics, we consider very satisfying, as starting points, the final evaluation metrics obtained:

- experiment 1 (with original DNA microarray images): top-20 hit-rate was 11.31 and average reciprocal top-20 hit-rate was 2.39.
- experiment 2 (with wavelet transformed images): top-20 hit-rate was 12.21 and average reciprocal top-20 hit-rate is 2.53.

Overall, from a universe of 345 drugs, our recommender system was able to suggest a list of 20 in which, on average, more than 50% were in fact relevant drugs. Once again, in a random scenario, the probability of obtaining this result would be higher than the previous one, but still less than 0.0000005%.

Also, in what concerns the average reciprocal top-20 hit-rate, measuring the ability of the recommender system to display relevant drugs on top positions, the results are valuable, keeping in mind that the maximum average reciprocal top-20 hit-rate (achieved when all the recommended items are relevant) is 3.60.

The next figures show the distribution of the results, top-20 hit-rate and average reciprocal top-20 hit-rate, across the 75 target cancer cell lines.



Figure 5.1 Hit-rate box plot

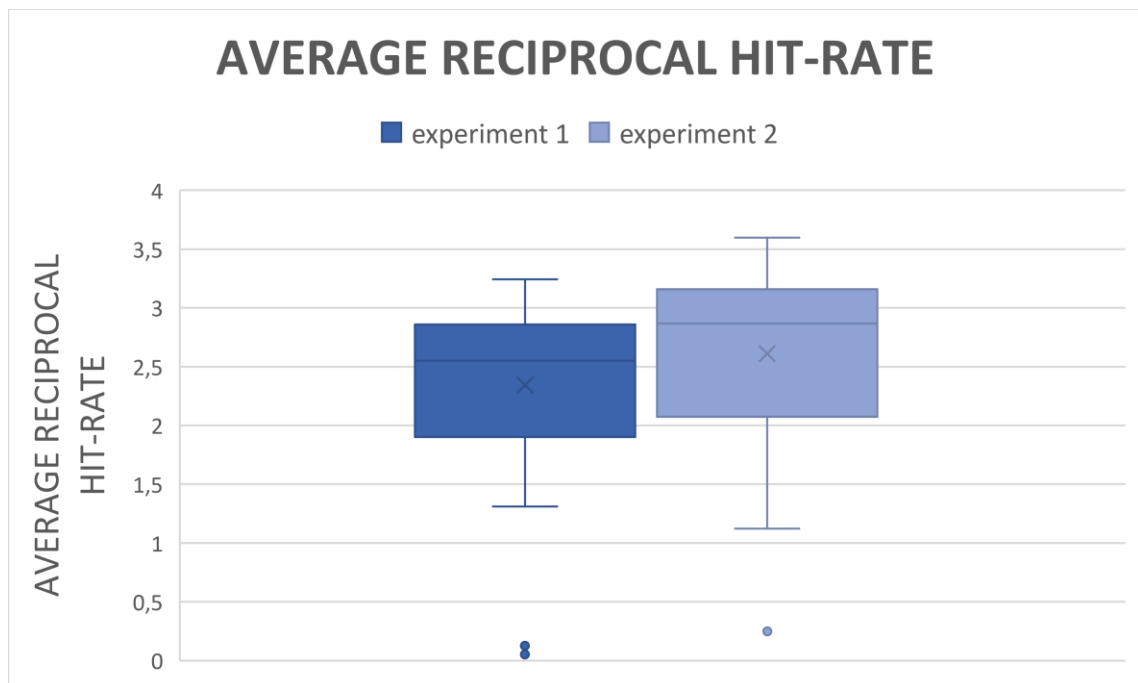


Figure 5.2 Average reciprocal hit-rate box plot

On one hand, Figure 5.1 allows to conclude that hit-rate results are less dispersed in experiment 2 (not only its interquartile range but also the range of scores is smaller regarding experiment 1). In addition, although both maximum hit-rate scores are on par, the minimum score of experiment 2 is higher. On the other hand, according to Figure 5.2, average reciprocal hit-rates are less dispersed in experiment 1.

Taking a closer look to the recommendation example given in Section 4.4, it is also interesting to notice that the retrieved similar cell line in experiment 2 has a higher SSIM Index (81.41%) with respect to the one retrieved in experiment 1 (69.12%). In fact, the SSIM range is higher in experiment 2. For example, regarding folder B, experiment 1 has a SSIM range of [67.49 – 81.78%] whereas in experiment 2 it is [76.46 – 89.57%] which reflects the denoising power of wavelets transforms.

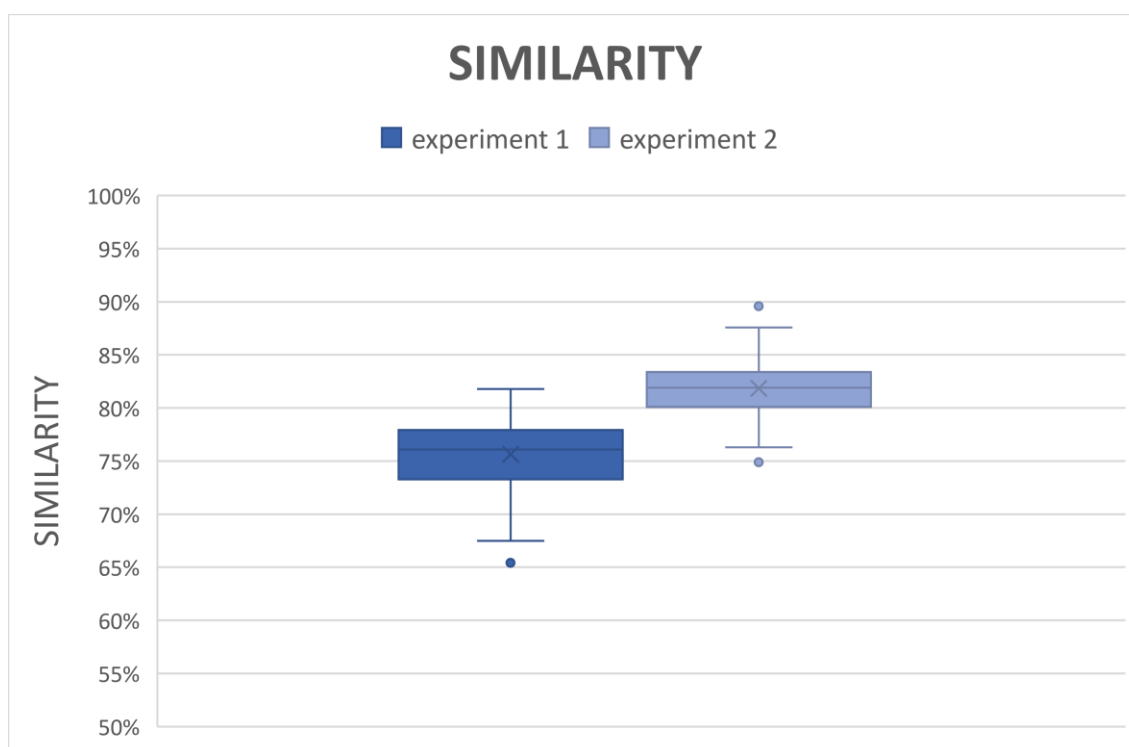


Figure 5.3 Similarity box plot

Also, although in the aforementioned example both similar cell lines (i.e. from experiment 1 and experiment 2) belong to the same body part as the target cell (i.e. “blood”), this is not always the case. For example, the target cancer cell line “HN” belongs to the body



part “endometrium” and, while experiment 1 retrieves a similar cell line from the same location, experiment 2 identifies a similar cell line from a different body part – “mouth”. Furthermore, in 52% of the instances, experiment 2 chooses a different similar cell line with respect to experiment 1. Figure 5.4 summarizes the impact of such choice on the results.

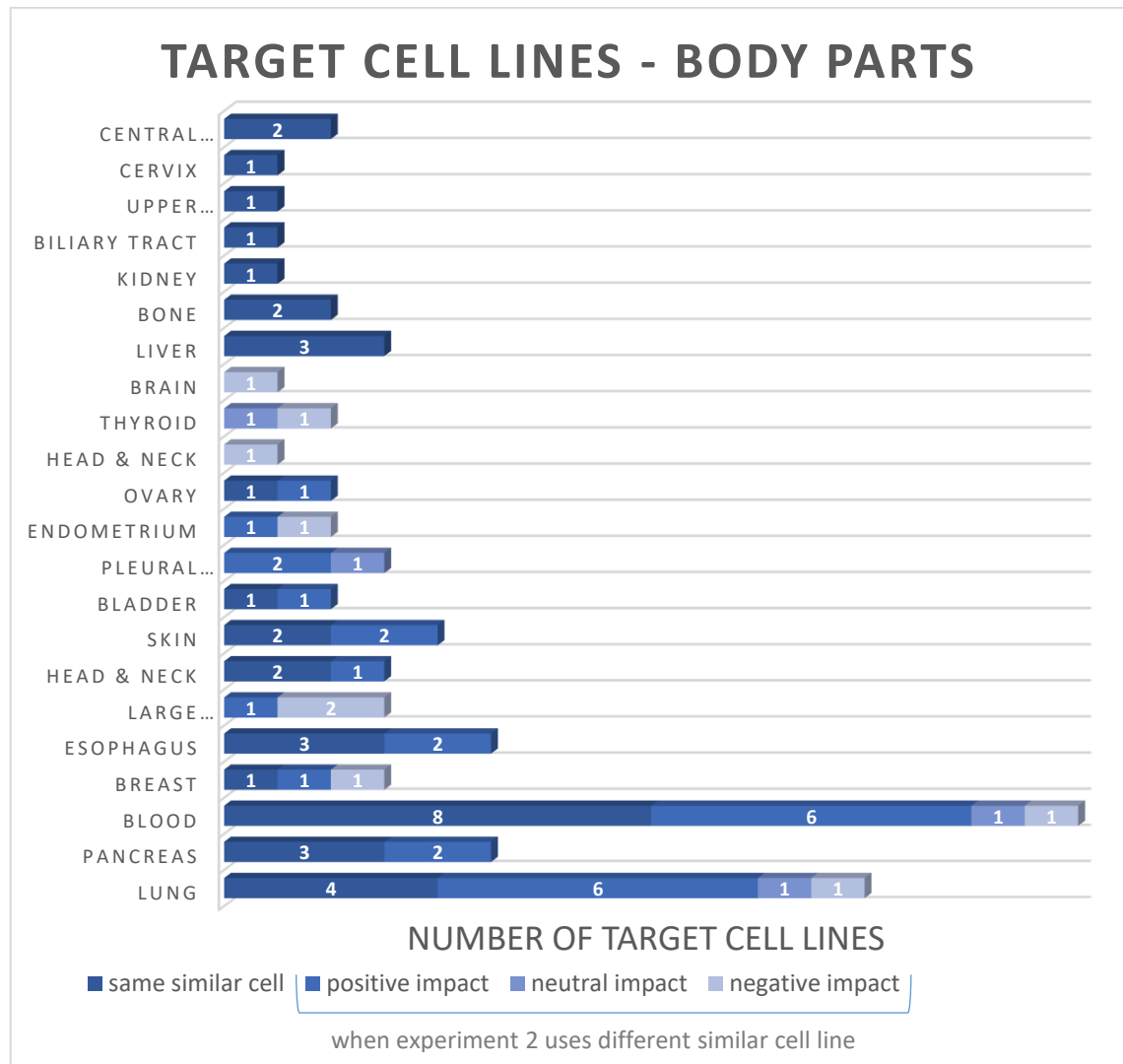


Figure 5.4 Experiments' final results according to target body part

Whenever experiment 2 chooses a different similar cell line, 66.67% of the times this conducts to better results and 23.08% this leads to worse performance with respect to experiment 1 (10.26% of the times the impact is neutral, conducting to similar results although different cell lines are used).

Some of the negative results of experiment 2 may be influenced by the drug sparsity on the target cell line side, i.e. the fact that the target similar cell line does not have all the IC50 values available. An example that illustrates this hypothesis is target cancer cell line “HCC-78”. Experiment 2 retrieves cell line “LC-2-ad” as the most similar, however, 8 drugs (“Epothilone B”, “Thapsigargin”, “Rapamycin”, “Paclitaxel”, “Bortezomib”, “GW843682X”, “BI-2536” and “Mitomycin-C”) of the top-20 drugs have not been tested in “HCC-78”, whereas experiment 1 retrieves cell line “HuCCT1 which has only 2 non tested drugs (“Epothilone B”, “Thapsigargin”). Experiment 1 has a top-20 hit-rate of 15 and an average reciprocal hit-rate of 2.48 while experiment 2 has a top-20 hit-rate of 11 and an average reciprocal hit-rate of 1.12.

Overall, the final results confirm the initial hypothesis: the prior preprocessing of DNA microarray images, using wavelet transforms, improves the recommender system performance since the evaluation metrics of experiment 2 are, in fact, higher than those of experiment 1. More specifically, experiment 2 has a top-20 hit-rate and an average reciprocal top-20 hit-rate, 4.5% and 3.89%, respectively, higher.

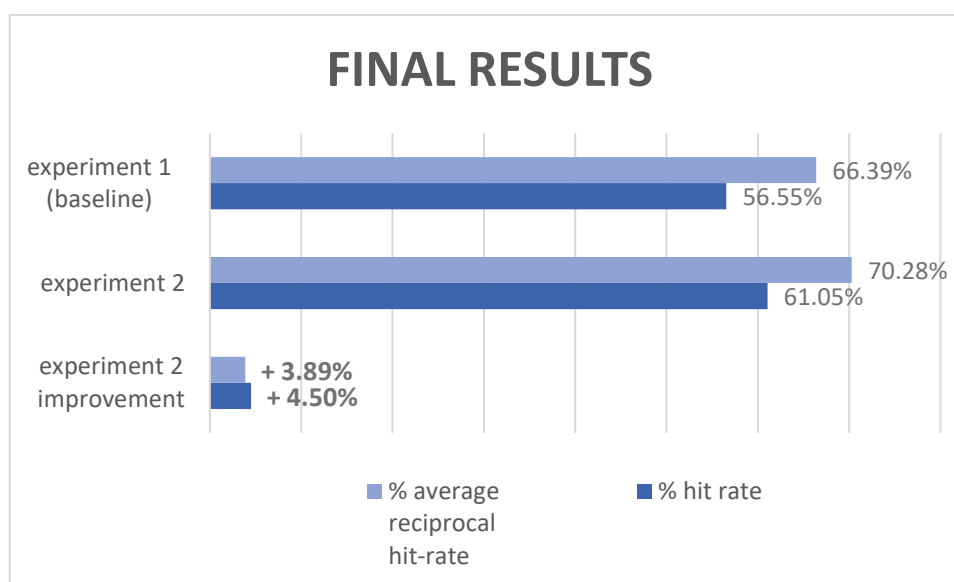


Figure 5.5 Final results

Moreover, it is also possible to conclude from the execution times of the experiments that this positive impact is also translated into an enhanced use of computational resources – 30 hours (experiment 1) versus 1.5 hours (experiment 2).



Figure 5.6 Experiments' execution time



## **6 CONCLUSIONS**

### **6.1 Contributions and implications**

Recommender systems, comprising the identification of the need and preferences of users, filtering the huge collection of data accordingly and displaying the best fitted options, are becoming more embracing. This study discusses and presents a novel framework used to implement a recommender system that proposes a personalized ranking of cancer drugs, combining techniques of image processing for feature enhancement.

The proposed framework has a first main stage that consists of measuring the user similarity, and a second main stage, consisting of the cancer drug recommendation. Then, two experiments are conducted. One using the original DNA microarray images and the other using wavelet transforms to preprocess the images before feeding them to the recommender system.

The central core of the project is the assessment of the impact of using wavelet transformed DNA microarray images for measuring users' similarity in a recommender system's framework. To the best of our knowledge, it is the first time that a research project addresses this problem.

The conducted experiments confirm the initial hypothesis that wavelet transformed DNA microarray images enhance the recommender system performance by improving the search of cancer cell lines with similar profile to the one of the target cancer cell line.

Moreover, experiment 2 takes only 5 percent of the execution time of experiment 1. So, also from a computational point of view, experiment 2 is more efficient and more suitable for a real-world application.

Therefore, we conclude that properly chosen wavelet transformed DNA microarray images, not only uncover richer information for the users' similarity search (with positive impact, as seen previously, in the recommendation task), but also efficiently compress the DNA microarray images, optimizing computational resources.

Future research can benefit from these findings by incorporating wavelet transformed inputs to their recommender systems frameworks.

## **6.2 Limitations**

Recommender systems need a lot of data to efficiently make recommendations. Not only data quantity but also diversity is important. A limitation of the dataset used in the experiments (besides data quantity) is that some body parts are over-represented (for example, “blood” with over 140 cancer cell lines) while others are under-represented (for example, “intestine” with less than 20 cancer cell lines). This may jeopardize the search of similar cancer cells lines if the target one belongs to an under-represented group. (Although, as previously discussed, it is not always the case that the retrieved similar cell lines belongs to the same body part that the target one.)

On the other hand, missing IC<sub>50</sub> may also compromise the evaluation results. For example, if a certain drug was tested in the target cell but not in the similar cell line, even if it was found to be relevant (or effective), the recommender engine will not suggest such drug. In this project we have decided to work under the worst-case scenario, accepting the sparsity on the drugs side.

Finally, due to computational constraints, especially during the similarity measurement of the original DNA microarray images, it was not possible to perform a cross-validation using the entire dataset, instead, 3 subsets of distinct 25 cancer cell lines were used to perform the evaluation.

## **6.3 Recommendations**

Since only 2<sup>nd</sup> level detail wavelet coefficients were used on this study, future research could investigate the effect of other variants of wavelet transformed DNA microarray images (e.g. simultaneous use of detail and approximation coefficients), with the scope of increasing even further the already existing gap between the evaluation metrics of the two experiments. Another interesting possibility is the replacement of the DNA microarray images for basal expression data, under the same scenarios - with and without wavelet transforms.

It might be also useful to limit the number of compounds to the most tested drugs or, alternatively, to use only cancer cell lines that have a certain minimum number of tested drugs. As mentioned in the previous section, if a similar cancer cell line lacks many IC<sub>50</sub> this may negatively impact the evaluation results. Surely, the imposition of such

thresholds will result in the decrease of the number of cancer cell lines available to test the framework.

At last, the implemented pipeline at Stage 2 (cancer drug recommendation) is only one among others that could be implemented. Hence, there is space for exploring other recommender systems techniques and assess if they can be a better fit.

## **6.4 Final considerations**

Recommender systems will inevitably push their boundaries beyond e-commerce applications. Throughout this project we show their potential in what concerns precision medicine and, more specifically, for personalized cancer drug recommendations within a clinical or laboratorial (pharmaceutical) context. Furthermore, not only the strengths but also the implementation challenges of such systems are highlighted so that they can be properly addressed in the future.

Domain knowledge from Genomics along with recommender systems and signal processing theory were also provided to enhance the understanding of the implementation details of this work.

There is still a long way to go, from shaping robust cancer cell line databases and drug-responses matrices to gaining the stakeholders trust towards such ML personalized systems. Nevertheless, the journey has started and all of us, from researchers to patients, can take part in it.

## REFERENCES

- Addison, P. S. (2017). *The Illustrated Wavelet Transform Handbook: Introductory Theory and Applications in Science, Engineering, Medicine and Finance* (2nd ed.). <https://doi.org/10.1201/9781420033397>
- Aggarwal, C. C. (2016). *Recommender Systems - The Textbook*. <https://doi.org/10.1007/978-3-319-29659-3>
- ArrayExpress. (2020). E-MTAB-3610 - Transcriptional Profiling of 1,000 human cancer cell lines. Retrieved March 10, 2020, from <https://www.ebi.ac.uk/arrayexpress/experiments/E-MTAB-3610/files/raw/>
- Brandão, L., Belfo, F. P., & Silva, A. (2020). Wavelet-based cancer drug recommendation system. *CENTERIS - International Conference on ENTERprise Information Systems*, (in press). Vilamoura, Portugal - October 21-23 2020: Elsevier.
- Carter, S. M., Rogers, W., Than, K., Frazer, H., Richards, B., & Houssami, N. (2020). The ethical , legal and social implications of using arti fi cial intelligence systems in breast cancer care. *The Breast*, 49, 25–32. <https://doi.org/10.1016/j.breast.2019.10.001>
- Cios, K. J., & Moore, G. W. (2002). Uniqueness of medical data mining. *Artificial Intelligence in Medicine*, 26(1–2), 1–24. [https://doi.org/10.1016/S0933-3657\(02\)00049-0](https://doi.org/10.1016/S0933-3657(02)00049-0)
- Costello, J. C., Heiser, L. M., Georgii, E., Gönen, M., Menden, M. P., Wang, N. J., ... Stolovitzky, G. (2014). A community effort to assess and improve drug sensitivity prediction algorithms. *Nature Biotechnology*, 32(12), 202–212. <https://doi.org/10.1038/nbt.2877>
- Cui, G., Wong, M. L., & Lui, H. K. (2006). Machine learning for direct marketing response models: Bayesian networks with evolutionary programming. *Management Science*, 52(4), 597–612. <https://doi.org/10.1287/mnsc.1060.0514>
- F. Ricci, L. Rokach, B. Shapira, P. B. K. (2011). *Recommender Systems Handbook*. Springer, Boston, MA.
- Falk, K. (2019). *Pratical Recommender Systems* (1st ed.). Manning.
- Gan, Z., Zou, F., Zeng, N., Xiong, B., Liao, L., Li, H. A. N., ... Du, M. I. N. (2019).



- Wavelet Denoising Algorithm Based on NDOA Compressed Sensing for Fluorescence Image of Microarray. *IEEE Access*, 7, 13338–13346. <https://doi.org/10.1109/ACCESS.2019.2891759>
- Genomics of Drug Sensitivity in Cancer. (2020). Database. Retrieved May 12, 2020, from Release 8.2 (Feb 2020) website: <https://www.cancerrxgene.org/>
- Google. (2020). Google Colaboratory. Retrieved January 28, 2020, from <https://colab.research.google.com/>
- He, X., Folkman, L., & Borgwardt, K. (2018). Kernelized rank learning for personalized drug recommendation. *Bioinformatics*, 34(16), 2808–2816. <https://doi.org/10.1093/bioinformatics/bty132>
- IMDb. (2020). IMDb. Retrieved June 1, 2020, from [www.imdb.com](http://www.imdb.com)
- Information Resources Management Association. (2019). *Data Analytics in Medicine: Concepts, Methodologies, Tools, and Applications*. IGI Global.
- Iorio, F., Knijnenburg, T. A., Vis, D. J., Saez-rodriguez, J., Mcdermott, U., & Garnett, M. J. (2016). A Landscape of Pharmacogenomic Interactions in Cancer. *Cell*, 166(3), 740–754. <https://doi.org/10.1016/j.cell.2016.06.017>
- Jensen, K. (2012). CRISP-DM process diagram. Retrieved June 12, 2020, from Wikimedia Commons website: <https://commons.wikimedia.org/w/index.php?curid=24930610>
- Li, S., Liao, C., & Kwok, J. T. (2006). Wavelet-Based Feature Extraction for Microarray Data Classification. *The 2006 IEEE International Joint Conference on Neural Network Proceedings*. <https://doi.org/10.1109/IJCNN.2006.247208>
- Light, H., Hihara, Y., Kamei, A., Kanehisa, M., Kaplan, A., & Ikeuchi, M. (2001). DNA Microarray Analysis of Cyanobacterial Gene Expression during DNA Microarray Analysis of Cyanobacterial Gene Expression during Acclimation to High Light. *The Plant Cell (American Society of Plants Physiologists)*, 13(April), 793–806. <https://doi.org/10.2307/3871341>
- Liu, Y., & Bai, L. (2009). Find Significant Gene Information Based on Changing Points of Microarray Data. *IEEE Transactions on Bio-Medical Engineering*, 56(4), 1108–1116. <https://doi.org/10.1109/TBME.2008.2009543>

- Loureiro, A., Lourenço, J., Costa, E., & Belfo, F. (2014). Indução de Árvores de Decisão na Descoberta de Conhecimento: Caso de Empresa de Organização de Eventos. *VI Congresso Internacional de Casos Docentes Em Marketing Público e Não Lucrativo*. Coimbra, Portugal - December 19 2014.
- Menden, M. P., Iorio, F., Garnett, M., Mcdermott, U., & Benes, C. H. (2013). Machine Learning Prediction of Cancer Cell Sensitivity to Drugs Based on Genomic and Chemical Properties. *PLOS ONE*, 8(4), 61318. <https://doi.org/10.1371/journal.pone.0061318>
- Nanni, L., & Lumini, A. (2011). Wavelet selection for disease classification by DNA microarray data. *Expert Systems with Applications*, 38(1), 990–995. <https://doi.org/10.1016/j.eswa.2010.07.104>
- National Center for Biotechnology Information. (2020). Gene Expression. Retrieved May 12, 2020, from <https://www.ncbi.nlm.nih.gov/probe/docs/applexpression/>
- Pimenta, C., Ribeiro, R., Sá, V., & Belfo, F. P. (2018). Fatores que Influenciam o Sucesso Escolar das Licenciaturas numa Instituição de Ensino Superior Portuguesa. *18a Conferência Da Associação Portuguesa de Sistemas de Informação (CAPSI 2018)*. Santarém, Portugal - 12-13 October 2018: Associação Portuguesa de Sistemas de Informação.
- Pittner, S., & Kamarthi, S. V. (1999). Feature extraction from wavelet coefficients for pattern recognition tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(1), 83–88. <https://doi.org/10.1109/34.745739>
- Rasheed, A., Younis, M. S., & Bilal, M. (2020). Classification of Chest Diseases using Wavelet Transforms and Transfer Learning. *International Conference on Medical Imaging and Computer-Aided Diagnosis 2020*, 158–165. [https://doi.org/https://doi.org/10.1007/978-981-15-5199-4\\_16](https://doi.org/https://doi.org/10.1007/978-981-15-5199-4_16)
- Resnick, P., Bergstrom, P., & Riedl, J. (1994). GroupLens : An Open Architecture for Collaborative Filtering of Netnews. *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, 175–186. Retrieved from <http://ccs.mit.edu/papers/CCSWP165.html>
- Seiça, A., Trigo, A., & Belfo, F. P. (2019). LexiNB - Uma Abordagem Bietápica de Classificação de Sentimentos em Tweets Relacionados com as Autoridades Fiscais

- Portuguesas. *Proceedings of The 19.ª Conferência Da Associação Portuguesa de Sistemas de Informação (CAPSI'2019) Paper 5*. Lisboa, Portugal - 11-12 October 2019.
- Sereday, S., & Cui, J. (2017). Using machine learning to predict future tv ratings. *Nielsen Journal of Measurement*, 1(3), 3–12. Retrieved from <https://www.nielsen.com/wp-content/uploads/sites/3/2019/04/using-machine-learning-to-predict-future-tv-ratings.pdf>
- Serra, J., & Angulo, J. (2003). Automatic analysis of DNA microarray images using mathematical morphology. *Bioinformatics*, 19(5), 553–562. <https://doi.org/10.1093/bioinformatics/btg057>
- Suphavitai, C., Bertrand, D., & Nagarajan, N. (2018). Predicting Cancer Drug Response using a Recommender System. *Bioinformatics*, 34(22), 3907–3914. <https://doi.org/10.1093/bioinformatics/bty452>
- Tulane University - School of Medicine. (2020). Basic Principles of Pharmacology. Retrieved May 12, 2020, from [http://tmedweb.tulane.edu/pharmwiki/doku.php/basic\\_principles\\_of\\_pharm](http://tmedweb.tulane.edu/pharmwiki/doku.php/basic_principles_of_pharm)
- Wan, J., & Zhou, S. (2010). Features extraction based on wavelet packet transform for B-mode ultrasound liver images. *2010 3rd International Congress on Image and Signal Processing*, 2, 949–955. <https://doi.org/10.1109/CISP.2010.5646917>
- Wang, L., Li, X., Zhang, L., & Gao, Q. (2017). Improved anticancer drug response prediction in cell lines using matrix factorization with similarity regularization. *BMC Cancer*, 17(Article Number 513). <https://doi.org/10.1186/s12885-017-3500-5>
- Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4), 600–612. <https://doi.org/10.1109/TIP.2003.819861>
- World Health Organization. (2018). cancer key facts. Retrieved June 12, 2020, from <https://www.who.int/news-room/fact-sheets/detail/cancer>
- World Health Organization. (2020). definitions of genetics and genomics. Retrieved May 12, 2020, from <https://www.who.int/genomics/geneticsVSgenomics/en/>
- Yang, W., Soares, J., Greninger, P., Edelman, E. J., Lightfoot, H., Forbes, S., ... Garnett, M. J. (2013). Genomics of Drug Sensitivity in Cancer ( GDSC ): a resource for

therapeutic biomarker discovery in cancer cells. *Nucleic Acids Research*, 41(D1), D955–D961. <https://doi.org/10.1093/nar/gks1111>

## **APPENDIXES**

## **APPENDIX 1. EXPERIMENT 1 / PART 1 – PYTHON CODE.**

### **Contents**

#### **1 INTRODUCTION**

##### **1.1 Google Drive Access**

##### **1.2 Packages**

#### **2 USERS SIMILARITY MEASUREMENT**

# 1 INTRODUCTION

## 1.1 Google Drive Access

In [0]:

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

Go to this URL in a browser: [https://accounts.google.com/o/oauth2/auth?client\\_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect\\_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aob&response\\_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly](https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly)

Enter your authorization code:

.....

Mounted at /content/drive

## 1.2 Packages

In [0]:

```
1 pip install biopython
```

Collecting biopython

Downloading [https://files.pythonhosted.org/packages/83/3d/e0c8a993dbea1136be90c31345aefc5babdd5046cd52f81c18fc3fdad865/biopython-1.76-cp36-cp36m-manylinux1\\_x86\\_64.whl](https://files.pythonhosted.org/packages/83/3d/e0c8a993dbea1136be90c31345aefc5babdd5046cd52f81c18fc3fdad865/biopython-1.76-cp36-cp36m-manylinux1_x86_64.whl) (https://files.pythonhosted.org/packages/83/3d/e0c8a993dbea1136be90c31345aefc5babdd5046cd52f81c18fc3fdad865/biopython-1.76-cp36-cp36m-manylinux1\_x86\_64.whl) (2.3MB)

2.3MB 5.1MB/s

Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from biopython) (1.18.2)

Installing collected packages: biopython

Successfully installed biopython-1.76

In [0]:

```
1 pip install scikit-image
```

```
Requirement already satisfied: scikit-image in /usr/local/lib/python3.6/dist-packages (0.16.2)
Requirement already satisfied: scipy>=0.19.0 in /usr/local/lib/python3.6/dist-packages (from scikit-image) (1.4.1)
Requirement already satisfied: pillow>=4.3.0 in /usr/local/lib/python3.6/dist-packages (from scikit-image) (7.0.0)
Requirement already satisfied: PyWavelets>=0.4.0 in /usr/local/lib/python3.6/dist-packages (from scikit-image) (1.1.1)
Requirement already satisfied: imageio>=2.3.0 in /usr/local/lib/python3.6/dist-packages (from scikit-image) (2.4.1)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.0 in /usr/local/lib/python3.6/dist-packages (from scikit-image) (3.2.0)
Requirement already satisfied: networkx>=2.0 in /usr/local/lib/python3.6/dist-packages (from scikit-image) (2.4)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.6/dist-packages (from scipy>=0.19.0->scikit-image) (1.18.1)
Requirement already satisfied: pyparsing!=2.0.4,!2.1.2,!2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib!=3.0.0,>=2.0.0->scikit-image) (2.4.6)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib!=3.0.0,>=2.0.0->scikit-image) (1.1.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib!=3.0.0,>=2.0.0->scikit-image) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib!=3.0.0,>=2.0.0->scikit-image) (2.8.1)
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.6/dist-packages (from networkx>=2.0->scikit-image) (4.4.2)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from kiwisolver>=1.0.1->matplotlib!=3.0.0,>=2.0.0->scikit-image) (45.2.0)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from cycler>=0.10->matplotlib!=3.0.0,>=2.0.0->scikit-image) (1.12.0)
```

In [0]:

```
1 from Bio.Affy import CelFile #package biopython (.cel files)
2 from skimage.metrics import structural_similarity as ssim #structural similarity index
3 import cv2 #handle images (read, save)
4 import pandas as pd
5 import numpy as np
6 import matplotlib.pyplot as plt
7 %matplotlib inline
8 import os
9 from os import listdir #handle path to directories/files
10
```

## 2 USERS SIMILARITY MEASUREMENT

Customized function that a) computes the SSIM between a given image and the other ones in the database and b) outputs a top-6 rank (from most similar to dissimilar)



In [0]:

```
1 #import .csv file that contains: (928) cell lines names | assays names | assays filenames
2 assaysDescript_cellLinesDrugScreen_pd = pd.read_csv('/content/drive/My Drive/RecSys_Co
```

In [0]:

```
1 assaysDescript_cellLinesDrugScreen_pd.head()
```

Out[6]:

	Characteristics[cell line]	Assay Name	Array Data File
0	UACC-812	5500994173212120213068_A01	5500994173212120213068_A01.cel
1	201T	5500994158987071513209_A01	5500994158987071513209_A01.cel
2	EVSA-T	5500994172383112813929_A01	5500994172383112813929_A01.cel
3	KYSE-520	5500994158987071513202_A01	5500994158987071513202_A01.cel
4	MS751	5500994158987071513207_A01	5500994158987071513207_A01.cel

In [0]:

```
1 #remove corrupted .CEL file - 5500994157493061613625_A01.cel
2 corrupted_file = assaysDescript_cellLinesDrugScreen_pd[assaysDescript_cellLinesDrugScreen_pd['Array Data File'] == '5500994157493061613625_A01.cel']
3 corrupted_file
```

Out[7]:

	Characteristics[cell line]	Assay Name	Array Data File
927	CAL-120	5500994157493061613625_A01	5500994157493061613625_A01.cel

In [0]:

```
1 assaysDescript_cellLinesDrugScreen_pd = assaysDescript_cellLinesDrugScreen_pd.drop(index=corrupted_file.index)
2 assaysDescript_cellLinesDrugScreen_pd.shape
```

Out[8]:

(927, 3)

In [0]:

```
1 #store the filenames in a list
2 filenames_927 = assaysDescript_cellLinesDrugScreen_pd['Array Data File'].to_list()
3 print(len(filenames_927))
4 print(filenames_927[0]) #1st element of the list
5 print(filenames_927[926]) #last element of the list
```

927  
5500994173212120213068\_A01.cel  
5500994175999120813240\_E04.cel

In [0]:

```

1 #find the folder where a .CEL file is located and add that info into the corresponding
2 #1)add column "folder" (its values will be temporarily filled with random numbers)
3 sLength = len(assaysDescript_cellLinesDrugScreen_pd['Array Data File'])
4 assaysDescript_cellLinesDrugScreen_pd = assaysDescript_cellLinesDrugScreen_pd.assign(f
5 assaysDescript_cellLinesDrugScreen_pd.head(2)

```

Out[10]:

	Characteristics[cell line]	Assay Name	Array Data File	folder
0	UACC-812	5500994173212120213068_A01	5500994173212120213068_A01.cel	-0.637904
1	201T	5500994158987071513209_A01	5500994158987071513209_A01.cel	1.293723

In [0]:

```

1 #2)find the folder and add that info into the column 'folder'
2 raw_folders = ['raw1','raw2', 'raw3', 'raw4', 'raw5', 'raw6', 'raw7', 'raw8', 'raw9',
3
4
5 for file in filenames_927:
6
7     for raw in raw_folders:
8         raw_path = os.path.join('/content/drive/My Drive/RecSys_Code/dataset/CellLines/Gen
9         raw_filenames = os.listdir(raw_path) #List all files in the folder
10
11
12         file_intersection_folder = list(set([file]) & set(raw_filenames)) #check if file i
13
14
15         if len(file_intersection_folder) != 0: #if the file is stored in the folder
16             file_row = assaysDescript_cellLinesDrugScreen_pd[assaysDescript_cellLinesDrugSc
17             row_index = file_row.index.values.astype(int)[0] #index of the row where the file
18             assaysDescript_cellLinesDrugScreen_pd.loc[row_index, 'folder'] = raw
19

```

In [0]:

```
1 assaysDescript_cellLinesDrugScreen_pd.iloc[777]
```

Out[12]:

Characteristics[cell line]	JHOS-2
Assay Name	5500994172948120113978_H07
Array Data File	5500994172948120113978_H07.cel
folder	raw21
Name: 777, dtype: object	

In [0]:

```
1 assaysDescript_cellLinesDrugScreen_pd['folder'].value_counts()
```

Out[13]:

```
raw16    41
raw10    40
raw12    40
raw13    40
raw8     39
raw23    39
raw3     39
raw15    38
raw17    38
raw24    38
raw14    38
raw4     38
raw7     37
raw20    37
raw6     37
raw22    37
raw2     37
raw11    37
raw19    37
raw5     36
raw9     36
raw1     36
raw18    35
raw21    34
raw25    23
Name: folder, dtype: int64
```

In [0]:

```
1 assaysDescript_cellLinesDrugScreen_pd['folder'].value_counts().sum() #927. OK, as expected
```

Out[14]:

```
927
```

In [0]:

```
1 #export final dataframe as .csv file (to reuse it in the notebooks STAGE2_EXP1.ipynb and STAGE2_EXP2.ipynb)
2 #.csv file contains: (927) cell lines names | assays names | assays filenames | folder names
3 assaysDescript_cellLinesDrugScreen_pd.to_csv(r"/content/drive/My Drive/RecSys_Code/data/assaysDescript_cellLinesDrugScreen_pd.csv")
```

In [0]:

```

1 def similarity_ssim(image):
2
3     """
4     Description:
5     This function takes an image (.cel) and computes its similarity, using the Structural
6     Similarity Index, regarding the other images (.cel) from the database.
7     It outputs the top-6 most similar images.
8
9     Parameter:
10    image - image (.cel) whose similarity to the other images (.cel) will be computed
11
12
13    Returns:
14    top-6 most similar images
15
16    Example:
17    >>> similarity_ssim('5500994157493061613625_A10.cel')
18
19    """
20
21    #import .csv file that contains: (927) cell lines names | assays names | assays files
22    assaysDescript_celllinesDrugScreen_pd = pd.read_csv('/content/drive/My Drive/RecSys_
23
24    #store the assays filenames into a list
25    assays_filenames_927_pd = assaysDescript_celllinesDrugScreen_pd['Array Data File']
26    assays_filenames_927_list = list(assays_filenames_927_pd)
27    assays_filenames_927_set = set(assays_filenames_927_list)
28
29    raw_folders = ['raw1', 'raw2', 'raw3', 'raw4', 'raw5', 'raw6', 'raw7', 'raw8', 'raw9']
30
31    #create an empty dataframe to store the results
32    columns_name = ["image", "SSIM"]
33    df = pd.DataFrame(columns = columns_name)
34    n=0
35
36    #Locate and read the given image:
37    #1)define the path to the directory where the image is stored
38    image_row = assaysDescript_celllinesDrugScreen_pd[assaysDescript_celllinesDrugScreen_
39    image_index = image_row.index.values.astype(int)[0] #index of the row where the image
40    raw = assaysDescript_celllinesDrugScreen_pd.loc[image_index].at['folder'] #folder w
41    image_path = os.path.join('/content/drive/My Drive/RecSys_Code/dataset/Celllines/Gene
42    images_path = '/content/drive/My Drive/RecSys_Code/dataset/Celllines/GeneExpressionP
43    images_filenames = os.listdir(images_path) #Lists all the files in the directory "4t
44
45    #2)read the image
46    with open(image_path) as handle:
47        c = CelFile.read(handle) #read CEL file
48    img = c.intensities #read the image given as parameter
49
50    #folder by folder: Locate other images, read them and compute SSIM; store the result
51    for raw in raw_folders:
52        print("The analysis of folder:", raw, "has started.")
53        images_path = os.path.join('/content/drive/My Drive/RecSys_Code/dataset/Celllines/
54        all_images_filenames_list = os.listdir(images_path) #Lists all the files in the fo
55        all_images_filenames_set = set(all_images_filenames_list)
56        #retrieve the filenames of "raw" that are also present in assays_filenames_927
57        #(remember that, although we have an Affymetrix database with 1018 DNA microarray
58        filtered_images_filenames = assays_filenames_927_set.intersection(all_images_filen
59        filtered_images_filenames = list(filtered_images_filenames)

```

```
60
61
62     for image_db in filtered_images_filenames:
63         img_database_path = os.path.join('/content/drive/My Drive/RecSys_Code/dataset/Ce
64         with open(img_database_path) as handle:
65             c = CelFile.read(handle) #read CEL file
66             img_database = c.intensities #read the image (in the folder 'raw')
67
68         img_ssim = ssim(img, img_database, multichannel = True) #compute SSIM
69
70
71         df.loc[n] = [image_db] + [img_ssim] #add result to the dataframe
72         n += 1
73
74     df = df.sort_values(by='SSIM', axis=0, ascending=False).head(6)
75     print()
76     print("Top-6 most similar images:")
77     print()
78     return df
79
```

In [0]:

```
1 similarity_ssim('5500994157493061613625_A10.cel')
```

The analysis of folder: raw1 has started.  
 The analysis of folder: raw2 has started.  
 The analysis of folder: raw3 has started.  
 The analysis of folder: raw4 has started.  
 The analysis of folder: raw5 has started.  
 The analysis of folder: raw6 has started.  
 The analysis of folder: raw7 has started.  
 The analysis of folder: raw8 has started.  
 The analysis of folder: raw9 has started.  
 The analysis of folder: raw10 has started.  
 The analysis of folder: raw11 has started.  
 The analysis of folder: raw12 has started.  
 The analysis of folder: raw13 has started.  
 The analysis of folder: raw14 has started.  
 The analysis of folder: raw15 has started.  
 The analysis of folder: raw16 has started.  
 The analysis of folder: raw17 has started.  
 The analysis of folder: raw18 has started.  
 The analysis of folder: raw19 has started.  
 The analysis of folder: raw20 has started.  
 The analysis of folder: raw21 has started.  
 The analysis of folder: raw22 has started.  
 The analysis of folder: raw23 has started.  
 The analysis of folder: raw24 has started.  
 The analysis of folder: raw25 has started.

Top-6 most similar images:

Out[24]:

	image	SSIM
101	5500994157493061613625_A10.cel	1.000000
246	5500994158987071513207_C06.cel	0.731897
554	5500994157493061613625_F04.cel	0.727921
359	5500994158987071513209_D06.cel	0.715566
74	5500994172383112813929_A11.cel	0.714804
697	5500994158987071513201_G10.cel	0.711776

In [0]:

```
1 #the execution of the function similarity_ssim() takes, aprox., 25 minutes
```



In [0]:

```
1 #test
2 image1_path = '/content/drive/My Drive/RecSys_Code/dataset/CellLines/GeneExpressionPro
3 with open(image1_path) as handle:
4     c1 = CelFile.read(handle)
5 image1 = c1.intensities
6
7 image2_path = '/content/drive/My Drive/RecSys_Code/dataset/CellLines/GeneExpressionPro
8 with open(image2_path) as handle:
9     c2 = CelFile.read(handle)
10 image2 = c2.intensities
11
12
13 ssim_test = ssim(image1, image2, multichannel = True)
14 ssim_test #0.731897 Ok, as expected.
```

Out[143]:

0.7318971581690118

## APPENDIX 2. EXPERIMENT 1 / PART 2 – PYTHON CODE.

### Contents

#### 1 INTRODUCTION

##### 1.1 Google Drive Access

##### 1.2 Packages

##### 1.3 Customized function *similarity\_ssim(image.cel)*

##### 1.4 Dataset split (for performance evaluation)

#### 2 CANCER DRUG RECOMMENDATION

##### 2.1 Drug response matrix

###### 2.1.1 Download original dataset

*GDSC1\_fitted\_dose\_response\_25Feb20.xlsx*

###### 2.1.2 (Brief) Exploratory analysis original dataset

###### 2.1.3 Filter the original dataset (from 928 to 927 cell lines)

###### 2.1.4 Final matrix rows/927 cell lines X columns/345 drugs

###### 2.1.4.1 One cell line – “LS-1034”

###### 2.1.4.2 927 cell lines

###### 2.1.5 Export drug\_response\_matrix\_ln as .csv file

##### 2.2 IC50

###### 2.2.1 From LN IC50 to IC50

###### 2.2. Normalize IC50

###### 2.2.3 Set missing IC50 values equal to 0.5

###### 2.2.4 ::Export drug\_response\_matrix\_norm\_withoutMissingValues as .csv file

##### 2.3 Top-N similar cell lines

###### 2.3.1 Folder B

###### 2.3.2 Folder C



### 2.3.3 Folder D

## 2.4 Drug candidates generation

### 2.4.1 Folder B

### 2.4.2 Folder C

### 2.4.3 Folder D

## 2.5 Drug candidates score

### 2.5.1 Folder B

### 2.5.2 Folder C

### 2.5.3 Folder D

## 2.6 Top-N recommendation list

### 2.6.1 Folder B

### 2.6.2 Folder C

### 2.6.3 Folder D

## 2.7 Evaluation – Top-N hit-rate & average reciprocal hit-rate

### 2.7.1 Folder B

#### 2.7.1.1. Top-N hit-rate

#### 2.7.1.2 Average reciprocal hit-rate

### 2.7.2 Folder C

#### 2.7.2.1. Top-N hit-rate

#### 2.7.2.2 Average reciprocal hit-rate

### 2.7.3 Folder D

#### 2.7.3.1. Top-N hit-rate

#### 2.7.3.2 Average reciprocal hit-rate

#### 2.7.4 Overall / Final results

# 1 INTRODUCTION

## 1.1 Google Drive Access

In [0]:

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

Go to this URL in a browser: [https://accounts.google.com/o/oauth2/auth?client\\_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect\\_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aob&response\\_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly](https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly)

Enter your authorization code:

.....

Mounted at /content/drive

## 1.2 Packages

In [0]:

```
1 pip install biopython
```

Collecting biopython

Downloading [https://files.pythonhosted.org/packages/a8/66/134dbd5f885fc71493c61b6cf04c9ea08082da28da5ed07709b02857cbd0/biopython-1.77-cp36-cp36m-manylinux1\\_x86\\_64.whl](https://files.pythonhosted.org/packages/a8/66/134dbd5f885fc71493c61b6cf04c9ea08082da28da5ed07709b02857cbd0/biopython-1.77-cp36-cp36m-manylinux1_x86_64.whl) (https://files.pythonhosted.org/packages/a8/66/134dbd5f885fc71493c61b6cf04c9ea08082da28da5ed07709b02857cbd0/biopython-1.77-cp36-cp36m-manylinux1\_x86\_64.whl) (2.3MB)

2.3MB 2.8MB/s

Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from biopython) (1.18.4)

Installing collected packages: biopython

Successfully installed biopython-1.77

In [0]:

```
1 import pandas as pd
2 import numpy as np
3 from skimage.metrics import structural_similarity as ssim #structural similarity index
4 from sklearn.utils import shuffle #shuffle (i.e., make random order)
5 from Bio.Affy import Celfile #package biopython (.cel files)
6 import os
7 from os import listdir #handle path to directories/files
```

### **1.3 Customized function *similarity\_ssim(image.cel)***

In [0]:

```

1 def similarity_ssim(image):
2
3     """
4     Description:
5     This function takes an image (.cel) and computes its similarity, using the Structural
6     Similarity Index, regarding the other images (.cel) from the database.
7     It outputs the top-50 most similar and the top-10 most dissimilar images.
8
9     Parameter:
10    image - image (.cel) whose similarity to the other images (.cel) will be computed
11
12
13    Returns:
14    top-50 most similar images
15    top-10 most dissimilar images
16
17    Example:
18    >>> similarity_ssim('5500994157493061613625_A10.cel')
19
20
21    """
22    #import .csv file that contains: (927) cell lines names | assays names | assays file
23    assaysDescript_cellLinesDrugScreen_pd = pd.read_csv('/content/drive/My Drive/RecSys_
24
25    #store the assays filenames into a list
26    assays_filenames_927_pd = assaysDescript_cellLinesDrugScreen_pd['Array Data File']
27    assays_filenames_927_list = list(assays_filenames_927_pd)
28    assays_filenames_927_set = set(assays_filenames_927_list) #transform the list object
29
30    raw_folders = ['raw1', 'raw2', 'raw3', 'raw4', 'raw5', 'raw6', 'raw7', 'raw8', 'raw9'
31
32    #create an empty dataframe to store the results
33    columns_name = ["image", "SSIM"]
34    df = pd.DataFrame(columns = columns_name)
35    n=0
36
37    #Locate and read the given image:
38    #1)define the path to the directory where the image is stored
39    image_row = assaysDescript_cellLinesDrugScreen_pd[assaysDescript_cellLinesDrugScreen
40    image_index = image_row.index.values.astype(int)[0] #index of the row where the image
41    raw = assaysDescript_cellLinesDrugScreen_pd.loc[image_index].at['folder'] #folder w
42    image_path = os.path.join('/content/drive/My Drive/RecSys_Code/dataset/CellLines/Gene
43    images_path = '/content/drive/My Drive/RecSys_Code/dataset/CellLines/GeneExpressionPi
44    images_filenames = os.listdir(images_path) #lists all the files in the directory "Gene
45
46    #2)read the image
47    with open(image_path) as handle:
48        c = CelFile.read(handle) #read CEL file
49        img = c.intensities #read the image given as parameter
50
51    #folder by folder: locate other images, read them and compute SSIM; store the result
52    for raw in raw_folders:
53        if raw == 'raw1' or raw == 'raw5' or raw == 'raw10' or raw == 'raw15' or raw == 'r
54            print("Current folder under analysis:", raw)
55            images_path = os.path.join('/content/drive/My Drive/RecSys_Code/dataset/CellLines/
56            all_images_filenames_list = os.listdir(images_path) #lists all the files in the fo
57            all_images_filenames_set = set(all_images_filenames_list)
58            #retrieve the filenames of "raw" that are also present at assays_filenames_927
59            #(remember that, although we have an Affymetrix database with 1018 DNA microarray

```

```

60 filtered_images_filenames = assays_filenames_927_set.intersection(all_images_filenames)
61 filtered_images_filenames = list(filtered_images_filenames)
62
63 for image_db in filtered_images_filenames:
64     img_database_path = os.path.join('/content/drive/My Drive/RecSys_Code/dataset/Ce
65     with open(img_database_path) as handle:
66         c = CelFile.read(handle) #read CEL file
67         img_database = c.intensities #read the image (in the folder 'raw')
68
69
70     img_ssim = ssim(img, img_database, multichannel = True) #compute SSIM
71
72
73     df.loc[n] = [image_db] + [img_ssim] #add result to the dataframe
74     n += 1
75
76 df_similar = df.sort_values(by='SSIM', axis=0, ascending=False).head(50)
77 df_dissimilar = df.sort_values(by='SSIM', axis=0, ascending=False).tail(10)
78 print()
79
80 return df_similar, df_dissimilar

```

## 1.4 Dataset split (for performance evaluation)

Function `similarity_ssim(image.cel)` takes aprox. 25 minutes to run (with Google Colab computational resources). Thus, to compute a user-user similarity matrix (which is symmetric along its diagonal) would take aprox.  $827 \times 827 / 2 \times 25$  minutes.

Hence, due to the previous constraint, we use an *adaptation* of a four-fold cross-validation to evaluate the proposed RecSys.

The 927 cell lines are split into 4 folders (mutually exclusive but of different sizes):

- Folder A with 852 cells
- Folder B with 25 cells
- Folder C with 25 cells
- Folder D with 25 cells

Using folders B, C and D, we perform 3 distinct validations and take their average as the final result.

In [0]:

```

1 #import .csv file that contains: (927) cell lines names | assays names | assays filenames
2 cells927_pd = pd.read_csv('/content/drive/My Drive/RecSys_Code/dataset/CellLines/927Cells.csv')
3 cells927_pd.head(2)

```

Out[5]:

	Characteristics[cell line]	Assay Name	Array Data File	folder
0	UACC-812	5500994173212120213068_A01	5500994173212120213068_A01.cel	raw1
1	201T	5500994158987071513209_A01	5500994158987071513209_A01.cel	raw1

In [0]:

```
1 cells927_pd[cells927_pd['Array Data File'] == '5500994172383112813928_C04.cel']
```

Out[6]:

Characteristics[cell line]	Assay Name	Array Data File	folder
----------------------------	------------	-----------------	--------

In [0]:

```
1 cells927 = cells927_pd['Array Data File'].values #store the filenames into an array
2 cells927[0:10]
```

Out[7]:

```
array(['5500994173212120213068_A01.cel', '5500994158987071513209_A01.cel',
      '5500994172383112813929_A01.cel', '5500994158987071513202_A01.cel',
      '5500994158987071513207_A01.cel', '5500994172383112813928_A01.cel',
      '5500994172383112813930_A01.cel', '5500994172948120113978_A01.cel',
      '5500994173212120213068_A02.cel', '5500994157493061613625_A02.cel'],
      dtype=object)
```

In [0]:

```
1 #split
2 cells927 = shuffle (cells927, random_state = 101) #shuffle the order of the cell lines
3 folderA = cells927[0:752] #752 cells
4 folderB = cells927[752:777] #25 cells
5 folderC = cells927[777:802] #25 cells
6 folderD = cells927[802:827] #25 cells
7 #missing_100_cells = [827:927] #100 cells (without impact in the notebooks or results,
8
9 #752 + 25 + 25 + 25 + 100 = 927 cells
```

In [0]:

```
1 #store the folders in dataframes and export them as .csv files (to reuse them in the ne
2 folderA_pd = pd.DataFrame(folderA)
3 folderA_pd.to_csv(r"/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderA/folderA.csv")
4
5 folderB_pd = pd.DataFrame(folderB)
6 folderB_pd.to_csv(r"/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderB/folderB.csv")
7
8 folderC_pd = pd.DataFrame(folderC)
9 folderC_pd.to_csv(r"/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderC/folderC.csv")
10
11 folderD_pd = pd.DataFrame(folderD)
12 folderD_pd.to_csv(r"/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderD/folderD.csv")
```

## 2 CANCER DRUG RECOMMENDATION

### 2.1 Drug response matrix

Description:



- rows: cancer cell lines
- columns: drugs
- values: IC50s

### 2.1.1 Download original dataset *GDSC1\_fitted\_dose\_response\_25Feb20.xlsx*

In [0]:

```
1 cell_drug_fitResponse = pd.read_excel("/content/drive/My Drive/RecSys_Code/dataset/Drug
2 cell_drug_fitResponse.head(2)
```

Out[6]:

	CELL_LINE_NAME	DRUG_NAME	LN_IC50
0	MC-CAR	Erlotinib	2.395685
1	ES3	Erlotinib	3.140923

In [0]:

```
1 cell_drug_fitResponse.shape
```

Out[7]:

(310904, 3)

### 2.1.2 (Brief) Exploratory analysis original dataset

In [0]:

```
1 #number of distinct cell lines
2 cells = cell_drug_fitResponse['CELL_LINE_NAME'].unique()
3 cells.shape
```

Out[10]:

(987,)

In [0]:

```
1 cells_freq = cell_drug_fitResponse['CELL_LINE_NAME'].value_counts()
2 cells_freq
```

Out[11]:

```
KCL-22      367
SH-4        367
LS-123      367
A253        367
AMO-1       367
...
ECC12       32
KP-N-RT-BM-1 25
CP67-MEL    23
NCI-H378     2
NCI-H250     1
Name: CELL_LINE_NAME, Length: 987, dtype: int64
```

In [0]:

```
1 #number of distinct drugs
2 drugs = cell_drug_fitResponse['DRUG_NAME'].unique() #retrieve drugs names
3 drugs.shape
```

Out[13]:

```
(345,)
```

In [0]:

```
1 drugs_freq = cell_drug_fitResponse['DRUG_NAME'].value_counts()
2 drugs_freq
```

Out[18]:

```
Cisplatin      1879
AZD7762        1878
SN-38          1876
PLX-4720       1869
Avagacestat    1861
...
Dasatinib      394
Tozasertib     394
Bortezomib     393
JW-7-52-1     384
Rapamycin      358
Name: DRUG_NAME, Length: 345, dtype: int64
```

### 2.1.3 Filter the original dataset (from 987 to 927 cell lines)



In [0]:

```

1 #from the total of 987 cell lines, we are only interested in 927 cell lines (the ones
2 #import .csv file that contains: (927) cell lines names | assays names | assays file names
3 cells927_pd = pd.read_csv('/content/drive/My Drive/RecSys_Code/dataset/CellLines/927CellLines.csv')
4 cells927_pd.head(2)

```

Out[28]:

	Characteristics[cell line]	Assay Name	Array Data File	folder
0	UACC-812	5500994173212120213068_A01	5500994173212120213068_A01.cel	raw1
1	201T	5500994158987071513209_A01	5500994158987071513209_A01.cel	raw1

In [0]:

```

1 cells927 = pd.DataFrame(cells927_pd['Characteristics[cell line]']).rename(columns = {'Characteristics[cell line]': 'CELL_LINE_NAME'})
2 cells927.head(2)

```

Out[29]:

	CELL_LINE_NAME
0	UACC-812
1	201T

In [0]:

```

1 #from the original dataframe cell_drug_fitResponse, keep only the results/rows related to the 927 cell lines
2 cell_drug_fitResponse_927 = pd.merge(cell_drug_fitResponse, cells927, on = 'CELL_LINE_NAME')
3 cell_drug_fitResponse_927.head(2)

```

Out[30]:

	CELL_LINE_NAME	DRUG_NAME	LN_IC50
0	MC-CAR	Erlotinib	2.395685
1	MC-CAR	Rapamycin	-0.658244

In [0]:

```
1 cell_drug_fitResponse_927.shape
```

Out[31]:

(294425, 3)

In [0]:

```

1 #check for cell lines that appear more than once
2 cell_drug_fitResponse_927['CELL_LINE_NAME'].value_counts()
3

```

Out[32]:

```

KNS-42      367
LS-1034     367
AMO-1       367
SK-MEL-1    367
TE-1        367
...
MM1S        63
EW-12       41
ECC12       32
NCI-H378     2
NCI-H250     1
Name: CELL_LINE_NAME, Length: 927, dtype: int64

```

In [0]:

```

1 #conclusion: all the cell lines that appear more than 345 (the number of distinct drugs)

```

## 2.1.4 Final matrix: rows/927 cell lines X columns/345 drugs

### 2.1.4.1 One cell line - 'LS-1034'

In [0]:

```

1 #from the filtered dataset, select the rows regarding "LS-1034" and store the result in df_LS_1034
2 df_LS_1034 = cell_drug_fitResponse_927[cell_drug_fitResponse_927['CELL_LINE_NAME'] == 'LS-1034']
3 df_LS_1034.head(3)

```

Out[34]:

	CELL_LINE_NAME	DRUG_NAME	LN_IC50
95974	LS-1034	Erlotinib	3.587308
95975	LS-1034	Rapamycin	-2.160668
95976	LS-1034	Sunitinib	4.013094

In [0]:

```
1 #find an example of repeated drug
2 drugs_LS_1034 = df_LS_1034['DRUG_NAME'].value_counts()
3 drugs_LS_1034
```

Out[35]:

```
JQ1          2
Afatinib     2
Doxorubicin  2
UNC0638      2
Selumetinib  2
..
AZD8835      1
Ruxolitinib  1
Alectinib    1
Vinorelbine  1
AZD6094      1
Name: DRUG_NAME, Length: 345, dtype: int64
```

In [0]:

```
1 # drug 'AZD4547' appears twice on the cell line 'LS-1034'
2 AZD4547 = df_LS_1034[df_LS_1034['DRUG_NAME'] == 'AZD4547']
3 AZD4547
```

Out[36]:

	CELL_LINE_NAME	DRUG_NAME	LN_IC50
96251	LS-1034	AZD4547	4.011553
96334	LS-1034	AZD4547	2.462752

In [0]:

```
1 #let's group the previous rows by their mean value
2 AZD4547 = AZD4547.groupby('DRUG_NAME').mean().reset_index()
3 AZD4547
```

Out[37]:

	DRUG_NAME	LN_IC50
0	AZD4547	3.237153

In [0]:

```
1 #replace the duplicated drugs by their mean value
2 df_LS_1034 = df_LS_1034.groupby('DRUG_NAME').mean().reset_index()
3 df_LS_1034
```

Out[38]:

	DRUG_NAME	LN_IC50
0	(5Z)-7-Oxozeaenol	1.530226
1	5-Fluorouracil	1.366590
2	A-443654	-0.406974
3	A-770041	2.177551
4	A-83-01	5.124398
...	...	...
340	ZSTK474	0.903746
341	Zibotentan	5.578559
342	eEF2K Inhibitor, A-484954	5.480570
343	kb NB 142-70	4.062590
344	rTRAIL	-1.095531

345 rows × 2 columns

In [0]:

```
1 #test
2 drug = 'AZD4547'
3
4 drug_row = df_LS_1034[df_LS_1034['DRUG_NAME'] == drug] #row of the dataframe where the
5 drug_row
6 #3.237153 - OK, as expected.
```

Out[39]:

	DRUG_NAME	LN_IC50
23	AZD4547	3.237153

1) transform the dataframe 'df\_LS\_1034' into row format

In [0]:

```
1 cell = list(['LS-1034'])
2 df_cell = pd.DataFrame(cell, columns = ['cell line'])
3 df_cell
```

Out[40]:

	cell line
0	LS-1034

In [0]:

```
1 df_drugs = df_LS_1034.transpose()
2 df_drugs
```

Out[41]:

	0	1	2	3	4	5	6	
<b>DRUG_NAME</b>	(5Z)-7-Oxozeaenol	5-Fluorouracil	A-443654	A-770041	A-83-01	ACY-1215	AGI-6780	Alt Ribonucleoti
<b>LN_IC50</b>	1.53023	1.36659	-0.406974	2.17755	5.1244	3.85712	2.16919	8.960

2 rows × 345 columns

◀ ▶

In [0]:

```
1 df_drugs = df_drugs.rename(columns = df_drugs.iloc[0]).drop(index = 'DRUG_NAME')
2 df_drugs
```

Out[42]:

	(5Z)-7-Oxozeaenol	5-Fluorouracil	A-443654	A-770041	A-83-01	ACY-1215	AGI-6780	AICA Ribonucleotide
<b>LN_IC50</b>	1.53023	1.36659	-0.406974	2.17755	5.1244	3.85712	2.16919	8.96068

1 rows × 345 columns

◀ ▶

In [0]:

```
1 df_drugs = df_drugs.reset_index(drop=True)
2 df = pd.concat([df_cell, df_drugs], axis = 1)
3 df
```

Out[43]:

	cell line	(5Z)-7-Oxozeaenol	5-Fluorouracil	A-443654	A-770041	A-83-01	ACY-1215	AGI-6780	AICA Ribonucleotide
<b>0</b>	LS-1034	1.53023	1.36659	-0.406974	2.17755	5.1244	3.85712	2.16919	8.96068

1 rows × 346 columns

◀ ▶

2) Create an empty dataframe to store the final response matrix and append the previous row

In [0]:

```

1 #empty dataframe to store the final matrix (1 cell line x 345 drugs)
2 df_drug = pd.DataFrame(columns = drugs) #empty dataframe
3 df_cell_line = pd.DataFrame(columns = ['cell line']) #empty dataframe
4 drug_response_matrix_ln = pd.concat([df_cell_line, df_drug], axis = 1) #concatenate pr
5 drug_response_matrix_ln.head(2) #305 columns = 1 cell line + 346 drugs. OK

```

Out[44]:

cell line	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG- 132	Paclitaxel	Cyclopamine	AZ628	Sorafenil
--------------	-----------	-----------	-----------	----------------	------------	------------	-------------	-------	-----------

0 rows × 346 columns



In [0]:

```

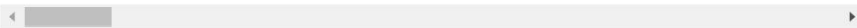
1 drug_response_matrix_ln = drug_response_matrix_ln.append(df.iloc[0])
2 drug_response_matrix_ln

```

Out[45]:

	cell line	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG-132	Paclitaxel	Cyclopamine	AZ628
0	LS-1034	3.587308	-2.160668	4.013094	3.569745	0.959943	-2.268914	5.134507	2.503155

1 rows × 346 columns



#### 2.1.4.2 927 cell lines

In [0]:

```

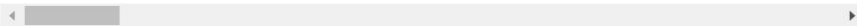
1 #empty dataframe to store the final matrix (i.e. drug response matrix with shape(927,346))
2 df_drug = pd.DataFrame(columns = drugs) #empty dataframe
3 df_cell_line = pd.DataFrame(columns = ['cell line']) #empty dataframe
4 drug_response_matrix_ln = pd.concat([df_cell_line, df_drug], axis = 1) #concatenate pr
5 drug_response_matrix_ln.head(2) #305 columns = 1 cell line + 345 drugs. OK

```

Out[46]:

cell line	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG- 132	Paclitaxel	Cyclopamine	AZ628	Sorafenil
--------------	-----------	-----------	-----------	----------------	------------	------------	-------------	-------	-----------

0 rows × 346 columns



In [0]:

```
1 cells_927 = list(cell_drug_fitResponse_927['CELL_LINE_NAME'].unique())
2 len(cells_927) #OK. 927 cell lines as expected.
```

Out[47]:

927

In [0]:

```
1 #generate "drug_response_matrix_ln" (with LN IC50 values) and export it as .csv file
2 n = 0
3
4 for cell in cells_927:
5     df_cell = cell_drug_fitResponse_927[cell_drug_fitResponse_927['CELL_LINE_NAME'] == c
6     df_cell = df_cell.groupby('DRUG_NAME').mean().reset_index() #replace the drug duplica
7                                     #after this command, df_
8
9
10
11 #1)transform df_cell into row format
12 cell_list = list([cell])
13 cell_df = pd.DataFrame(cell_list, columns = ['cell line'])
14
15 drugs_df = df_cell.transpose() #transform columns in rows
16 drugs_df = drugs_df.rename(columns = drugs_df.iloc[0]).drop(index = 'DRUG_NAME')
17 drugs_df = drugs_df.reset_index(drop=True)
18
19 df = pd.concat([cell_df, drugs_df], axis = 1)
20
21 #2)append df to the final matrix
22 drug_response_matrix_ln = drug_response_matrix_ln.append(df.iloc[0])
23
24 n += 1
25 if n==100 or n==200 or n==300 or n==400 or n==500 or n==600 or n==700 or n==800 or n
26     print("Progress report:", n, "cell lines were added to the Drug Response Matrix")
27
28 if n==927:
29     print("Task completed. Saving Drug Response Matrix into file drugResponseMatrix_ln
30     drug_response_matrix_ln.to_csv(r"/content/drive/My Drive/RecSys_Code/dataset/DrugS
31
32
33
```

```
Progress report: 100 cell lines were added to the Drug Response Matrix
Progress report: 200 cell lines were added to the Drug Response Matrix
Progress report: 300 cell lines were added to the Drug Response Matrix
Progress report: 400 cell lines were added to the Drug Response Matrix
Progress report: 500 cell lines were added to the Drug Response Matrix
Progress report: 600 cell lines were added to the Drug Response Matrix
Progress report: 700 cell lines were added to the Drug Response Matrix
Progress report: 800 cell lines were added to the Drug Response Matrix
Progress report: 900 cell lines were added to the Drug Response Matrix
Task completed. Saving Drug Response Matrix into file drugResponseMatrix_ln.
csv
```

In [0]:

1 drug\_response\_matrix\_ln.shape

Out[49]:

(927, 346)

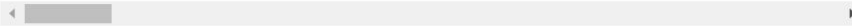
In [0]:

1 drug\_response\_matrix\_ln.head()

Out[50]:

	cell line	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG-132	Paclitaxel	Cyclopamine	AZ628
0	MC-CAR	2.395685	-0.658244	2.161095	2.613997	0.530615	-3.647573	4.296779	2.055377
0	ES3	3.140923	-0.067752	3.043634	2.935678	1.296998	-0.356496	4.887534	2.070470
0	ES5	3.968757	0.586881	3.774145	2.696152	0.930800	0.680364	5.695762	2.666071
0	ES7	2.692768	-0.025451	2.991234	2.491081	0.550690	0.075821	4.998695	2.392732
0	EW-11	2.478678	-2.123159	3.600942	2.775492	0.232020	0.920329	5.716290	1.087535

5 rows × 346 columns



In [0]:

1 drug\_response\_matrix\_ln.shape

Out[51]:

(927, 346)

In [0]:

```

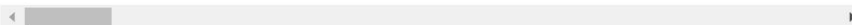
1 #check if the row regarding cell "LS-1034" matches with previous section "One cell line
2 LS_1034_row = drug_response_matrix_ln[drug_response_matrix_ln['cell line'] == 'LS-1034'
3 LS_1034_row
4 #Conclusion: OK. It matches.

```

Out[52]:

	cell line	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG-132	Paclitaxel	Cyclopamine	AZ628
0	LS-1034	3.587308	-2.160668	4.013094	3.569745	0.959943	-2.268914	5.134507	2.503155

1 rows × 346 columns





In [0]:

```
1 #check matrix sparsity (by columns)
2 drug_response_matrix_ln.isnull().sum()
```

Out[53]:

```
cell line      0
Erlotinib     551
Rapamycin     586
Sunitinib     549
PHA-665752    540
...
AZD1332       40
TTK_3146      41
SN-38         19
Pevonedistat  261
PFI-3         119
Length: 346, dtype: int64
```

In [0]:

```
1 #check matrix sparsity (total)
2 drug_response_matrix_ln.isnull().sum().sum()
3 #Conclusion: 927 X 345 = 319 815; 42 594 / 319 815 = 13,32% of missing values
4 #I.e., Among the 948 X 345 = 319 815 interacting pairs (cell-drug), 86,68% of the IC50
5 #and 13,32% of the IC50 are missing.
6
```

Out[54]:

42594

### 2.1.5 :: Export drug\_response\_matrix\_ln as .csv file

In [0]:

```
1 #export "drug_response_matrix_ln" (with LN IC50 values) as .csv file (already done in :
2 #drug_response_matrix_ln.to_csv(r"/content/drive/My Drive/RecSys_Code/dataset/DrugScre
3
```

## 2.2 IC50s

In [0]:

```
1 drug_response_matrix_ln = pd.read_csv("/content/drive/My Drive/RecSys_Code/dataset/Drug")
2 drug_response_matrix_ln.head()
```

Out[55]:

	cell line	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG-132	Paclitaxel	Cyclopamine	AZ628
0	MC-CAR	2.395685	-0.658244	2.161095	2.613997	0.530615	-3.647573	4.296779	2.055377
1	ES3	3.140923	-0.067752	3.043634	2.935678	1.296998	-0.356496	4.887534	2.070470
2	ES5	3.968757	0.586881	3.774145	2.696152	0.930800	0.680364	5.695762	2.666071
3	ES7	2.692768	-0.025451	2.991234	2.491081	0.550690	0.075821	4.998695	2.392732
4	EW-11	2.478678	-2.123159	3.600942	2.775492	0.232020	0.920329	5.716290	1.087535

5 rows × 346 columns

In [0]:

```
1 drug_response_matrix_ln.shape
```

Out[56]:

(927, 346)

We follow the method in "Machine Learning Prediction of Cancer Cell Sensitivity to Drugs Based on Genomic and Chemical Properties" Paper 5.1. to normalize the logarithmic IC50 values into a (0, 1) interval.

- 0 - most sensitive
- 1 - most resistant

For cell-drug pairs that lack IC50 values, we set their normalized IC50 to 0.5 (neutral point).

### 2.2.1 From LN IC50s to IC50s

In [0]:

```
1 drug_response_matrix_ic50 = drug_response_matrix_ln.iloc[:, 1:346].transform(func = 'e')
```

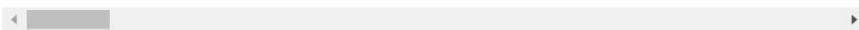
In [0]:

```
1 drug_response_matrix_ic50.head()
```

Out[58]:

	Erlotinib	Rapamycin	Sunitinib	PHA-665752	MG-132	Paclitaxel	Cyclopamine	AZ628
0	10.975714	0.517760	8.680638	13.653515	1.699977	0.026054	73.462789	7.809782
1	23.125202	0.934492	20.981351	18.834268	3.658298	0.700125	132.626114	7.928549
2	52.918712	1.798371	43.560248	14.822585	2.536538	1.974596	297.603481	14.383346
3	14.772510	0.974870	19.910237	12.074321	1.734449	1.078769	148.219606	10.943350
4	11.925488	0.119653	36.632726	16.046520	1.261145	2.510116	303.775821	2.966952

5 rows × 345 columns



## 2.2.2 Normalize IC50s

In [0]:

```
1 drug_response_matrix_norm = drug_response_matrix_ic50.transform(func = lambda x : (1/(
```

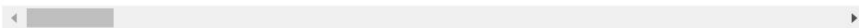
In [0]:

```
1 drug_response_matrix_norm.head()
```

Out[60]:

	Erlotinib	Rapamycin	Sunitinib	PHA-665752	MG-132	Paclitaxel	Cyclopamine	AZ628	Sorafenib
0	0.559607	0.483550	0.553818	0.564980	0.513262	0.409808	0.605797	0.551204	0.513262
1	0.577884	0.498306	0.575509	0.572869	0.532380	0.491089	0.619813	0.551578	0.513262
2	0.597937	0.514668	0.593249	0.566998	0.523253	0.517003	0.638665	0.566260	0.513262
3	0.566915	0.499364	0.574228	0.561957	0.513764	0.501896	0.622429	0.559535	0.513262
4	0.561652	0.447120	0.589063	0.568945	0.505800	0.522992	0.639139	0.527162	0.605797

5 rows × 345 columns



In [0]:

```
1 drug_response_matrix_norm.max().max() #the most resistant value
```

Out[61]:

0.7748505206703983

In [0]:

```
1 drug_response_matrix_norm.min().min() #the most sensitive value
```

Out[62]:

0.2694718762793556

### 2.2.3 Set missing IC50 values equal to 0.5

In [0]:

```
1 drug_response_matrix_norm_withoutMissingValues = drug_response_matrix_norm.fillna(0.5)
```

In [0]:

```
1 drug_response_matrix_norm_withoutMissingValues.head()
```

Out[64]:

	Erlotinib	Rapamycin	Sunitinib	PHA-665752	MG-132	Paclitaxel	Cyclopamine	AZ628	Sorafenib
0	0.559607	0.483550	0.553818	0.564980	0.513262	0.409808	0.605797	0.551204	0.511204
1	0.577884	0.498306	0.575509	0.572869	0.532380	0.491089	0.619813	0.551578	0.511578
2	0.597937	0.514668	0.593249	0.566998	0.523253	0.517003	0.638665	0.566260	0.516260
3	0.566915	0.499364	0.574228	0.561957	0.513764	0.501896	0.622429	0.559535	0.519535
4	0.561652	0.447120	0.589063	0.568945	0.505800	0.522992	0.639139	0.527162	0.617162

5 rows × 345 columns

In [0]:

```
1 drug_response_matrix_norm_withoutMissingValues.shape
```

Out[65]:

(927, 345)

### 2.2.4 :: Export drug\_response\_matrix\_norm\_withoutMissingValues as .csv file

1) Add column "cell line" to the previous dataframe

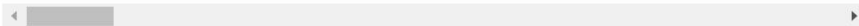
In [0]:

```
1 drug_response_matrix_norm_withoutMissingValues.insert(0, 'cell line', drug_response_mat
2 drug_response_matrix_norm_withoutMissingValues.head())
```

Out[66]:

	cell line	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG-132	Paclitaxel	Cyclopamine	AZ628
0	MC-CAR	0.559607	0.483550	0.553818	0.564980	0.513262	0.409808	0.605797	0.551204
1	ES3	0.577884	0.498306	0.575509	0.572869	0.532380	0.491089	0.619813	0.551578
2	ES5	0.597937	0.514668	0.593249	0.566998	0.523253	0.517003	0.638665	0.566260
3	ES7	0.566915	0.499364	0.574228	0.561957	0.513764	0.501896	0.622429	0.559535
4	EW-11	0.561652	0.447120	0.589063	0.568945	0.505800	0.522992	0.639139	0.527162

5 rows × 346 columns



In [0]:

```
1 drug_response_matrix_norm_withoutMissingValues.shape
```

Out[67]:

(927, 346)

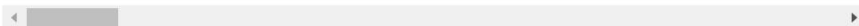
In [0]:

```
1 drug_response_matrix_norm_withoutMissingValues[drug_response_matrix_norm_withoutMissing
```

Out[68]:

	cell line	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG-132	Paclitaxel	Cyclopamine	AZ628
261	UACC-812	0.58974	0.448857	0.5	0.563572	0.535375	0.5	0.638925	0.58974

1 rows × 346 columns



In [0]:

```
1 drug_response_matrix_norm_withoutMissingValues[drug_response_matrix_norm_withoutMissing
```

Out[69]:

	cell line	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG- 132	Paclitaxel	Cyclopamine	AZ628	S
773	201T	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	

1 rows × 346 columns

2) Add column "array data file" to the previous dataframe

In [0]:

```
1 #import .csv file that contains: (927) cell lines names | assays names | assays file names
2 cells927_pd_2 = pd.read_csv('/content/drive/My Drive/RecSys_Code/dataset/CellLines/927CellLines.csv')
3 cells927_pd_2.head(2)
```

Out[70]:

	Characteristics[cell line]	Array Data File
0	UACC-812	5500994173212120213068_A01.cel
1	201T	5500994158987071513209_A01.cel

In [0]:

```
1 #rename columns
2 cells927_pd_2 = cells927_pd_2.rename(columns = {"Characteristics[cell line]": "cell line", "Array Data File": "array_data_file"})
3 print(cells927_pd_2.shape)
4 print(cells927_pd_2.head(2))
```

In [0]:

```
1 drug_response_matrix_norm_withoutMissingValues = cells927_pd_2.merge(drug_response_matrix_norm_withoutMissingValues, on="cell line", how="left")
```

In [0]:

```
1 drug_response_matrix_norm_withoutMissingValues.head(5)
```

Out[73]:

	cell line	array data file	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG-132
0	UACC-812	5500994173212120213068_A01.cel	0.589740	0.448857	0.500000	0.563572	0.535375
1	201T	5500994158987071513209_A01.cel	0.500000	0.500000	0.500000	0.500000	0.500000
2	EVSA-T	5500994172383112813929_A01.cel	0.569937	0.430087	0.548258	0.570864	0.465061
3	KYSE-520	5500994158987071513202_A01.cel	0.500000	0.500000	0.500000	0.500000	0.500000
4	MS751	5500994158987071513207_A01.cel	0.500000	0.500000	0.500000	0.500000	0.500000

5 rows × 347 columns

In [0]:

```
1 drug_response_matrix_norm_withoutMissingValues.shape
```

Out[74]:

(927, 347)

3) Export the final matrix (normalized and with missing values set to 0.5) as .csv file (to reuse it in notebook STAGE2\_EXP2.ipynb)

In [0]:

```
1 #export the final matrix (normalized and with missing values set to 0.5) as .csv file
2 drug_response_matrix_norm_withoutMissingValues.to_csv(r"/content/drive/My Drive/RecSys_
```

## 2.3 Top-N similar cell lines

### 2.3.1 Folder B

In [0]:

```
1 folderB = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderB/"
2 folderB = list(folderB[0])
3 folderB[0]
```

Out[76]:

'5500994158987071513209\_E09.cel'

In [0]:

```
1 for cell in folderB:
2     print("Analysing cell/image:", cell)
3
4     df_similarity, df_dissimilarity = similarity_ssim(cell)
5
6     cell_name = os.path.splitext(cell)[0] #removes the extension .cel from the filename
7     cell_name = str(cell_name) + ".csv"
8
9     #top-50 similar images
10    similar_folderB_path = "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderB/"
11    similar_path = os.path.join(similar_folderB_path, cell_name)
12    df_similarity.to_csv(similar_path, index = False, header=True) #export the dataframe
13
14    #top-10 dissimilar images
15    dissimilar_folderB_path = "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderB/"
16    dissimilar_cell_name = "d" + cell_name
17    dissimilar_path = os.path.join(dissimilar_folderB_path, dissimilar_cell_name)
18    df_dissimilarity.to_csv(dissimilar_path, index = False, header=True) #export the dataframe
19
20
```

Analysing cell/image: 5500994158987071513209\_E09.cel

Current folder under analysis: raw1  
Current folder under analysis: raw5  
Current folder under analysis: raw10  
Current folder under analysis: raw15  
Current folder under analysis: raw20  
Current folder under analysis: raw25

Analysing cell/image: 5500994158987071513207\_G11.cel

Current folder under analysis: raw1  
Current folder under analysis: raw5  
Current folder under analysis: raw10  
Current folder under analysis: raw15  
Current folder under analysis: raw20

### 2.3.2 Folder C

In [0]:

```
1 folderC = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderC/"
2 folderC = list(folderC[0])
3 folderC[0]
```



In [0]:

```

1  for cell in folderC:
2      print("Analysing cell/image:", cell)
3
4      df_similarity, df_dissimilarity = similarity_ssim(cell)
5
6      cell_name = os.path.splitext(cell)[0] #removes the extension .cel from the filename
7      cell_name = str(cell_name) + ".csv"
8
9      #top-50 similar images
10     similar_folderC_path = "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderC/"
11     similar_path = os.path.join(similar_folderC_path, cell_name)
12     df_similarity.to_csv(similar_path, index = False, header=True) #export the dataframe
13
14     #top-10 dissimilar images
15     dissimilar_folderC_path = "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderC/"
16     dissimilar_cell_name = "d" + cell_name
17     dissimilar_path = os.path.join(dissimilar_folderC_path, dissimilar_cell_name)
18     df_dissimilarity.to_csv(dissimilar_path, index = False, header=True) #export the dataframe

```

### 2.3.3 Folder D

In [0]:

```

1  folderD = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderD/"
2  folderD = list(folderD[0])
3  folderD[0]

```

In [0]:

```

1  for cell in folderD:
2      print("Analysing cell/image:", cell)
3
4      df_similarity, df_dissimilarity = similarity_ssim(cell)
5
6      cell_name = os.path.splitext(cell)[0] #removes the extension .cel from the filename
7      cell_name = str(cell_name) + ".csv"
8
9      #top-50 similar images
10     similar_folderD_path = "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderD/"
11     similar_path = os.path.join(similar_folderD_path, cell_name)
12     df_similarity.to_csv(similar_path, index = False, header=True) #export the dataframe
13
14     #top-10 dissimilar images
15     dissimilar_folderD_path = "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderD/"
16     dissimilar_cell_name = "d" + cell_name
17     dissimilar_path = os.path.join(dissimilar_folderD_path, dissimilar_cell_name)
18     df_dissimilarity.to_csv(dissimilar_path, index = False, header=True) #export the dataframe

```

## 2.4 Drug candidates generation

In [0]:

```
1 drug_response_matrix_norm_withoutMissingValues = pd.read_csv("/content/drive/My Drive/1
2 drug_response_matrix_norm_withoutMissingValues.head(2)
```

Out[9]:

	cell line	array data file	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG-132
0	UACC-812	5500994173212120213068_A01.cel	0.58974	0.448857	0.5	0.563572	0.535375
1	201T	5500994158987071513209_A01.cel	0.50000	0.500000	0.5	0.500000	0.500000

2 rows x 347 columns

### 2.4.1 Folder B

STEP 1: For every new user (i.e. cell line in folder B), retrieve the most similar user (top-1 similarity) and store that info into "df1" dataframe

In [0]:

```

1 #create an empty dataframe df1 with the structure: new user | similar user | ssim | array data file
2 columns= ['new user', 'similar user', 'ssim', 'array data file']
3 index = list(range(25)) #25 rows (one for every new user)
4 df1 = pd.DataFrame(index = index, columns = columns)
5
6
7 #import folder B
8 folderB = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderB/"
9 folderB = list(folderB[0])
10 folderB = [(str(os.path.splitext(cell)[0])) for cell in folderB]
11 folderB[0] #e.g. '5500994158987071513209_E09'
12
13
14 #access to the .csv files which contain the top-50 similar cell lines
15 folderB_experiment1_path = "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderB_experiment1/"
16
17
18 #for every new user (i.e. cell line in folder B), retrieve the most similar user (top-50)
19 #and store that info in the "df1" dataframe
20
21 n = 0
22
23 for cell in folderB:
24
25     cell_csv = cell + '.csv'
26     path_cell = os.path.join(folderB_experiment1_path, cell_csv)
27     df_cell = pd.read_csv(path_cell)
28
29     df1.loc[n].at['new user'] = cell + '.jpg'
30     df1.loc[n].at['similar user'] = df_cell.loc[1].at['image']
31     df1.loc[n].at['ssim'] = df_cell.loc[1].at['SSIM']
32
33     array_data_file = df_cell.loc[1].at['image'] #e.g. 5500994157493061613625_C10.jpg
34     array_data_file = str(os.path.splitext(array_data_file)[0]) + '.cel'
35     df1.loc[n].at['array data file'] = array_data_file
36     n += 1
37
38 df1.head(2)
39

```

Out[10]:

	new user	similar user	ssim	
0	5500994158987071513209_E09.jpg	5500994158987071513209_E12.cel	0.744766	55009941589870
1	5500994158987071513207_G11.jpg	5500994158987071513207_C09.cel	0.754121	55009941589870

STEP 2: For every similar user, retrieve the drugs' response and store that info into "df2" dataframe

In [0]:

```

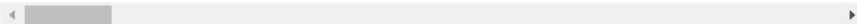
1 #store the similar users .cel filenames (e.g."5500994158987071513209_E12.cel") into a
2 similar_users = df1.loc[:, 'array data file'].tolist()
3
4
5 #retrieve rows, from drug response matrix, that belong to the similar users
6 df2 = drug_response_matrix_norm_withoutMissingValues
7
8 indexes = [] #empty list to store the row indexes (from drug response matrix) that belong
9 n=0         #the order of df1 is kept (i.e., the 1st index belongs to the 1st similar
10
11 for similar_user in similar_users:
12     index = df2[df2["array data file"] == similar_user].index.values.astype(int)[0]
13     indexes.append(index)
14     n += 1
15
16
17 df2 = df2.loc[indexes, :] #retrieve the rows of the drug response matrix
18 df2 = df2.reset_index(drop=True)
19 df2 = df2.drop(columns = ['array data file'])
20
21 df2.head()
22

```

Out[11]:

	cell line	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG-132	Paclitaxel	Cyclopamine	AZ6
0	ES-2	0.569146	0.500000	0.555500	0.527795	0.468730	0.434597	0.596898	0.4888
1	SNB75	0.598558	0.500000	0.585036	0.591141	0.502335	0.398194	0.621480	0.5913
2	PANC- 03-27	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.5000
3	BC-1	0.571770	0.447419	0.562951	0.544063	0.509149	0.462440	0.586561	0.5383
4	SUIT-2	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.5000

5 rows × 346 columns



STEP 3: Concatenate df1 and df2

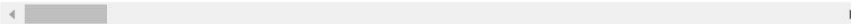
In [0]:

```
1 drug_candidates_generation_folderB = pd.concat([df1, df2], axis = 1)
2 drug_candidates_generation_folderB.head(25)
```

Out[12]:

	new user		similar user	ssim	
0	5500994158987071513209_E09.jpg	5500994158987071513209_E12.cel	0.744766	5500994158987	
1	5500994158987071513207_G11.jpg	5500994158987071513207_C09.cel	0.754121	5500994158987	
2	5500994158987071513209_F08.jpg	5500994158987071513201_H04.cel	0.762577	5500994158987	
3	5500994158987071513209_A10.jpg	5500994158987071513209_H03.cel	0.774402	5500994158987	
4	5500994173212120213068_B04.jpg	5500994172383112813928_C03.cel	0.73787	5500994172383	
5	5500994158987071513201_A08.jpg	5500994172383112813930_C02.cel	0.765332	5500994172383	
6	5500994172383112813928_F06.jpg	5500994172383112813930_D05.cel	0.749291	5500994172383	
7	5500994172383112813928_B03.jpg	5500994172383112813929_D06.cel	0.817804	5500994172383	
8	5500994158987071513209_G04.jpg	5500994158987071513209_D08.cel	0.786151	5500994158987	
9	5500994158987071513207_G07.jpg	5500994158987071513207_D07.cel	0.774367	5500994158987	
10	5500994172383112813929_F11.jpg	5500994172383112813928_B11.cel	0.691162	5500994172383	
11	5500994172383112813930_C02.jpg	5500994158987071513201_G12.cel	0.778825	5500994158987	
12	5500994175999120813240_H10.jpg	5500994172948120113978_H05.cel	0.674877	5500994172948	
13	5500994157493061613625_D11.jpg	5500994158987071513209_D05.cel	0.730805	5500994158987	
14	5500994158987071513202_D01.jpg	5500994158987071513201_B06.cel	0.764124	5500994158987	
15	5500994158987071513207_C10.jpg	5500994158987071513202_C05.cel	0.773456	5500994158987	
16	5500994157493061613625_D08.jpg	5500994157493061613625_E03.cel	0.713918	5500994157493	
17	5500994158987071513202_G09.jpg	5500994172948120113978_H04.cel	0.755039	5500994172948	
18	5500994158987071513207_D12.jpg	5500994157493061613625_C11.cel	0.746729	5500994157493	
19	5500994172948120113978_C09.jpg	5500994172948120113978_D01.cel	0.737993	5500994172948	
20	5500994172383112813929_D11.jpg	5500994172383112813929_E08.cel	0.759326	5500994172383	
21	5500994158987071513201_C06.jpg	5500994173603120813304_A10.cel	0.7067	5500994173603	
22	5500994157493061613625_A07.jpg	5500994158987071513201_G03.cel	0.811949	5500994158987	
23	5500994172383112813929_F06.jpg	5500994172383112813929_B01.cel	0.742445	5500994172383	
24	5500994172383112813930_H09.jpg	5500994172383112813928_F01.cel	0.76188	5500994172383	

25 rows × 350 columns



In [0]:

```
1 #check sparsity (remember: IC50 values not available were replaced by 0.5)
2 sparsityB = drug_candidates_generation_folderB[drug_candidates_generation_folderB.iloc
3 sparsityB
```

Out[13]:

565

### 2.4.2 Folder C

STEP 1: For every new user (i.e. cell line in folder C), retrieve the most similar user (top-1 similarity) and store that info into "df1" dataframe



In [0]:

```

1 #create an empty dataframe df1 with the structure: new user | similar user | ssim | array data file
2 columns= ['new user', 'similar user', 'ssim', 'array data file' ]
3 index = list(range(25))
4 df1 = pd.DataFrame(index = index, columns = columns)
5
6
7 #import folder C
8 folderC = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderC/"
9 folderC = list(folderC[0])
10 folderC = [(str(os.path.splitext(cell)[0])) for cell in folderC]
11 folderC[0] #e.g. '5500994158987071513209_E09'
12
13
14 #access to the .csv files which contain the top-50 similar cell lines
15 folderC_experiment1_path = "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderC_experiment1/"
16
17
18 #for every new user (i.e. cell line in folder C), retrieve the most similar user (top-50)
19 #and store that info in the "df1" dataframe
20
21 n = 0
22
23 for cell in folderC:
24
25     cell_csv = cell + '.csv'
26     path_cell = os.path.join(folderC_experiment1_path, cell_csv)
27     df_cell = pd.read_csv(path_cell)
28
29     df1.loc[n].at['new user'] = cell + '.jpg'
30     df1.loc[n].at['similar user'] = df_cell.loc[1].at['image']
31     df1.loc[n].at['ssim'] = df_cell.loc[1].at['SSIM']
32
33     array_data_file = df_cell.loc[1].at['image'] #e.g. 5500994157493061613625_C10.jpg
34     array_data_file = str(os.path.splitext(array_data_file)[0]) + '.cel'
35     df1.loc[n].at['array data file'] = array_data_file
36     n += 1
37
38
39 df1.head(2)
40

```

Out[14]:

	new user	similar user	ssim	
0	5500994158987071513201_G04.jpg	5500994175999120813240_D04.cel	0.654131	55009941759991
1	5500994158987071513209_G10.jpg	5500994158987071513202_E10.cel	0.76525	55009941589870

STEP 2: For every similar user, retrieve the drugs' response and store that info into "df2" dataframe

In [0]:

```

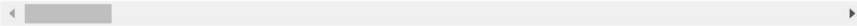
1 #store the similar users .cel filenames (e.g."5500994158987071513209_E12.cel") into a
2 similar_users = df1.loc[:, 'array data file'].tolist()
3
4
5 #retrieve rows, from drug response matrix, that belong to the similar users
6 df2 = drug_response_matrix_norm_withoutMissingValues
7
8 indexes = [] #empty list to store the row indexes (from drug response matrix) that belong
9 n=0         #the order of df1 is kept (i.e., the 1st index belongs to the 1st similar
10
11 for similar_user in similar_users:
12     index = df2[df2["array data file"] == similar_users[n]].index.values.astype(int)[0]
13     indexes.append(index)
14     n += 1
15
16
17 df2 = df2.loc[indexes, :] #retrieve the rows of the drug response matrix
18 df2 = df2.reset_index(drop=True)
19 df2 = df2.drop(columns = ['array data file'])
20
21 df2.head(2)

```

Out[15]:

	cell line	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG-132	Paclitaxel	Cyclopamine	AZ628	
0	RCH- ACV	0.500000		0.5	0.500000	0.500000	0.500000	0.500000	0.500000	
1	GR- ST	0.593271		0.5	0.567969	0.58731	0.540033	0.430408	0.59567	0.549635

2 rows × 346 columns



STEP 3: Concatenate df1 and df2



In [0]:

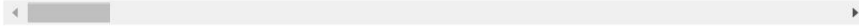
```
1 drug_candidates_generation_folderC = pd.concat([df1, df2], axis = 1)
2 drug_candidates_generation_folderC.head(25)
```

Out[16]:

	new user		similar user	ssim
0	5500994158987071513201_G04.jpg	5500994175999120813240_D04.cel	0.654131	5500994175999
1	5500994158987071513209_G10.jpg	5500994158987071513202_E10.cel	0.76525	5500994158987
2	5500994172383112813929_H11.jpg	5500994172383112813928_F11.cel	0.732257	5500994172383
3	5500994172948120113978_D06.jpg	5500994158987071513201_B09.cel	0.734016	5500994158987
4	5500994157493061613625_F09.jpg	5500994158987071513201_B06.cel	0.78136	5500994158987
5	5500994158987071513207_A10.jpg	5500994173212120213068_D07.cel	0.789542	5500994173212
6	5500994175999120813240_D09.jpg	5500994175999120813240_B11.cel	0.774402	5500994175999
7	5500994172383112813928_C02.jpg	5500994172383112813928_C12.cel	0.775961	5500994172383
8	5500994173212120213068_D01.jpg	5500994172383112813929_B05.cel	0.69964	5500994172383
9	5500994173603120813304_H10.jpg	5500994158987071513207_F04.cel	0.787144	5500994158987
10	5500994172383112813930_F03.jpg	5500994158987071513201_E07.cel	0.775811	5500994158987
11	5500994172383112813930_G11.jpg	5500994158987071513207_G05.cel	0.776452	5500994158987
12	5500994172383112813928_D06.jpg	5500994172383112813928_H11.cel	0.732648	5500994172383
13	5500994157493061613625_A12.jpg	5500994157493061613625_A06.cel	0.763146	5500994157493
14	5500994158987071513207_D09.jpg	5500994158987071513201_C08.cel	0.779163	5500994158987
15	5500994172383112813930_C06.jpg	5500994172383112813928_F07.cel	0.803567	5500994172383
16	5500994158987071513201_C04.jpg	5500994158987071513207_B11.cel	0.810649	5500994158987
17	5500994158987071513209_F03.jpg	5500994158987071513202_E11.cel	0.71624	5500994158987
18	5500994158987071513209_E07.jpg	5500994158987071513209_D12.cel	0.784199	5500994158987
19	5500994172383112813928_A03.jpg	5500994172383112813929_E04.cel	0.802642	5500994172383
20	5500994175999120813240_C11.jpg	5500994172383112813930_G05.cel	0.726803	5500994172383
21	5500994158987071513202_B06.jpg	5500994157493061613625_B08.cel	0.705583	5500994157493
22	5500994158987071513201_E07.jpg	5500994172383112813930_F03.cel	0.775811	5500994172383

	new user	similar user	ssim	
23	5500994158987071513207_H10.jpg	5500994172383112813928_C12.cel	0.792985	5500994172383
24	5500994158987071513209_E12.jpg	5500994172383112813928_A06.cel	0.76942	5500994172383

25 rows × 350 columns



In [0]:

```
1 #check sparsity (remember: IC50 values not available were replaced by 0.5)
2 sparsityC = drug_candidates_generation_folderC[drug_candidates_generation_folderC.iloc
3 sparsityC
```

Out[17]:

1406

### 2.4.3 Folder D

STEP 1: For every new user (i.e. cell line in folder D), retrieve the most similar user (top-1 similarity) and store that info into "df1" dataframe

In [0]:

```

1 #create an empty dataframe df1 with the structure: new user | similar user | ssim | array data file
2 columns= ['new user', 'similar user', 'ssim', 'array data file' ]
3 index = list(range(25))
4 df1 = pd.DataFrame(index = index, columns = columns)
5
6
7 #import folder D
8 folderD = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderD/"
9 folderD = list(folderD[0])
10 folderD = [(str(os.path.splitext(cell)[0])) for cell in folderD]
11 folderD[0] #e.g. '5500994158987071513209_E09'
12
13
14 #access to the .csv files which contain the top-50 similar cell lines
15 folderD_experiment1_path = "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderD_experiment1/"
16
17
18 #for every new user (i.e. cell line in folder D), retrieve the most similar user (top-50)
19 #and store that info in the "df1" dataframe
20
21 n = 0
22
23 for cell in folderD:
24
25     cell_csv = cell + '.csv'
26     path_cell = os.path.join(folderD_experiment1_path, cell_csv)
27     df_cell = pd.read_csv(path_cell)
28
29     df1.loc[n].at['new user'] = cell + '.jpg'
30     df1.loc[n].at['similar user'] = df_cell.loc[1].at['image']
31     df1.loc[n].at['ssim'] = df_cell.loc[1].at['SSIM']
32
33     array_data_file = df_cell.loc[1].at['image'] #e.g. 5500994157493061613625_C10.jpg
34     array_data_file = str(os.path.splitext(array_data_file)[0]) + '.cel'
35     df1.loc[n].at['array data file'] = array_data_file
36     n += 1
37
38
39 df1.head(2)

```

Out[18]:

	new user	similar user	ssim	
0	5500994173212120213068_F06.jpg	5500994158987071513201_G08.cel	0.732	55009941589870
1	5500994158987071513202_G03.jpg	5500994158987071513202_A04.cel	0.788274	55009941589870

STEP 2: For every similar user, retrieve the drugs' response and store that info into "df2" dataframe

In [0]:

```

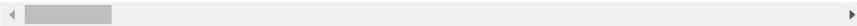
1 #store the similar users .cel filenames (e.g."5500994158987071513209_E12.cel") into a
2 similar_users = df1.loc[:, 'array data file'].tolist()
3
4
5 #retrieve rows, from drug response matrix, that belong to the similar users
6 df2 = drug_response_matrix_norm_withoutMissingValues
7
8 indexes = [] #empty list to store the row indexes (from drug response matrix) that belong
9 n=0         #the order of df1 is kept (i.e., the 1st index belongs to the 1st similar
10
11 for similar_user in similar_users:
12     index = df2[df2["array data file"] == similar_users[n]].index.values.astype(int)[0]
13     indexes.append(index)
14     n += 1
15
16
17 df2 = df2.loc[indexes, :] #retrieve the rows of the drug response matrix
18 df2 = df2.reset_index(drop=True)
19 df2 = df2.drop(columns = ['array data file'])
20
21 df2.head(2)

```

Out[19]:

	cell line	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG-132	Paclitaxel	Cyclopamine	AZI
0	NCI- H1568	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000
1	KINGS- 1	0.585343	0.397651	0.585886	0.460329	0.460236	0.405243	0.635257	0.5236

2 rows × 346 columns



STEP 3: Concatenate df1 and df2

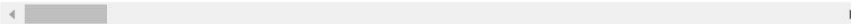
In [0]:

```
1 drug_candidates_generation_folderD = pd.concat([df1, df2], axis = 1)
2 drug_candidates_generation_folderD.head(25)
```

Out[20]:

		new user	similar user	ssim	
0	5500994173212120213068_F06.jpg	5500994158987071513201_G08.cel	0.732	5500994158987	
1	5500994158987071513202_G03.jpg	5500994158987071513202_A04.cel	0.788274	5500994158987	
2	5500994158987071513202_A06.jpg	5500994158987071513209_E12.cel	0.756887	5500994158987	
3	5500994158987071513201_E06.jpg	5500994175999120813240_C08.cel	0.73558	5500994175999	
4	5500994158987071513207_E05.jpg	5500994158987071513207_H10.cel	0.789839	5500994158987	
5	5500994172383112813928_C03.jpg	5500994172383112813930_B02.cel	0.749748	5500994172383	
6	5500994158987071513202_C01.jpg	5500994158987071513209_H10.cel	0.734725	5500994158987	
7	5500994157493061613625_G07.jpg	5500994172383112813930_C07.cel	0.806106	5500994172383	
8	5500994158987071513201_C05.jpg	5500994172383112813928_B01.cel	0.758861	5500994172383	
9	5500994158987071513201_D06.jpg	5500994158987071513202_F08.cel	0.731668	5500994158987	
10	5500994172383112813929_A09.jpg	5500994158987071513201_B06.cel	0.792263	5500994158987	
11	5500994158987071513201_D07.jpg	5500994172383112813928_E01.cel	0.702753	5500994172383	
12	5500994172948120113978_C03.jpg	5500994172948120113978_E11.cel	0.747097	5500994172948	
13	5500994157493061613625_B01.jpg	5500994157493061613625_C08.cel	0.798118	5500994157493	
14	5500994172383112813928_C07.jpg	5500994172383112813928_C09.cel	0.730629	5500994172383	
15	5500994157493061613625_G02.jpg	5500994172383112813928_F12.cel	0.774692	5500994172383	
16	5500994158987071513209_B12.jpg	5500994158987071513202_B02.cel	0.760935	5500994158987	
17	5500994158987071513201_E11.jpg	5500994158987071513201_B06.cel	0.782648	5500994158987	
18	5500994172383112813928_G11.jpg	5500994172383112813930_D12.cel	0.779884	5500994172383	
19	5500994158987071513202_C06.jpg	5500994173212120213068_H08.cel	0.765912	5500994173212	
20	5500994172383112813929_F02.jpg	5500994172383112813929_G03.cel	0.737251	5500994172383	
21	5500994158987071513201_C12.jpg	5500994172383112813930_B01.cel	0.712978	5500994172383	
22	5500994173212120213068_D05.jpg	5500994158987071513207_C09.cel	0.759586	5500994158987	
23	5500994158987071513202_A01.jpg	5500994158987071513202_A05.cel	0.731257	5500994158987	
24	5500994157493061613625_F04.jpg	5500994172383112813929_H10.cel	0.732153	5500994172383	

25 rows × 350 columns



In [0]:

```

1 #check sparsity (remember: IC50 values not available were replaced by 0.5)
2 sparsityD = drug_candidates_generation_folderD[drug_candidates_generation_folderD.iloc
3 sparsityD

```

Out[21]:

1293

## 2.5 Drug candidates score

### 2.5.1 Folder B

In [0]:

```

1 #SSIM x IC50s
2 drug_candidates_score_folderB = drug_candidates_generation_folderB.iloc[:, 5:350].mult
3
4 #add columns "new user", "similar user", "ssim", "array data file", "cell line"
5 drug_candidates_score_folderB = pd.concat([drug_candidates_generation_folderB.iloc[:,0
6
7 drug_candidates_score_folderB.head(2)

```

Out[22]:

	new user	similar user	ssim	
0	5500994158987071513209_E09.jpg	5500994158987071513209_E12.cel	0.744766	55009941589870
1	5500994158987071513207_G11.jpg	5500994158987071513207_C09.cel	0.754121	55009941589870

2 rows × 350 columns

### 2.5.2 Folder C



In [0]:

```

1 #SSIM x IC50s
2 drug_candidates_score_folderC = drug_candidates_generation_folderC.iloc[:, 5:350].mult
3
4 #add columns "new user", "similar user", "ssim", "array data file", "cell line"
5 drug_candidates_score_folderC = pd.concat([drug_candidates_generation_folderC.iloc[:,0
6
7 drug_candidates_score_folderC.head(2)

```

Out[23]:

	new user	similar user	ssim
0	5500994158987071513201_G04.jpg	5500994175999120813240_D04.cel	0.654131
1	5500994158987071513209_G10.jpg	5500994158987071513202_E10.cel	0.76525

2 rows × 350 columns

### 2.5.3 Folder D

In [0]:

```

1 #SSIM x IC50s
2 drug_candidates_score_folderD = drug_candidates_generation_folderD.iloc[:, 5:350].mult
3
4 #add columns "new user", "similar user", "ssim", "array data file", "cell line"
5 drug_candidates_score_folderD = pd.concat([drug_candidates_generation_folderD.iloc[:,0
6
7 drug_candidates_score_folderD.head(2)

```

Out[24]:

	new user	similar user	ssim
0	5500994173212120213068_F06.jpg	5500994158987071513201_G08.cel	0.732
1	5500994158987071513202_G03.jpg	5500994158987071513202_A04.cel	0.788274

2 rows × 350 columns

## 2.6 Top-N recommendation list

### 2.6.1 Folder B

In [0]:

```

1 #import folder B
2 folderB = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderB/"
3 folderB = list(folderB[0])
4 folderB = [(str(os.path.splitext(cell)[0])) for cell in folderB] #e.g. '55009941589870
5
6 #create empty dictionary to store the top20 recommendations; key values correspond to
7 #e.g. folderB_top20_rec['5500994158987071513209_E09'] gives the top-20 recommendations
8 folderB_top20_rec = {}
9 row_index = 0
10
11 for new_user in folderB:
12
13     recommendation = drug_candidates_score_folderB.iloc[row_index, 5:350].sort_values(asc
14     top20 = recommendation.iloc[0:20]
15     folderB_top20_rec[new_user] = top20
16
17     row_index +=1
18

```

In [0]:

```

1 folderB_top20_rec['5500994158987071513209_E09']

```

Out[26]:

Bortezomib	0.264645
Trametinib	0.27564
Docetaxel	0.276623
Sepantronium bromide	0.279906
SN-38	0.289451
Elesclomol	0.290565
Vinblastine	0.301368
Gemcitabine	0.301491
PD0325901	0.30446
AZD4877	0.313895
Epothilone B	0.314956
Panobinostat	0.315119
Thapsigargin	0.316282
Dabrafenib	0.318894
Bryostatin 1	0.321202
Methotrexate	0.322148
GW843682X	0.32338
Dactolisib	0.323534
Paclitaxel	0.323674
ARRY-520	0.327603

Name: 0, dtype: object

### 2.6.2 Folder C



In [0]:

```

1 #import folder C
2 folderC = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderC/"
3 folderC = list(folderC[0])
4 folderC = [(str(os.path.splitext(cell)[0])) for cell in folderC] #e.g. '55009941589870
5
6 #create empty dictionary to store the top20 recommendations; key values correspond to
7 #e.g. folderC_top20_rec['5500994158987071513209_E09'] gives the top-20 recommendations
8 folderC_top20_rec = {}
9 row_index = 0
10
11 for new_user in folderC:
12
13     recommendation = drug_candidates_score_folderC.iloc[row_index, 5:350].sort_values(asc
14     top20 = recommendation.iloc[0:20]
15     folderC_top20_rec[new_user] = top20
16
17     row_index +=1
18

```

### 2.6.3 Folder D

In [0]:

```

1 #import folder D
2 folderD = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderD/"
3 folderD = list(folderD[0])
4 folderD = [(str(os.path.splitext(cell)[0])) for cell in folderD] #e.g. '55009941589870
5
6 #create empty dictionary to store the top20 recommendations; key values correspond to
7 #e.g. folderD_top20_rec['5500994158987071513209_E09'] gives the top-20 recommendations
8 folderD_top20_rec = {}
9 row_index = 0
10
11 for new_user in folderD:
12
13     recommendation = drug_candidates_score_folderD.iloc[row_index, 5:350].sort_values(asc
14     top20 = recommendation.iloc[0:20]
15     folderD_top20_rec[new_user] = top20
16
17     row_index +=1
18

```

## 2.7 Evaluation - Top-N hit-rate & average reciprocal hit-rate

### 2.7.1 Folder B

#### 2.7.1.1 Top-N hit-rate

In [0]:

```

1 #import folder B
2 folderB = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderB/"
3 folderB_cel = list(folderB[0]) #e.g. '5500994158987071513209_E09.cel'
4 folderB = [(str(os.path.splitext(cell)[0])) for cell in folderB_cel] # e.g. '550099415
5
6
7 #1) for every new user, retrieve the real top-20 drugs' response (from the drug response
8 df = drug_response_matrix_norm_withoutMissingValues
9
10 indexes = [] #empty list to store the row indexes (from drug response matrix) that belo
11 n=0
12
13 for new_user in folderB_cel:
14     index = df[df["array data file"] == folderB_cel[n]].index.values.astype(int)[0]
15     indexes.append(index)
16     n += 1
17
18
19 df = df.loc[indexes, :] #retrieve the rows of the drug response matrix
20
21
22 #2) create an empty dictionary to store the real top-20 ; key values correspond to new
23 #e.g. folderB_top20_real['5500994157493061613625_A07'] gives the real top-20 for the ne
24 folderB_top20_real = {}
25 row_index = 0
26
27 for new_user in folderB:
28
29     recommendation = df.iloc[row_index, 2:347].sort_values(ascending=True)
30     top20 = recommendation.iloc[0:20]
31     folderB_top20_real[new_user] = top20
32
33     row_index +=1
34
35
36 #compute hit-rate
37 number_of_hits = 0 #a hit occurs when a drug appears on both folderB_top20_rec and fol
38
39 for new_user in folderB:
40     set_rec_drugs = set(folderB_top20_rec[new_user].index) #store recommended drugs into
41     set_real_drugs = set(folderB_top20_real[new_user].index) #store real/relevant drugs
42     common_drugs = set_rec_drugs.intersection(set_real_drugs) #perform set intersection
43
44     number_of_hits = number_of_hits + len(common_drugs) #cumulative number of hits
45
46 hit_rate_B = number_of_hits/(len(folderB)) #hit_rate = (total number of hits in folder
47
48
49 print("Folder B hit-rate:", hit_rate_B)
50 print("(Number of recommended drugs:", len(set_rec_drugs), ")")
51

```

Folder B hit-rate: 11.84  
(Number of recommended drugs: 20 )

### 2.7.1.2 Average reciprocal hit-rate

In [0]:

```

1 #import folder B
2 folderB = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderB/"
3 folderB = list(folderB[0])
4 folderB = [(str(os.path.splitext(cell)[0])) for cell in folderB] # e.g. '5500994158987
5
6 weights = [1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/10, 1/11,
7            1/12, 1/13, 1/14, 1/15, 1/16, 1/17, 1/18, 1/19, 1/20]
8
9 total_sum_weights = 0 #store the weights'sum of all new users
10
11 for new_user in folderB:
12
13     #create dataframe "real top-20 drugs & weights"
14     real_list = list(folderB_top20_real[new_user].index)
15
16     df_real_weights = pd.DataFrame(weights).transpose()
17     df_real_weights.columns = real_list
18
19     #remove drugs/columns of dataframe "df_real_weights" that weren't recommended
20     rec_set = set(folderB_top20_rec[new_user].index)
21     real_set = set(folderB_top20_real[new_user].index)
22     not_recommended_drugs = real_set.difference(rec_set) #identify drugs in real_set tha
23     not_recommended_drugs = list(not_recommended_drugs)
24
25     df_real_weights = df_real_weights.drop(columns = not_recommended_drugs)
26
27     #compute ARHR
28     total_sum_weights = total_sum_weights + df_real_weights.sum().sum()
29
30     aver_recip_hit_rate_B = total_sum_weights/len(folderB)
31
32 print("Folder B average reciprocal hit-rate:", aver_recip_hit_rate_B)
33 print("(Maximum average reciprocal hit-rate:", "3.597739657143682")
34 #note: maximum average reciprocal hit-rate was computed in the notebook STAGE2_EXP2

```

Folder B average reciprocal hit-rate: 2.5922828459809875  
 (Maximum average reciprocal hit-rate: 3.597739657143682

## 2.7.2 Folder C

### 2.7.2.1 Top-N hit-rate

In [0]:

```

1 #import folder C
2 folderC = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderC/"
3 folderC_cel = list(folderC[0]) #e.g. '5500994158987071513209_E09.cel'
4 folderC = [(str(os.path.splitext(cell)[0])) for cell in folderC_cel] # e.g. '550099415
5
6
7 #1) for every new user, retrieve the real top-20 drugs' response (from the drug response
8 df = drug_response_matrix_norm_withoutMissingValues
9
10 indexes = [] #empty list to store the row indexes (from drug response matrix) that belo
11 n=0
12
13
14 for new_user in folderC_cel:
15     index = df[df["array data file"] == folderC_cel[n]].index.values.astype(int)[0]
16     indexes.append(index)
17     n += 1
18
19
20 df = df.loc[indexes, :] #retrieve the rows of the drug response matrix
21
22
23 #2) create an empty dictionary to store the real top-20 ; key values correspond to new
24 #e.g. folderC_top20_real['5500994157493061613625_A07'] gives the real top-20 for the n
25 folderC_top20_real = {}
26 row_index = 0
27
28 for new_user in folderC:
29
30     recommendation = df.iloc[row_index, 2:347].sort_values(ascending=True)
31     top20 = recommendation.iloc[0:20]
32     folderC_top20_real[new_user] = top20
33
34     row_index +=1
35
36
37 #compute hit-rate
38 number_of_hits = 0 #a hit occurs when a drug appears on both folderC_top20_rec and fol
39
40 for new_user in folderC:
41     set_rec_drugs = set(folderC_top20_rec[new_user].index) #store recommended drugs into
42     set_real_drugs = set(folderC_top20_real[new_user].index) #store real/relevant drugs
43     common_drugs = set_rec_drugs.intersection(set_real_drugs) #perform set intersection
44
45     number_of_hits = number_of_hits + len(common_drugs) #cumulative number of hits
46
47 hit_rate_C = number_of_hits/(len(folderC)) #hit_rate = (total number of hits in folder
48
49
50 print("Folder C hit-rate:", hit_rate_C)
51 print("(Number of recommended drugs:", len(set_rec_drugs), ")")

```

Folder C hit-rate: 10.24  
(Number of recommended drugs: 20 )

### 2.7.2.2 Average reciprocal hit-rate



In [0]:

```

1 #import folder C
2 folderC = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderC/"
3 folderC = list(folderC[0])
4 folderC = [(str(os.path.splitext(cell)[0])) for cell in folderC] # e.g. '55009941589870
5
6 weights = [1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/10, 1/11,
7            1/12, 1/13, 1/14, 1/15, 1/16, 1/17, 1/18, 1/19, 1/20]
8
9 total_sum_weights = 0 #store the weights'sum of all new users
10
11 for new_user in folderC:
12
13     #create dataframe "real top-20 drugs & weights"
14     real_list = list(folderC_top20_real[new_user].index)
15
16     df_real_weights = pd.DataFrame(weights).transpose()
17     df_real_weights.columns = real_list
18
19     #remove drugs/columns of dataframe "df_real_weights" that weren't recommended
20     rec_set = set(folderC_top20_rec[new_user].index)
21     real_set = set(folderC_top20_real[new_user].index)
22     not_recommended_drugs = real_set.difference(rec_set) #identify drugs in real_set tha
23     not_recommended_drugs = list(not_recommended_drugs)
24
25     df_real_weights = df_real_weights.drop(columns = not_recommended_drugs)
26
27     #compute ARHR
28     total_sum_weights = total_sum_weights + df_real_weights.sum().sum()
29
30     aver_recip_hit_rate_C = total_sum_weights/len(folderC)
31
32 print("Folder C average reciprocal hit-rate:", aver_recip_hit_rate_C)
33 print("(Maximum average reciprocal hit-rate:", "3.597739657143682")
34 #note: maximum average reciprocal hit-rate was computed in the notebook STAGE2_EXP2

```

Folder C average reciprocal hit-rate: 2.1510852051285485  
 (Maximum average reciprocal hit-rate: 3.597739657143682

## 2.7.3 Folder D

### 2.7.3.1 Top-N hit-rate

In [0]:

```

1 #import folder D
2 folderD = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderD/"
3 folderD_cel = list(folderD[0]) #e.g. '5500994158987071513209_E09.cel'
4 folderD = [(str(os.path.splitext(cell)[0])) for cell in folderD_cel] # e.g. '550099415
5
6
7 #1) for every new user, retrieve the real top-20 drugs' response (from the drug response
8 df = drug_response_matrix_norm_withoutMissingValues
9
10 indexes = [] #empty list to store the row indexes (from drug response matrix) that belo
11 n=0
12
13 for new_user in folderD_cel:
14     index = df[df["array data file"] == folderD_cel[n]].index.values.astype(int)[0]
15     indexes.append(index)
16     n += 1
17
18
19 df = df.loc[indexes, :] #retrieve the rows of the drug response matrix
20
21
22 #2) create an empty dictionary to store the real top-20 ; key values correspond to new
23 #e.g. folderD_top20_real['5500994157493061613625_A07'] gives the real top-20 for the ne
24 folderD_top20_real = {}
25 row_index = 0
26
27 for new_user in folderD:
28
29     recommendation = df.iloc[row_index, 2:347].sort_values(ascending=True)
30     top20 = recommendation.iloc[0:20]
31     folderD_top20_real[new_user] = top20
32
33     row_index +=1
34
35
36 #compute hit-rate
37 number_of_hits = 0 #a hit occurs when a drug appears on both folderD_top20_rec and fol
38
39 for new_user in folderD:
40     set_rec_drugs = set(folderD_top20_rec[new_user].index) #store recommended drugs into
41     set_real_drugs = set(folderD_top20_real[new_user].index) #store real/relevant drugs
42     common_drugs = set_rec_drugs.intersection(set_real_drugs) #perform set intersection
43
44     number_of_hits = number_of_hits + len(common_drugs) #cumulative number of hits
45
46 hit_rate_D = number_of_hits/(len(folderD)) #hit_rate = (total number of hits in folder
47
48
49 print("Folder D hit-rate:", hit_rate_D)
50 print("(Number of recommended drugs:", len(set_rec_drugs), ")")

```

Folder D hit-rate: 11.84  
(Number of recommended drugs: 20 )

### 2.7.3.2 Average reciprocal hit-rate

In [0]:

```

1 #import folder D
2 folderD = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderD/"
3 folderD = list(folderD[0])
4 folderD = [(str(os.path.splitext(cell)[0])) for cell in folderD] # e.g. '5500994158987
5
6 weights = [1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/10, 1/11,
7            1/12, 1/13, 1/14, 1/15, 1/16, 1/17, 1/18, 1/19, 1/20]
8
9 total_sum_weights = 0 #store the weights'sum of all new users
10
11 for new_user in folderD:
12
13     #create dataframe "real top-20 drugs & weights"
14     real_list = list(folderD_top20_real[new_user].index)
15
16     df_real_weights = pd.DataFrame(weights).transpose()
17     df_real_weights.columns = real_list
18
19     #remove drugs/columns of dataframe "df_real_weights" that weren't recommended
20     rec_set = set(folderD_top20_rec[new_user].index)
21     real_set = set(folderD_top20_real[new_user].index)
22     not_recommended_drugs = real_set.difference(rec_set) #identify drugs in real_set tha
23     not_recommended_drugs = list(not_recommended_drugs)
24
25     df_real_weights = df_real_weights.drop(columns = not_recommended_drugs)
26
27     #compute ARHR
28     total_sum_weights = total_sum_weights + df_real_weights.sum().sum()
29
30     aver_recip_hit_rate_D = total_sum_weights/len(folderD)
31
32 print("Folder D average reciprocal hit-rate:", aver_recip_hit_rate_D)
33 print("(Maximum average reciprocal hit-rate:", "3.597739657143682")
34 #note: maximum average reciprocal hit-rate was computed in the notebook STAGE2_EXP2

```

Folder D average reciprocal hit-rate: 2.4278351995441776  
(Maximum average reciprocal hit-rate: 3.597739657143682)

## 2.7.4 Overall / Final results

In [0]:

```

1 hit_rate_average = (hit_rate_B + hit_rate_C + hit_rate_D)/3
2 hit_rate_average

```

Out[35]:

11.306666666666667

In [0]:

```

1 arht_average = (aver_recip_hit_rate_B + aver_recip_hit_rate_C + aver_recip_hit_rate_D)
2 arht_average

```

Out[36]:

2.390401083551238

## APPENDIX 3. EXPERIMENT 2 – PYTHON CODE.

### Contents

#### 1 INTRODUCTION

##### 1.1 Google Drive Access

##### 1.2 Packages

##### 1.3 Customized function *similarity\_ssim(image.jpg)*

##### 1.4 Drug response matrix

##### 1.5 Dataset split

#### 2 WAVELET TRANSFORM

2.1 Read CEL file, extract 2<sup>nd</sup> level detail coeffs & store the 3 detail channels image with the assay filename (containing the gene expression profile) of the cell line

#### 3 FOLDER B

##### 3.1 Top-N similar cell lines

##### 3.2 Drug candidates generation

##### 3.3 Drug candidates score

##### 3.4 Top-N recommendation list

##### 3.5 Evaluation – hit-rate & average reciprocal hit-rate

###### 3.5.1 Top-N hit-rate

###### 3.5.2 Average reciprocal hit-rate

#### 4 FOLDER C

##### 4.1 Top-N similar cell lines

##### 4.2 Drug candidates generation

##### 4.3 Drug candidates score

##### 4.4 Top-N recommendation list

##### 4.5 Evaluation – hit-rate & average reciprocal hit-rate



4.5.1 Top-N hit-rate

4.5.2 Average reciprocal hit-rate

5 FOLDER D

5.1 Top-N similar cell lines

5.2 Drug candidates generation

5.3 Drug candidates score

5.4 Top-N recommendation list

5.5 Evaluation – hit-rate & average reciprocal hit-rate

5.5.1 Top-N hit-rate

5.5.2 Average reciprocal hit-rate

6 Overall/Final Results

# 1 INTRODUCTION

## 1.1 Google Drive Access

In [0]:

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

Go to this URL in a browser: [https://accounts.google.com/o/oauth2/auth?client\\_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect\\_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aob&response\\_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly](https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly)

Enter your authorization code:

.....

Mounted at /content/drive

## 1.2 Packages

In [0]:

```
1 pip install biopython
```

Requirement already satisfied: biopython in /usr/local/lib/python3.6/dist-packages (1.76)

Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from biopython) (1.18.2)

In [0]:

```
1 from Bio.Affy import CelFile #package biopython (.cel files)
2 import pywt #package PyWavelets (wavelet transform)
3 from pywt import wavedec2
4 import cv2 #merge 2nd level detail channels into a _x_x3 image, read & save images
5 from skimage.metrics import structural_similarity as ssim #structural similarity index
6 import cv2 #merge 4th level detail channels into a 74x74x3 image, read & save images
7 import pandas as pd
8 import numpy as np
9 import os #handle paths to directories/files
```

## 1.3 Customized function *similarity\_ssim(image.jpg)*

In [0]:

```

1 def similarity_ssim(image):
2
3     """
4     Description:
5     This function takes a 3-detail channels image (.jpg) and computes its similarity,
6     using the Structural Similarity Index, regarding the other 3-detail channels
7     images (.jpg) from the database.
8     It outputs the top-50 most similar and the top-10 most dissimilar images.
9
10    Parameter:
11    image - image (.jpg) whose similarity to the other images (.jpg) will be computed
12
13    Returns:
14    top-50 most similar images
15    top-10 most dissimilar images
16
17    Example:
18    >>> similarity_ssim('5500994157493061613625_A10.jpg')
19
20    """
21
22    #create an empty dataframe to store the results
23    columns_name = ["image", "SSIM"]
24    df = pd.DataFrame(columns = columns_name)
25    n=0
26
27    image_path = os.path.join('/content/drive/My Drive/RecSys_Code/dataset/Celllines/2ndWaveletImages')
28    images_path = '/content/drive/My Drive/RecSys_Code/dataset/Celllines/2ndWaveletImages'
29    images_filenames = os.listdir(images_path) #Lists all the files in the directory "2ndWaveletImages"
30
31    img = cv2.imread(image_path) #read the image given as parameter
32
33
34
35    for image in images_filenames:
36        img_database_path = os.path.join('/content/drive/My Drive/RecSys_Code/dataset/Celllines/2ndWaveletImages')
37        img_database = cv2.imread(img_database_path) #read the images in the database
38
39        img_ssim = ssim(img, img_database, multichannel = True)
40
41        df.loc[n] = [image] + [img_ssim]
42        n += 1
43        if n==10 or n==50 or n==100 or n==200 or n==300 or n==400 or n==500 or n==600 or n==700 or n==800 or n==900 or n==1000:
44            print("Number of database cell lines analysed so far:", n)
45
46    df_similar = df.sort_values(by='SSIM', axis=0, ascending=False).head(50)
47    df_dissimilar = df.sort_values(by='SSIM', axis=0, ascending=False).tail(10)
48    print()
49
50    return df_similar, df_dissimilar

```

## 1.4 Drug response matrix

In [0]:

```
1 drug_response_matrix_norm_withoutMissingValues = pd.read_csv("/content/drive/My Drive/RecSys_Code/dataset/CellLines/DrugResponseMatrix/DrugResponseMatrix_norm_withoutMissingValues.csv")
2 drug_response_matrix_norm_withoutMissingValues.head(2)
```

Out[6]:

	cell line	array data file	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG-132
0	UACC-812	5500994173212120213068_A01.cel	0.58974	0.448857	0.5	0.563572	0.535375
1	201T	5500994158987071513209_A01.cel	0.50000	0.500000	0.5	0.500000	0.500000

2 rows × 347 columns

## 1.5 Dataset split

In [0]:

```
1 #import .csv file which identifies the cell line names of folder B
2 folderB = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderB/CellLines_B.csv")
3 folderB = list(folderB[0])
4 folderB[0]
```

Out[8]:

'5500994158987071513209\_E09.cel'

In [0]:

```
1 #import .csv file which identifies the cell line names of folder C
2 folderC = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderC/CellLines_C.csv")
3 folderC = list(folderC[0])
4 folderC[0]
```

Out[9]:

'5500994158987071513201\_G04.cel'

In [0]:

```
1 #import .csv file which identifies the cell line names of folder D
2 folderD = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderD/CellLines_D.csv")
3 folderD = list(folderD[0])
4 folderD[0]
```

Out[10]:

'5500994173212120213068\_F06.cel'

## 2 WAVELET TRANSFORM

### 2.1 Read CEL file, extract 2nd level detail coeffs & store the 3

**detail channels image with the assay filename (containing the gene expression profile) of the cell line**

In [0]:

```
1 #c = pywt.wavedec2(sample_microarray, 'db7', mode='periodization', Level=2)
2 #Output: list [cAn, (cHn, cVn, cDn), ... (cH1, cV1, cD1)], i.e.,
3 #Level 2 c[0], c[1]: cA2, (cH2, cV2, cD2)
4 #Level 1 c[3]: (cH1, cV1, cD1)
```

In [0]:

```

1 #customized function - input: raw folder; output: 3 detail channels
2 def details_2ndlevel_extractor(raw):
3
4     """
5     Description:
6     This function performs a 2-D wavelet transform and outputs the images concerning horizontal,
7     vertical and diagonal 2-level details as a single image with 3 channels.
8
9     Parameter:
10    raw - folder that contains the original images
11
12    Returns:
13    3-detail channels image (channels = horizontal, vertical and diagonal details)
14
15    Example:
16    >>> details_2ndlevel_extractor('raw1')
17
18    """
19
20
21    raw_path = os.path.join('/content/drive/My Drive/RecSys_Code/dataset/CellLines/GeneE)
22    raw_filenames = os.listdir(raw_path)
23
24
25    filenames_927 = os.listdir('/content/drive/My Drive/RecSys_Code/dataset/CellLines/4tl)
26    filenames_927 = [((os.path.splitext(file)[0]) + '.cel') for file in filenames_927] #
27    filenames = list(set(filenames_927) & set(raw_filenames))
28
29    for file in filenames:
30        file_cel = file
31        #print("This is the file under analysis:", file)
32
33        img_name = (os.path.splitext(file)[0]) + '.jpg'
34
35        path = os.path.join(raw_path, file_cel)
36        with open(path) as handle:
37            c = CelFile.read(handle) #read CEL file
38            dna_microarray = c.intensities
39
40            c = pywt.wavedec2(dna_microarray, 'db7', mode='periodization', level=2) #perform W
41            h = np.array(c[1][0]) #img 2nd level horizontal details
42            v = np.array(c[1][1]) #img 2nd level vertical details
43            d = np.array(c[1][2]) #img 2nd level diagonal details
44            img = cv2.merge((h,v,d)) #merge single imgs to obtain a 3 detail channels image
45
46            path = os.path.join('/content/drive/My Drive/RecSys_Code/dataset/CellLines/2ndWave)
47            cv2.imwrite(path, img) #save final img into Google Drive
48            print("The extraction finished.", len(filenames), "images were saved.")

```

In [0]:

```
1 details_2ndlevel_extractor('raw1')
```

The extraction finished. 36 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw2')
```

The extraction finished. 37 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw3')
```

The extraction finished. 39 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw4')
```

The extraction finished. 38 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw5')
```

The extraction finished. 36 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw6')
```

The extraction finished. 37 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw7')
```

The extraction finished. 37 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw8')
```

The extraction finished. 39 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw9')
```

The extraction finished. 36 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw10')
```

The extraction finished. 40 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw11')
```

The extraction finished. 37 images were saved.



In [0]:

```
1 details_2ndlevel_extractor('raw12')
```

The extraction finished. 40 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw13')
```

The extraction finished. 40 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw14')
```

The extraction finished. 38 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw15')
```

The extraction finished. 38 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw16')
```

The extraction finished. 41 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw17')
```

The extraction finished. 38 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw18')
```

The extraction finished. 35 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw19')
```

The extraction finished. 37 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw20')
```

The extraction finished. 37 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw21')
```

The extraction finished. 34 images were saved.



In [0]:

```
1 details_2ndlevel_extractor('raw22')
```

The extraction finished. 37 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw23')
```

The extraction finished. 39 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw24')
```

The extraction finished. 38 images were saved.

In [0]:

```
1 details_2ndlevel_extractor('raw25')
```

The extraction finished. 23 images were saved.

## 3 FOLDER B

### 3.1 Top-N similar cell lines

In [0]:

```
1 #replace the .cel extension by .jpg (in order to use the function similarity_ssim(image1, image2))
2 folderB = [(str(os.path.splitext(cell)[0]) + ".jpg") for cell in folderA]
3 folderB[0]
```

Out[51]:

'5500994158987071513209\_E09.jpg'

In [0]:

```

1 for cell in folderB:
2     print("Analysing cell/image:", cell)
3
4     df_similarity, df_dissimilarity = similarity_ssim(cell)
5
6     cell_name = os.path.splitext(cell)[0] #removes the extension .jpg from the filename
7     cell_name = str(cell_name) + ".csv"
8
9     #top-50 similar images
10    similar_folderB_path = "/content/drive/My Drive/RecSys_Code/dataset/Celllines/folderB/"
11    similar_path = os.path.join(similar_folderB_path, cell_name)
12    df_similarity.to_csv(similar_path, index = False, header=True) #export the dataframe
13
14    #top-10 dissimilar images
15    dissimilar_folderB_path = "/content/drive/My Drive/RecSys_Code/dataset/Celllines/folderB/"
16    dissimilar_cell_name = "d" + cell_name
17    dissimilar_path = os.path.join(dissimilar_folderB_path, dissimilar_cell_name)
18    df_dissimilarity.to_csv(dissimilar_path, index = False, header=True) #export the dataframe
19

```

```

Analysing cell/image: 5500994158987071513209_E09.jpg
Number of database cell lines analysed so far: 10
Number of database cell lines analysed so far: 50
Number of database cell lines analysed so far: 100
Number of database cell lines analysed so far: 200
Number of database cell lines analysed so far: 300
Number of database cell lines analysed so far: 400
Number of database cell lines analysed so far: 500
Number of database cell lines analysed so far: 600
Number of database cell lines analysed so far: 700
Number of database cell lines analysed so far: 800

```

```

Analysing cell/image: 5500994158987071513207_G11.jpg
Number of database cell lines analysed so far: 10
Number of database cell lines analysed so far: 50
Number of database cell lines analysed so far: 100
Number of database cell lines analysed so far: 200
Number of database cell lines analysed so far: 300
Number of database cell lines analysed so far: 400
Number of database cell lines analysed so far: 500

```

## 3.2 Drug candidates generation

STEP 1: For every new user (i.e. cell line in folder B), retrieve the most similar user (top-1 similarity) and store that info into "df1" dataframe

In [0]:

```

1 #create an empty dataframe df1 with the structure: new user | similar user | ssim | array data file
2 columns= ['new user', 'similar user', 'ssim', 'array data file']
3 index = list(range(25))
4 df1 = pd.DataFrame(index = index, columns = columns)
5
6
7 #import folder B
8 folderB = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderB/"
9 folderB = list(folderB[0])
10 folderB = [(str(os.path.splitext(cell)[0])) for cell in folderB] #e.g. '55009941589870
11
12
13 folderB_experiment3_path = "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderB_experiment3/"
14
15
16 #for every new user (i.e. cell line in folder B), retrieve the most similar user (top-1)
17 #and store that info in the "df1" dataframe
18
19 n = 0
20
21 for cell in folderB:
22
23     cell_csv = cell + '.csv'
24     path_cell = os.path.join(folderB_experiment3_path, cell_csv)
25     df_cell = pd.read_csv(path_cell)
26
27     df1.loc[n].at['new user'] = cell + '.jpg'
28     df1.loc[n].at['similar user'] = df_cell.loc[1].at['image']
29     df1.loc[n].at['ssim'] = df_cell.loc[1].at['SSIM']
30
31     array_data_file = df_cell.loc[1].at['image'] #e.g. 5500994157493061613625_C10.jpg
32     array_data_file = str(os.path.splitext(array_data_file)[0]) + '.cel'
33     df1.loc[n].at['array data file'] = array_data_file
34     n += 1
35
36 df1.head(2)

```

Out[20]:

	new user	similar user	ssim	
0	5500994158987071513209_E09.jpg	5500994157493061613625_C10.jpg	0.792672	5500994157493061613625_C10.jpg
1	5500994158987071513207_G11.jpg	5500994158987071513207_C09.jpg	0.814494	5500994158987071513207_C09.jpg

STEP 2: For every similar user, retrieve the drugs' response and store that info into "df2" dataframe

In [0]:

```

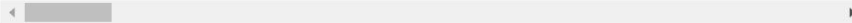
1 #store the similar users .cel filenames (e.g."5500994158987071513209_E12.cel") into a
2 similar_users = df1.loc[:, 'array data file'].tolist()
3
4
5 #retrieve rows, from drug response matrix, that belong to the similar users
6 df2 = drug_response_matrix_norm_withoutMissingValues
7
8 indexes = [] #empty list to store the row indexes (from drug response matrix) that belong
9 n=0         #the order of df1 is kept (i.e., the 1st index belongs to the 1st similar
10
11 for similar_user in similar_users:
12     index = df2[df2["array data file"] == similar_users[n]].index.values.astype(int)[0]
13     indexes.append(index)
14     n += 1
15
16
17 df2 = df2.loc[indexes, :] #retrieve the rows of the drug response matrix
18 df2 = df2.reset_index(drop=True)
19 df2 = df2.drop(columns = ['array data file'])
20
21 df2.head()

```

Out[21]:

	cell line	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG-132	Paclitaxel	Cyclopamine	AZ6
0	HCT-116	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.5000
1	SNB75	0.598558	0.500000	0.585036	0.591141	0.502335	0.398194	0.621480	0.5913
2	PANC-03-27	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.5000
3	BC-1	0.571770	0.447419	0.562951	0.544063	0.509149	0.462440	0.586561	0.5383
4	SUIT-2	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.5000

5 rows × 346 columns



STEP 3: Concatenate df1 and df2

In [0]:

```
1 drug_candidates_generation_folderB = pd.concat([df1, df2], axis = 1)
2 drug_candidates_generation_folderB.head(25)
```

Out[22]:

	new user		similar user	ssim	
0	5500994158987071513209_E09.jpg	5500994157493061613625_C10.jpg	0.792672	5500994157493	
1	5500994158987071513207_G11.jpg	5500994158987071513207_C09.jpg	0.814494	5500994158987	
2	5500994158987071513209_F08.jpg	5500994158987071513201_H04.jpg	0.815679	5500994158987	
3	5500994158987071513209_A10.jpg	5500994158987071513209_H03.jpg	0.837917	5500994158987	
4	5500994173212120213068_B04.jpg	5500994172383112813928_C03.jpg	0.81978	5500994172383	
5	5500994158987071513201_A08.jpg	5500994172383112813930_C02.jpg	0.805321	5500994172383	
6	5500994172383112813928_F06.jpg	5500994173212120213068_A02.jpg	0.79795	5500994173212	
7	5500994172383112813928_B03.jpg	5500994172383112813929_D06.jpg	0.864549	5500994172383	
8	5500994158987071513209_G04.jpg	5500994158987071513207_F04.jpg	0.849332	5500994158987	
9	5500994158987071513207_G07.jpg	5500994158987071513207_D07.jpg	0.833984	5500994158987	
10	5500994172383112813929_F11.jpg	5500994158987071513201_E04.jpg	0.81408	5500994158987	
11	5500994172383112813930_C02.jpg	5500994158987071513201_G12.jpg	0.834953	5500994158987	
12	5500994175999120813240_H10.jpg	5500994175999120813240_E09.jpg	0.764571	5500994175999	
13	5500994157493061613625_D11.jpg	5500994172383112813928_C01.jpg	0.790761	5500994172383	
14	5500994158987071513202_D01.jpg	5500994175999120813240_B05.jpg	0.809064	5500994175999	
15	5500994158987071513207_C10.jpg	5500994172383112813930_H10.jpg	0.818	5500994172383	
16	5500994157493061613625_D08.jpg	5500994175999120813240_E02.jpg	0.778173	5500994175999	
17	5500994158987071513202_G09.jpg	5500994172948120113978_H04.jpg	0.82366	5500994172948	
18	5500994158987071513207_D12.jpg	5500994158987071513207_E04.jpg	0.855842	5500994158987	
19	5500994172948120113978_C09.jpg	5500994158987071513202_C06.jpg	0.814064	5500994158987	
20	5500994172383112813929_D11.jpg	5500994158987071513202_C05.jpg	0.817998	5500994158987	
21	5500994158987071513201_C06.jpg	5500994173603120813304_A10.jpg	0.818197	5500994173603	
22	5500994157493061613625_A07.jpg	5500994158987071513201_G03.jpg	0.895716	5500994158987	
23	5500994172383112813929_F06.jpg	5500994172383112813929_A07.jpg	0.800849	5500994172383	
24	5500994172383112813930_H09.jpg	5500994173212120213068_D04.jpg	0.827526	5500994173212	

25 rows × 350 columns

In [0]:

```

1 #check sparsity (remember: IC50 values not available were replaced by 0.5)
2 sparsityB = drug_candidates_generation_folderB[drug_candidates_generation_folderB.iloc
3 sparsityB

```

Out[23]:

645

### 3.3 Drug candidates score

In [0]:

```

1 #SSIM x IC50s
2 drug_candidates_score_folderB = drug_candidates_generation_folderB.iloc[:, 5:350].mult
3
4 #add columns "new user", "similar user", "ssim", "array data file", "cell line"
5 drug_candidates_score_folderB = pd.concat([drug_candidates_generation_folderB.iloc[:,0
6 drug_candidates_score_folderB.head(2)

```

Out[24]:

	new user	similar user	ssim	
0	5500994158987071513209_E09.jpg	5500994157493061613625_C10.jpg	0.792672	55009941574930
1	5500994158987071513207_G11.jpg	5500994158987071513207_C09.jpg	0.814494	55009941589870

2 rows × 350 columns

### 3.4 Top-N recommendation list



In [0]:

```
1 #import folder B
2 folderB = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderB/"
3 folderB = list(folderB[0])
4 folderB = [(str(os.path.splitext(cell)[0])) for cell in folderB]
5
6
7 #create empty dictionary to store the top20 recommendations; key values correspond to
8 #e.g. folderB_top20_rec['5500994157493061613625_A07'] gives the top-20 recommendations
9 folderB_top20_rec = {}
10 row_index = 0
11
12 for new_user in folderB:
13
14     recommendation = drug_candidates_score_folderB.iloc[row_index, 5:350].sort_values(asc
15     top20 = recommendation.iloc[0:20]
16     folderB_top20_rec[new_user] = top20
17
18     row_index +=1
19
```

### 3.5 Evaluation - hit-rate & average reciprocal hit-rate

#### 3.5.1 Top-N hit-rate

In [0]:

```

1 #import folder B
2 folderB = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderB/"
3 folderB_cel = list(folderB[0]) #e.g. '5500994158987071513209_E09.cel'
4 folderB = [(str(os.path.splitext(cell)[0])) for cell in folderB_cel] # e.g. '550099415
5
6 #1) for every new user, retrieve the real top-20 drugs' response (from the drug response
7 df = drug_response_matrix_norm_withoutMissingValues
8
9 indexes = [] #empty list to store the row indexes (from drug response matrix) that belong
10 n=0
11
12 for new_user in folderB:
13     index = df[df["array data file"] == folderB_cel[n]].index.values.astype(int)[0]
14     indexes.append(index)
15     n += 1
16
17
18 df = df.loc[indexes, :] #retrieve the rows of the drug response matrix
19 df.head()
20
21
22 #2) create an empty dictionary to store the real top-20 ; key values correspond to new
23 #e.g. folderB_top20_real['5500994157493061613625_A07'] gives the real top-20 for the new
24 folderB_top20_real = {}
25 row_index = 0
26
27 for new_user in folderB:
28
29     recommendation = df.iloc[row_index, 2:347].sort_values(ascending=True)
30     top20 = recommendation.iloc[0:20]
31     folderB_top20_real[new_user] = top20
32
33     row_index +=1
34
35
36 #compute hit-rate
37 number_of_hits = 0 #a hit occurs when a drug appears on both folderB_top20_rec and folderB_top20_real
38
39 for new_user in folderB:
40     set_rec_drugs = set(folderB_top20_rec[new_user].index) #store recommended drugs into a set
41     set_real_drugs = set(folderB_top20_real[new_user].index) #store real/relevant drugs into a set
42     common_drugs = set_rec_drugs.intersection(set_real_drugs) #perform set intersection
43
44     number_of_hits = number_of_hits + len(common_drugs) #cumulative number of hits
45
46 hit_rate_B = number_of_hits/(len(folderB)) #hit_rate = (total number of hits in folderB) / (total number of new users)
47
48
49 print("Folder B hit-rate:", hit_rate_B)
50 print("(Number of recommended drugs:", len(set_rec_drugs), ")")
51

```

Folder B hit-rate: 12.64  
(Number of recommended drugs: 20 )

### 3.5.2 Average reciprocal hit-rate



In [0]:

```

1 #import folder B
2 folderB = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderB/"
3 folderB = list(folderB[0])
4 folderB = [(str(os.path.splitext(cell)[0])) for cell in folderB] # e.g. '55009941589870
5
6 weights = [1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/10, 1/11,
7            1/12, 1/13, 1/14, 1/15, 1/16, 1/17, 1/18, 1/19, 1/20]
8
9 total_sum_weights = 0 #store the weights'sum of all new users
10
11 for new_user in folderB:
12
13     #create dataframe "real top-20 drugs & weights"
14     real_list = list(folderB_top20_real[new_user].index)
15
16     df_real_weights = pd.DataFrame(weights).transpose()
17     df_real_weights.columns = real_list
18
19     #remove drugs/columns of dataframe "df_real_weights" that weren't recommended
20     rec_set = set(folderB_top20_rec[new_user].index)
21     real_set = set(folderB_top20_real[new_user].index)
22     not_recommended_drugs = real_set.difference(rec_set) #identify drugs in real_set tha
23     not_recommended_drugs = list(not_recommended_drugs)
24
25     df_real_weights = df_real_weights.drop(columns = not_recommended_drugs)
26
27     #compute ARHR
28     total_sum_weights = total_sum_weights + df_real_weights.sum().sum()
29
30     aver_recip_hit_rate_B = total_sum_weights/len(folderB)
31
32 print("Folder B average reciprocal hit-rate:", aver_recip_hit_rate_B)
33 print("(Maximum average reciprocal hit-rate:", "3.597739657143682")
34 #note: maximum average reciprocal hit-rate was computed in the notebook STAGE2_EXP2

```

Folder B average reciprocal hit-rate: 2.721471768169911  
(Maximum average reciprocal hit-rate: 3.597739657143682)

## 4 FOLDER C

### 4.1 Top-N similar cell lines

In [0]:

```

1 #replace the .cel extension by .jpg (in order to use the function similarity_ssim(image
2 folderC = [(str(os.path.splitext(cell)[0]) + ".jpg") for cell in folderC]
3 folderC[0]

```

Out[60]:

'5500994158987071513201\_G04.jpg'

In [0]:

```

1 for cell in folderC:
2     print("Analysing cell/image:", cell)
3
4     df_similarity, df_dissimilarity = similarity_ssim(cell)
5
6     cell_name = os.path.splitext(cell)[0] #removes the extension .jpg from the filename
7     cell_name = str(cell_name) + ".csv"
8
9     #top-50 similar images
10    similar_folderC_path = "/content/drive/My Drive/RecSys_Code/dataset/Celllines/folderC"
11    similar_path = os.path.join(similar_folderC_path, cell_name)
12    df_similarity.to_csv(similar_path, index = False, header=True) #export the dataframe
13
14    #top-10 dissimilar images
15    dissimilar_folderC_path = "/content/drive/My Drive/RecSys_Code/dataset/Celllines/folderC"
16    dissimilar_cell_name = "d" + cell_name
17    dissimilar_path = os.path.join(dissimilar_folderC_path, dissimilar_cell_name)
18    df_dissimilarity.to_csv(dissimilar_path, index = False, header=True) #export the dataframe
19

```

```

Analysing cell/image: 5500994158987071513201_G04.jpg
Number of database cell lines analysed so far: 10
Number of database cell lines analysed so far: 50
Number of database cell lines analysed so far: 100
Number of database cell lines analysed so far: 200
Number of database cell lines analysed so far: 300
Number of database cell lines analysed so far: 400
Number of database cell lines analysed so far: 500
Number of database cell lines analysed so far: 600
Number of database cell lines analysed so far: 700
Number of database cell lines analysed so far: 800

```

```

Analysing cell/image: 5500994158987071513209_G10.jpg
Number of database cell lines analysed so far: 10
Number of database cell lines analysed so far: 50
Number of database cell lines analysed so far: 100
Number of database cell lines analysed so far: 200
Number of database cell lines analysed so far: 300
Number of database cell lines analysed so far: 400
Number of database cell lines analysed so far: 500

```

## 4.2 Drug candidates generation

STEP 1: For every new user (i.e. cell line in folder C), retrieve the most similar user (top-1 similarity) and store that info into "df1" dataframe

In [0]:

```

1 #create an empty dataframe df1 with the structure: new user | similar user | ssim | array data file
2 columns= ['new user', 'similar user', 'ssim', 'array data file']
3 index = list(range(25))
4 df1 = pd.DataFrame(index = index, columns = columns)
5
6
7 #import folder C
8 folderC = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderC/"
9 folderC = list(folderC[0])
10 folderC = [(str(os.path.splitext(cell)[0])) for cell in folderC] #e.g. '55009941589870
11
12
13 folderC_experiment3_path = "/content/drive/My Drive/RecSys_Code/dataset/CellLines/fold
14
15
16 #for every new user (i.e. cell line in folder C), retrieve the most similar user (top-
17 #and store that info in the "df1" dataframe
18
19 n = 0
20
21 for cell in folderC:
22
23     cell_csv = cell + '.csv'
24     path_cell = os.path.join(folderC_experiment3_path, cell_csv)
25     df_cell = pd.read_csv(path_cell)
26
27     df1.loc[n].at['new user'] = cell + '.jpg'
28     df1.loc[n].at['similar user'] = df_cell.loc[1].at['image']
29     df1.loc[n].at['ssim'] = df_cell.loc[1].at['SSIM']
30
31     array_data_file = df_cell.loc[1].at['image'] #e.g. 5500994157493061613625_C10.jpg
32     array_data_file = str(os.path.splitext(array_data_file)[0]) + '.cel'
33     df1.loc[n].at['array data file'] = array_data_file
34     n += 1
35
36 df1.head(2)

```

Out[27]:

	new user		similar user	ssim	
0	5500994158987071513201_G04.jpg	5500994175999120813240_D04.jpg	0.779256	55009941759991	
1	5500994158987071513209_G10.jpg	5500994158987071513209_F07.jpg	0.827881	55009941589870	

STEP 2: For every similar user, retrieve the drugs' response and store that info into "df2" dataframe

In [0]:

```

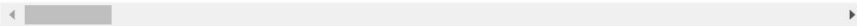
1 #store the similar users .cel filenames (e.g."5500994158987071513209_E12.cel") into a
2 similar_users = df1.loc[:, 'array data file'].tolist()
3
4
5 #retrieve rows, from drug response matrix, that belong to the similar users
6 df2 = drug_response_matrix_norm_withoutMissingValues
7
8 indexes = [] #empty list to store the row indexes (from drug response matrix) that belong
9 n=0         #the order of df1 is kept (i.e., the 1st index belongs to the 1st similar
10
11 for similar_user in similar_users:
12     index = df2[df2["array data file"] == similar_users[n]].index.values.astype(int)[0]
13     indexes.append(index)
14     n += 1
15
16
17 df2 = df2.loc[indexes, :] #retrieve the rows of the drug response matrix
18 df2 = df2.reset_index(drop=True)
19 df2 = df2.drop(columns = ['array data file'])
20
21 df2.head()

```

Out[28]:

	cell line	Erlotinib	Rapamycin	Sunitinib	PHA- 665752	MG-132	Paclitaxel	Cyclopamine	AZ66
0	RCH- ACV	0.50000	0.500000	0.500000	0.500000	0.500000	0.500000	0.50000	0.50000
1	CA46	0.57393	0.495945	0.555723	0.575169	0.552311	0.455882	0.59261	0.55344
2	JHH-7	0.50000	0.500000	0.500000	0.500000	0.500000	0.500000	0.50000	0.50000
3	NCI- H2122	0.50000	0.500000	0.500000	0.500000	0.500000	0.500000	0.50000	0.50000
4	PE- CA- PJ15	0.50000	0.500000	0.500000	0.500000	0.500000	0.500000	0.50000	0.50000

5 rows × 346 columns



STEP 3: Concatenate df1 and df2

In [0]:

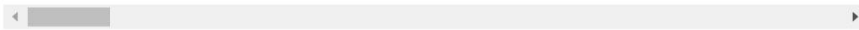
```
1 drug_candidates_generation_folderC = pd.concat([df1, df2], axis = 1)
2 drug_candidates_generation_folderC.head(25)
```

Out[29]:

		new user	similar user	ssim	
0	5500994158987071513201_G04.jpg	5500994175999120813240_D04.jpg	0.779256	5500994175999	
1	5500994158987071513209_G10.jpg	5500994158987071513209_F07.jpg	0.827881	5500994158987	
2	5500994172383112813929_H11.jpg	5500994172383112813928_F11.jpg	0.775788	5500994172383	
3	5500994172948120113978_D06.jpg	5500994175999120813240_F11.jpg	0.812921	5500994175999	
4	5500994157493061613625_F09.jpg	5500994158987071513201_B06.jpg	0.825241	5500994158987	
5	5500994158987071513207_A10.jpg	5500994172383112813928_F01.jpg	0.820293	5500994172383	
6	5500994175999120813240_D09.jpg	5500994175999120813240_B11.jpg	0.828107	5500994175999	
7	5500994172383112813928_C02.jpg	5500994158987071513207_A02.jpg	0.819471	5500994158987	
8	5500994173212120213068_D01.jpg	5500994172383112813929_B05.jpg	0.842599	5500994172383	
9	5500994173603120813304_H10.jpg	5500994158987071513207_F04.jpg	0.849867	5500994158987	
10	5500994172383112813930_F03.jpg	5500994158987071513201_D08.jpg	0.818987	5500994158987	
11	5500994172383112813930_G11.jpg	5500994158987071513207_G05.jpg	0.875754	5500994158987	
12	5500994172383112813928_D06.jpg	5500994173603120813304_D12.jpg	0.776809	5500994173603	
13	5500994157493061613625_A12.jpg	5500994172383112813929_C02.jpg	0.797552	5500994172383	
14	5500994158987071513207_D09.jpg	5500994158987071513201_C08.jpg	0.825477	5500994158987	
15	5500994172383112813930_C06.jpg	5500994172383112813928_F07.jpg	0.861852	5500994172383	
16	5500994158987071513201_C04.jpg	5500994158987071513207_B11.jpg	0.859242	5500994158987	
17	5500994158987071513209_F03.jpg	5500994175999120813240_A04.jpg	0.802041	5500994175999	
18	5500994158987071513209_E07.jpg	5500994158987071513209_D12.jpg	0.816018	5500994158987	
19	5500994172383112813928_A03.jpg	5500994158987071513207_C03.jpg	0.87062	5500994158987	
20	5500994175999120813240_C11.jpg	5500994158987071513207_B01.jpg	0.821722	5500994158987	
21	5500994158987071513202_B06.jpg	5500994158987071513202_B08.jpg	0.763047	5500994158987	
22	5500994158987071513201_E07.jpg	5500994172948120113978_A03.jpg	0.825634	5500994172948	
23	5500994158987071513207_H10.jpg	5500994172383112813929_C02.jpg	0.839188	5500994172383	
24	5500994158987071513209_E12.jpg	5500994172383112813928_A06.jpg	0.821691	5500994172383	



25 rows × 350 columns



In [0]:

```
1 #check sparsity (remember: IC50 values not available were replaced by 0.5)
2 sparsityC = drug_candidates_generation_folderC[drug_candidates_generation_folderC.iloc
3 sparsityC
```

Out[30]:

915

### 4.3 Drug candidates score

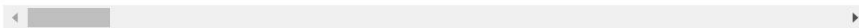
In [0]:

```
1 #SSIM x IC50s
2 drug_candidates_score_folderC = drug_candidates_generation_folderC.iloc[:, 5:350].mult
3
4 #add columns "new user", "similar user", "ssim", "array data file", "cell line"
5 drug_candidates_score_folderC = pd.concat([drug_candidates_generation_folderC.iloc[:,0
6 drug_candidates_score_folderC.head(2)
```

Out[31]:

		new user	similar user	ssim	
0	5500994158987071513201_G04.jpg	5500994175999120813240_D04.jpg	0.779256	55009941759991	
1	5500994158987071513209_G10.jpg	5500994158987071513209_F07.jpg	0.827881	55009941589870	

2 rows × 350 columns



### 4.4 Top-N recommendation list

In [0]:

```
1 #import folder C
2 folderC = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderC/"
3 folderC = list(folderC[0])
4 folderC = [(str(os.path.splitext(cell)[0])) for cell in folderC]
5
6
7 #create empty dictionary to store the top20 recommendations; key values correspond to
8 #e.g. folderC_top20_rec['5500994157493061613625_A07'] gives the top-20 recommendations
9 folderC_top20_rec = {}
10 row_index = 0
11
12 for new_user in folderC:
13
14     recommendation = drug_candidates_score_folderC.iloc[row_index, 5:350].sort_values(asc
15     top20 = recommendation.iloc[0:20]
16     folderC_top20_rec[new_user] = top20
17
18     row_index +=1
```

## 4.5 Evaluation - hit-rate & average reciprocal hit-rate

### 4.5.1 Top-N hit-rate

In [0]:

```

1 #import folder C
2 folderC = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderC/"
3 folderC_cel = list(folderC[0]) #e.g. '5500994158987071513209_E09.cel'
4 folderC = [(str(os.path.splitext(cell)[0])) for cell in folderC_cel] # e.g. '550099415
5
6 #1) for every new user, retrieve the real top-20 drugs' response (from the drug response
7 df = drug_response_matrix_norm_withoutMissingValues
8
9 indexes = [] #empty list to store the row indexes (from drug response matrix) that belo
10 n=0
11
12 for new_user in folderC:
13     index = df[df["array data file"] == folderC_cel[n]].index.values.astype(int)[0]
14     indexes.append(index)
15     n += 1
16
17
18 df = df.loc[indexes, :] #retrieve the rows of the drug response matrix
19 df.head()
20
21
22 #2) create an empty dictionary to store the real top-20 ; key values correspond to new
23 #e.g. folderC_top20_real['5500994157493061613625_A07'] gives the real top-20 for the ne
24 folderC_top20_real = {}
25 row_index = 0
26
27 for new_user in folderC:
28
29     recommendation = df.iloc[row_index, 2:347].sort_values(ascending=True)
30     top20 = recommendation.iloc[0:20]
31     folderC_top20_real[new_user] = top20
32
33     row_index +=1
34
35
36 #compute hit-rate
37 number_of_hits = 0 #a hit occurs when a drug appears on both folderC_top20_rec and fol
38
39 for new_user in folderC:
40     set_rec_drugs = set(folderC_top20_rec[new_user].index) #store recommended drugs into
41     set_real_drugs = set(folderC_top20_real[new_user].index) #store real/relevant drugs
42     common_drugs = set_rec_drugs.intersection(set_real_drugs) #perform set intersection
43
44     number_of_hits = number_of_hits + len(common_drugs) #cumulative number of hits
45
46 hit_rate_C = number_of_hits/(len(folderC)) #hit_rate = (total number of hits in folder
47
48
49 print("Folder C hit-rate:", hit_rate_C)
50 print("(Number of recommended drugs:", len(set_rec_drugs), ")")
51

```

Folder C hit-rate: 11.8  
(Number of recommended drugs: 20 )

#### 4.5.2 Average reciprocal hit-rate



In [0]:

```

1 #import folder C
2 folderC = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderC/"
3 folderC = list(folderC[0])
4 folderC = [(str(os.path.splitext(cell)[0])) for cell in folderC] # e.g. '55009941589870
5
6 weights = [1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/10, 1/11,
7            1/12, 1/13, 1/14, 1/15, 1/16, 1/17, 1/18, 1/19, 1/20]
8
9 total_sum_weights = 0 #store the weights'sum of all new users
10
11 for new_user in folderC:
12
13     #create dataframe "real top-20 drugs & weights"
14     real_list = list(folderC_top20_real[new_user].index)
15
16     df_real_weights = pd.DataFrame(weights).transpose()
17     df_real_weights.columns = real_list
18
19     #remove drugs/columns of dataframe "df_real_weights" that weren't recommended
20     rec_set = set(folderC_top20_rec[new_user].index)
21     real_set = set(folderC_top20_real[new_user].index)
22     not_recommended_drugs = real_set.difference(rec_set) #identify drugs in real_set tha
23     not_recommended_drugs = list(not_recommended_drugs)
24
25     df_real_weights = df_real_weights.drop(columns = not_recommended_drugs)
26
27     #compute ARHR
28     total_sum_weights = total_sum_weights + df_real_weights.sum().sum()
29
30     aver_recip_hit_rate_C = total_sum_weights/len(folderC)
31
32 print("Folder C average reciprocal hit-rate:", aver_recip_hit_rate_C)
33 print("(Maximum average reciprocal hit-rate:", "3.597739657143682")
34 #note: maximum average reciprocal hit-rate was computed in the notebook STAGE2_EXP2

```

Folder C average reciprocal hit-rate: 2.465275496089738  
 (Maximum average reciprocal hit-rate: 3.597739657143682

## 5 FOLDER D

### 5.1 Top-N similar cell lines

In [0]:

```

1 #replace the .cel extension by .jpg (in order to use the function similarity_ssim(image
2 folderD = [(str(os.path.splitext(cell)[0]) + ".jpg") for cell in folderD]
3 folderD[0]

```

Out[71]:

'5500994173212120213068\_F06.jpg'

In [0]:

```

1  for cell in folderD:
2      print("Analysing cell/image:", cell)
3
4      df_similarity, df_dissimilarity = similarity_ssim(cell)
5
6      cell_name = os.path.splitext(cell)[0] #removes the extension .jpg from the filename
7      cell_name = str(cell_name) + ".csv"
8
9      #top-50 similar images
10     similar_folderD_path = "/content/drive/My Drive/RecSys_Code/dataset/Celllines/folderD"
11     similar_path = os.path.join(similar_folderD_path, cell_name)
12     df_similarity.to_csv(similar_path, index = False, header=True) #export the dataframe
13
14     #top-10 dissimilar images
15     dissimilar_folderD_path = "/content/drive/My Drive/RecSys_Code/dataset/Celllines/folderD"
16     dissimilar_cell_name = "d" + cell_name
17     dissimilar_path = os.path.join(dissimilar_folderD_path, dissimilar_cell_name)
18     df_dissimilarity.to_csv(dissimilar_path, index = False, header=True) #export the dataframe

```

```

Analysing cell/image: 5500994173212120213068_F06.jpg
Number of database cell lines analysed so far: 10
Number of database cell lines analysed so far: 50
Number of database cell lines analysed so far: 100
Number of database cell lines analysed so far: 200
Number of database cell lines analysed so far: 300
Number of database cell lines analysed so far: 400
Number of database cell lines analysed so far: 500
Number of database cell lines analysed so far: 600
Number of database cell lines analysed so far: 700
Number of database cell lines analysed so far: 800

```

```

Analysing cell/image: 5500994158987071513202_G03.jpg
Number of database cell lines analysed so far: 10
Number of database cell lines analysed so far: 50
Number of database cell lines analysed so far: 100
Number of database cell lines analysed so far: 200
Number of database cell lines analysed so far: 300
Number of database cell lines analysed so far: 400
Number of database cell lines analysed so far: 500

```

## 5.2 Drug candidates generation

STEP 1: For every new user (i.e. cell line in folder D), retrieve the most similar user (top-1 similarity) and store that info into "df1" dataframe

In [0]:

```

1 #create an empty dataframe df1 with the structure: new user | similar user | ssim | array data file
2 columns= ['new user', 'similar user', 'ssim', 'array data file']
3 index = list(range(25))
4 df1 = pd.DataFrame(index = index, columns = columns)
5
6
7 #import folder D
8 folderD = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderD/"
9 folderD = list(folderD[0])
10 folderD = [(str(os.path.splitext(cell)[0])) for cell in folderD] #e.g. '55009941589870
11
12
13 folderD_experiment3_path = "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderD_experiment3/"
14
15
16 #for every new user (i.e. cell line in folder D), retrieve the most similar user (top-1)
17 #and store that info in the "df1" dataframe
18
19 n = 0
20
21 for cell in folderD:
22
23     cell_csv = cell + '.csv'
24     path_cell = os.path.join(folderD_experiment3_path, cell_csv)
25     df_cell = pd.read_csv(path_cell)
26
27     df1.loc[n].at['new user'] = cell + '.jpg'
28     df1.loc[n].at['similar user'] = df_cell.loc[1].at['image']
29     df1.loc[n].at['ssim'] = df_cell.loc[1].at['SSIM']
30
31     array_data_file = df_cell.loc[1].at['image'] #e.g. 5500994157493061613625_C10.jpg
32     array_data_file = str(os.path.splitext(array_data_file)[0]) + '.cel'
33     df1.loc[n].at['array data file'] = array_data_file
34     n += 1
35
36 df1.head(2)

```

Out[35]:

	new user		similar user	ssim	
0	5500994173212120213068_F06.jpg	5500994158987071513201_G08.jpg	0.815748	55009941589870	
1	5500994158987071513202_G03.jpg	5500994175999120813240_A06.jpg	0.806401	55009941759991	

STEP 2: For every similar user, retrieve the drugs' response and store that info into "df2" dataframe

In [0]:

```

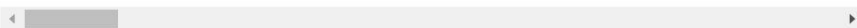
1 #store the similar users .cel filenames (e.g."5500994158987071513209_E12.cel") into a
2 similar_users = df1.loc[:, 'array data file'].tolist()
3
4
5 #retrieve rows, from drug response matrix, that belong to the similar users
6 df2 = drug_response_matrix_norm_withoutMissingValues
7
8 indexes = [] #empty list to store the row indexes (from drug response matrix) that belong
9 n=0         #the order of df1 is kept (i.e., the 1st index belongs to the 1st similar
10
11 for similar_user in similar_users:
12     index = df2[df2["array data file"] == similar_users[n]].index.values.astype(int)[0]
13     indexes.append(index)
14     n += 1
15
16
17 df2 = df2.loc[indexes, :] #retrieve the rows of the drug response matrix
18 df2 = df2.reset_index(drop=True)
19 df2 = df2.drop(columns = ['array data file'])
20
21 df2.head()

```

Out[36]:

	cell line	Erlotinib	Rapamycin	Sunitinib	PHA-665752	MG-132	Paclitaxel	Cyclopamine	AZ628	Sor
0	NCI-H1568	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	
1	DBTRG-05MG	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	
2	CAL-62	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	
3	PANC-10-05	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	
4	SK-MEL-28	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	

5 rows × 346 columns



STEP 3: Concatenate df1 and df2

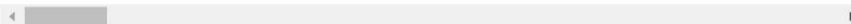
In [0]:

```
1 drug_candidates_generation_folderD = pd.concat([df1, df2], axis = 1)
2 drug_candidates_generation_folderD.head(25)
```

Out[37]:

		new user	similar user	ssim	
0	5500994173212120213068_F06.jpg	5500994158987071513201_G08.jpg	0.815748	5500994158987	
1	5500994158987071513202_G03.jpg	5500994175999120813240_A06.jpg	0.806401	5500994175999	
2	5500994158987071513202_A06.jpg	5500994175999120813240_C08.jpg	0.817187	5500994175999	
3	5500994158987071513201_E06.jpg	5500994158987071513201_B05.jpg	0.841482	5500994158987	
4	5500994158987071513207_E05.jpg	5500994158987071513207_H10.jpg	0.836718	5500994158987	
5	5500994172383112813928_C03.jpg	5500994173212120213068_B04.jpg	0.81978	5500994173212	
6	5500994158987071513202_C01.jpg	5500994158987071513209_H10.jpg	0.779388	5500994158987	
7	5500994157493061613625_G07.jpg	5500994172383112813930_C07.jpg	0.869738	5500994172383	
8	5500994158987071513201_C05.jpg	5500994172383112813928_B01.jpg	0.800671	5500994172383	
9	5500994158987071513201_D06.jpg	5500994173212120213068_B01.jpg	0.81495	5500994173212	
10	5500994172383112813929_A09.jpg	5500994158987071513201_B06.jpg	0.824486	5500994158987	
11	5500994158987071513201_D07.jpg	5500994172383112813928_E01.jpg	0.794525	5500994172383	
12	5500994172948120113978_C03.jpg	5500994173603120813304_F10.jpg	0.799984	5500994173603	
13	5500994157493061613625_B01.jpg	5500994157493061613625_C08.jpg	0.837031	5500994157493	
14	5500994172383112813928_C07.jpg	5500994172383112813929_G03.jpg	0.777451	5500994172383	
15	5500994157493061613625_G02.jpg	5500994172383112813928_F12.jpg	0.843923	5500994172383	
16	5500994158987071513209_B12.jpg	5500994158987071513202_B02.jpg	0.81763	5500994158987	
17	5500994158987071513201_E11.jpg	5500994158987071513201_B06.jpg	0.822841	5500994158987	
18	5500994172383112813928_G11.jpg	5500994172383112813928_F01.jpg	0.819031	5500994172383	
19	5500994158987071513202_C06.jpg	5500994158987071513201_D11.jpg	0.829996	5500994158987	
20	5500994172383112813929_F02.jpg	5500994172383112813929_G03.jpg	0.789264	5500994172383	
21	5500994158987071513201_C12.jpg	5500994172948120113978_G05.jpg	0.748813	5500994172948	
22	5500994173212120213068_D05.jpg	5500994158987071513207_C09.jpg	0.819187	5500994158987	
23	5500994158987071513202_A01.jpg	5500994173603120813304_H10.jpg	0.812143	5500994173603	
24	5500994157493061613625_F04.jpg	5500994158987071513207_C06.jpg	0.792147	5500994158987	

25 rows × 350 columns





In [0]:

```

1 #check sparsity (remember: IC50 values not available were replaced by 0.5)
2 sparsityD = drug_candidates_generation_folderD[drug_candidates_generation_folderD.iloc[
3 sparsityD

```

Out[38]:

1252

### 5.3 Drug candidates score

In [0]:

```

1 #SSIM x IC50s
2 drug_candidates_score_folderD = drug_candidates_generation_folderD.iloc[:, 5:350].mult
3
4 #add columns "new user", "similar user", "ssim", "array data file", "cell line"
5 drug_candidates_score_folderD = pd.concat([drug_candidates_generation_folderD.iloc[:,0
6 drug_candidates_score_folderD.head(2)

```

Out[39]:

	new user	similar user	ssim
0	5500994173212120213068_F06.jpg	5500994158987071513201_G08.jpg	0.815748
1	5500994158987071513202_G03.jpg	5500994175999120813240_A06.jpg	0.806401

2 rows × 350 columns

### 5.4 Top-N recommendation list

In [0]:

```
1 #import folder D
2 folderD = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderD/"
3 folderD = list(folderD[0])
4 folderD = [(str(os.path.splitext(cell)[0])) for cell in folderD]
5
6
7 #create empty dictionary to store the top20 recommendations; key values correspond to
8 #e.g. folderD_top20_rec['5500994157493061613625_A07'] gives the top-20 recommendations
9 folderD_top20_rec = {}
10 row_index = 0
11
12 for new_user in folderD:
13
14     recommendation = drug_candidates_score_folderD.iloc[row_index, 5:350].sort_values(asc
15     top20 = recommendation.iloc[0:20]
16     folderD_top20_rec[new_user] = top20
17
18     row_index +=1
```

## 5.5 Evaluation - hit-rate & average reciprocal hit-rate

### 5.5.1 Top-N hit rate

In [0]:

```

1 #import folder D
2 folderD = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderD/"
3 folderD_cel = list(folderD[0]) #e.g. '5500994158987071513209_E09.cel'
4 folderD = [(str(os.path.splitext(cell)[0])) for cell in folderD_cel] # e.g. '550099415
5
6 #1) for every new user, retrieve the real top-20 drugs' response (from the drug response
7 df = drug_response_matrix_norm_withoutMissingValues
8
9 indexes = [] #empty list to store the row indexes (from drug response matrix) that belong
10 n=0
11
12 for new_user in folderD:
13     index = df[df["array data file"] == folderD_cel[n]].index.values.astype(int)[0]
14     indexes.append(index)
15     n += 1
16
17
18 df = df.loc[indexes, :] #retrieve the rows of the drug response matrix
19 df.head()
20
21
22 #2) create an empty dictionary to store the real top-20 ; key values correspond to new
23 #e.g. folderD_top20_real['5500994157493061613625_A07'] gives the real top-20 for the new
24 folderD_top20_real = {}
25 row_index = 0
26
27 for new_user in folderD:
28
29     recommendation = df.iloc[row_index, 2:347].sort_values(ascending=True)
30     top20 = recommendation.iloc[0:20]
31     folderD_top20_real[new_user] = top20
32
33     row_index +=1
34
35
36 #compute hit-rate
37 number_of_hits = 0 #a hit occurs when a drug appears on both folderD_top20_rec and folderD_top20_real
38
39 for new_user in folderD:
40     set_rec_drugs = set(folderD_top20_rec[new_user].index) #store recommended drugs into set
41     set_real_drugs = set(folderD_top20_real[new_user].index) #store real/relevant drugs into set
42     common_drugs = set_rec_drugs.intersection(set_real_drugs) #perform set intersection
43
44     number_of_hits = number_of_hits + len(common_drugs) #cumulative number of hits
45
46 hit_rate_D = number_of_hits/(len(folderD)) #hit_rate = (total number of hits in folderD) / (total number of new users)
47
48
49 print("Folder D hit-rate:", hit_rate_D)
50 print("(Number of recommended drugs:", len(set_rec_drugs), ")")
51

```

Folder D hit-rate: 12.2  
(Number of recommended drugs: 20 )

### 5.5.2 Average reciprocal hit-rate



In [0]:

```

1 #import folder D
2 folderD = pd.read_csv( "/content/drive/My Drive/RecSys_Code/dataset/CellLines/folderD/"
3 folderD = list(folderD[0])
4 folderD = [(str(os.path.splitext(cell)[0])) for cell in folderD] # e.g. '55009941589870
5
6 weights = [1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/10, 1/11,
7            1/12, 1/13, 1/14, 1/15, 1/16, 1/17, 1/18, 1/19, 1/20]
8
9 total_sum_weights = 0 #store the weights'sum of all new users
10
11 for new_user in folderD:
12
13     #create dataframe "real top-20 drugs & weights"
14     real_list = list(folderD_top20_real[new_user].index)
15
16     df_real_weights = pd.DataFrame(weights).transpose()
17     df_real_weights.columns = real_list
18
19     #remove drugs/columns of dataframe "df_real_weights" that weren't recommended
20     rec_set = set(folderD_top20_rec[new_user].index)
21     real_set = set(folderD_top20_real[new_user].index)
22     not_recommended_drugs = real_set.difference(rec_set) #identify drugs in real_set tha
23     not_recommended_drugs = list(not_recommended_drugs)
24
25     df_real_weights = df_real_weights.drop(columns = not_recommended_drugs)
26
27     #compute ARHR
28     total_sum_weights = total_sum_weights + df_real_weights.sum().sum()
29
30     aver_recip_hit_rate_D = total_sum_weights/len(folderD)
31
32 print("Folder D average reciprocal hit-rate:", aver_recip_hit_rate_D)
33 print("(Maximum average reciprocal hit-rate:", "3.597739657143682")
34 #note: maximum average reciprocal hit-rate was computed in the notebook STAGE2_EXP2

```

Folder D average reciprocal hit-rate: 2.401255092516702  
(Maximum average reciprocal hit-rate: 3.597739657143682)

## 6 Overall/Final Results

In [0]:

```

1 hit_rate_average = (hit_rate_B + hit_rate_C + hit_rate_D)/3
2 hit_rate_average

```

Out[43]:

12.213333333333333

In [0]:

```

1 arht_average = (aver_recip_hit_rate_B + aver_recip_hit_rate_C + aver_recip_hit_rate_D)
2 arht_average

```

Out[44]:

2.529334118925451