Polytechnic University of Catalonia

Barcelona School of Informatics

Department of Computer Science

A workflow for the analysis and modelling of high-dimensional omics data

Author: Gerard Otero Martín

Director: Luis Antonio Belanche Muñoz

September 2019

Abstract

In the recent years, advances in the field of Computer Science have made Machine Learning a viable method of obtaining very precise information and making great predictions in studies related to the entire field of biology.

In this project we create a fast and accurate Machine Learning workflow with an easy and user-friendly interface, many customizable parameters, informing plots and feedback, and several data input methods, using the latest available free technologies.

Acknowledgments

First of all I would like to thank my director, Luis Antonio Belanche Muñoz, for his support throughout the entire process and the help he has brought me to be able to successfully develop this entire project.

I would also like to thank my family for supporting me during the last six years and giving me an exemplary education, and the friends and workmates that have been working with me all these past years helping me learn new things every day.

Lastly, I would like to thank Kiara for all her love and support during the development of this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

For a long time, humans have been trying to understand the full nature of organic life. Many advances have been made and we have a good understanding of a very big amount of different data and facts. However, the further biology advances, the harder it is to understand the data we possess. In order to comprehend the behavior of cells, organs, and a whole organism at molecular level, we need to study omics data, which are responsible for the translation of biological molecules into the structure, function and dynamics of an organism.

Thanks to the latest advances in Computer Science, there has been a big development in the field of Machine Learning, allowing computers to find solutions to problems much faster than a human being would.

In this project, we aim to use Machine Learning techniques and algorithms in order to create a software capable of reducing a set of omics data exponentially to a much smaller subset of them, leaving only the most related data to the differentiation of a sample between two known classes.

1.1 Formalization of the problem

The problems this project aims to solve are high-dimensional data analysis, and the difficulty of having to build and test that data analysis software by oneself. Although the final software developed in this project will be able to work with any kinds of high-dimensional data, this project is designed to analyze omics data. After inputting the data the user possesses, the output should be a whole set of data indicating the entire process those data have gone through, and graphics and explanations of the results at the end of it and during each one of its parts. The software must also be accurate, easy to use and ready for the user to use it at any moment.

1.2 Objectives

In this section there are defined the different objectives and goals that, with this project, I aim to achieve and complete:

- First of all, I need to acquire knowledge about machine learning, omics and basically every subject that could have relationship with the next steps to develop, since having a good base of knowledge is essential.
- I want to develop a workflow able to reduce a set of variables of a problem to a much smaller set of them that contains the most relevant ones to the problem the user is analyzing.
- I want to develop an interface to guide the user through the entire analysis process, giving them feedback and allowing them to customize the parameters of every step.

- Finally, I need to validate the project using data from other studies in order to test the accuracy of the results and the execution time.

1.3 Actors involved

In this section there are listed the different actors involved in the project, and what their role on it is.

1.3.1 Developer

The developer will be the person in charge of developing the software. That means their job will be to investigate on Machine Learning, collect information, techniques and algorithms, find tools to implement them, build the workflow, develop the user interface, build the feedback and results reports, and test the performance of the complete software. I will be the person in charge of performing all of these tasks.

1.3.2 Project director

The project director will be the person who will guide the developer through the different problems he might face, give advice on how to continue when reaching certain milestones and, in general, making sure the project is finished and working in time. Luis Antonio Belanche Muñoz will be the director of this project.

1.3.3 Beneficiaries

The beneficiaries will be the people who will make use of this project. It can be the scientist community, having a fast and already available way to obtain results from certain data without having to go through the trouble of finding the tools and building the software themselves; it can be students or people who are starting to work with machine learning, to learn about different ways to analyze and process data; or it could be curious people who just want to play around with some data and see if they find something interesting.

1.4 State of the art

Machine learning is the branch of artificial intelligence that studies algorithms and statistical models that computer systems use to be able to perform tasks and solve problems without explicitly programming them for that purpose, but building a mathematical model of sample data, known as training data, in order to make predictions or decisions when computing the data from real cases.

In the last years, big improvements in machine learning have been made. It is everywhere in our daily lives, in the camera of our mobile phone[2], in our e-mail's inbox[3], nowadays even self-driving cars are being made[4].

However, different cases require different machine learning techniques and algorithms, as not all of them are useful for each problem. Omics refers to a field of biology that ends in -omics, such as genomics, and omics data is data obtained studying these fields. With the new technologies, we can extract much more information from these sources[5][6]. Because of this, the amount of features, attributes and variables collected from each sample has increased exponentially[7]. This makes omics data become a high-dimensional environ-

ment, and so is the data we will use in this project, since they belong to the field of genomics.

According to recent researches in the field, when dealing with high-dimensional data, researchers first have to take care of the high complexity of the data, being crucial to reduce it. Because of this, a Feature Selection (FS) approach becomes crucial and non trivial. FS is the process of selecting a subset of relevant features from a data model to construct a new and reduced one. This provides a deeper insignt into the underlying processes that are the foundation of the data, improves the performance of the computation of the machine learning step, produces better model results avoiding overfitting and avoids the curse of dimensionality[9], which describes problems that arise when analyzing and organizing data in high-dimensional spaces that do not happen in low-dimensional spaces, for example the increase and then decrease of the predictive power of a classifier as the number of dimensions/features used increases with a fixed amount of training data, also known as Hughes phenomenon[10]. However, using FS requires knowing if there are redundant features in the dataset that are irrelevant for the study of it.

The most common way to deal with the FS problem is to use a variable ranking method to remove the least promising variables[11], known as filter methods. Although these methods have been broadly used in computational biology for cancer classification using microarray data[12], correlation filters could remove features that are irrelevant by themselves but become useful in combination. To solve this problem, new algorithms were proposed to combine the original variables into a new and smaller subset of features, for example Principal Component Analysis (PCA) which, according to Ringner[32], can reduce the dimensionality of the samples to at most the number of samples without losing information. PCA methods can reduce

the number of variables by looking into the dependencies without considering the final learning model.

In 1997, a stragegy combining a FS algorithm with a learning/classification appeared. It is called wrapper methods. Wrapper methods evaluate subsets of variables which allows to detect the possible interactions between variables[14]. However, these methods have two disadvantages: when the number of observations is insufficient, there is a risk of increasing overfitting; when the number of variables is large, the computation time increases.

A new technique that tried to combine the advantages of both previous methods, called embedded methods, was proposed. Embedded methods combine feature selection with the learning process, reducing the dataset and classificating at the same time[15]. These methods are less computationally intensive than wrapper methods, but they are specific to one learning machine. The most common technique are decision tree algorithms, which select a feature in each recursive step of the tree growth, and partition the initial set into subsets based on a value test with the selected feature. This process is repeated until a new partition adds no value to the predictions.

Clustering methods have also been proposed as gene selection techniques. Most FS techniques assume that the features are independent. However, it is know that this is untrue, since genes interact with each other. Clustering methods cluster together a subset of strongly related features as a new unique feature, reducing the amount of variables and also being able to improve the performance of the classifiers that will later use this data[16].

In order to analyze data, Principal Components Analysis (PCA) was invented in 1901 by Karl Pearson[32]. It is a procedure that converts a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called "principal components". PCA first tries

to maximize the variance of the first component, and then tries to maximize the variance of the rest under the constraint that they are orthogonal to the previous one.

We also have Partial Least Squares regression (PLS), which is an extension of the Multiple Linear Regression model, and has gained much attention in the analysis of high dimensional genomic data[17].

Discriminant Analysis (DA) is a statistical tool that aims to assess the adequacy of a classification. In 1930, 3 different people (Fisher, Hoteling and Mahalanob) were trying to solve the same problem via 3 different methods. Later, their methods would combine to devise what is today called Discriminant Analysis. It is widely used to determine which predictor variables are related to the dependant variable and to predict the value of it given certain values of the predictor variables. Its basis is also the Multiple Linear Regression model.

PLS was not originally designed for classification and discrimination problems, but has often been used for that purpose[18][19]. This way, PLS and DA have been used together as PLS-DA and, more recently, as sparse PLS-DA (sPLS-DA), allowing to perform variable classification and selection in one step, with multiple classes and unifying data from multiple datasets[31].

1.5 Scope

The scope of this project is pretty wide. The main purpose of it is to offer a Machine Learning tool to analyze a set of high-dimensional data. This tool is made for the purpose of analyzing omics data, but it works as well for any kind of high-dimensional data. Since it is a generalized tool for any kinds of data and studies, the results it outputs might not be as good as other more

specific tools and workflows. However, it still offers fast and accurate results to start a new investigation with.

This tool will offer three steps in its workflow: a fast filter for a quick removal of a high percentage of the total variables, a slower but more accurate one which will be optional, and a last step of variable analysis and selection. Each one of these steps will offer several parameters that the user will be able to freely customize, also with graphical feedback. At the end of the workflow, the user will be able to download a pdf report of the entire process, showing all the data and graphs the user wants.

This tool will be served as a web service, which means that it will be used via an internet explorer, regardless of it being run locally or online.

1.6 Obstacles

There are many possible obstacles to face while developing this project:

First of all, The tools we will be using to write the code are brand new for us, including the code itself. Because of this, a period of practise and familiarization will be needed in order to get used to it. However, since it is a software project, we still expect to find new peculiarities and little adversities every day.

Another problem is that the subject of Machine Learning, although familiar, is new for me in the context of development. Because of this it will be very important to read as much as possible about every step of the process in order to offer a tool fully devised under a solid knowledge of what it covers.

However, the main problem will be time. It will take time to properly use the language, the tools and the methods and algorithms. Also, as previously stated, to retrieve all the information we need about all of them. The

processing of the data in the developed software is relatively slow too, and in case it has not been properly coded, the time might increase exponentially, leading to more code fixing and optimization.

1.7 Methodology

We will develop this project using Scrum methodology, since it is ideal for projects with high flexibility, where the requirements are rapidly changing and needing regular reviews and feedback from the director.

1.7.1 Reviews and meetings

Since intensive feedback from the professor in charge of the project will be very important because of the problems or doubts that might be arised from the development process of it, biweekly meetings will be held, apart from contacting via email every time it is needed. This way we assure good communication and a nice way to review the completed work, solve the problems we might encounter, and clear the path for our next steps.

1.7.2 Development tools

For the development of the code we have selected the R language. It is a very popular programming language, widely used among statisticians and data miners for developing statistical software and data analysis. Since machine learning directly involves both statistics and, of course, data analysis, it is also used to develop software and tools in this field, and has many packages with algorithms and other functions that we might need. We will also use RStudio, the official IDE for R, which provides a document editor, an R command line, and a plot viewer, among other features.

To save, version and manage the code we will use GitHub.

The rest of materials, such as data to test the software, will be provided by the director.

1.7.3 Validation

In order to validate our project, we need to compare our results to the results or conclusions of a study that analyzes the same data. We can not compare the results of all the studies in the world, so this validation will not be 100% accurate, but we can test a small set.

Also, since a relevant part of this project will be the presentation, meaning the user interface, the director and the developer will both test the usability of the software, suggesting and implementing the necessary changes in order to make the user experience as satisfying as possible.

Chapter 2

Design and implementation

2.1 Summary

In this chapter we explain the design and the implementation of both the workflow and the user interface. We will explain how we did it and why we took certain decisions along the process.

2.2 Workflow

In this section we explain the design and implementation of the algorithms and techniques in the workflow, without going into detail about the user interface and graphical visualization (that part will be left for a later section).

2.2.1 Design

Since the problem we are trying to solve deals with omics data, it is designed to take into consideration certain peculiarities they possess, such as many relationships between variables. However, this workflow is able to work with any kinds of data. The only requirement is that each and every sample of

those data must belong to one and only one of two classes and this information must be inside the inputted dataset.

The datasets can be input in two formats: csv (comma separated values) and RData. The reason for this is because csv is a widely acknowledged format for managing data, with just the right amount of information: values of a row separated by commas, and rows separated by a like break[21]. Many languages include tools for writing and reading csv files, R included. Because of this, it is very probable that many datasets are stored in csv files, or that people know how to write them in one from any source, and we can immediately receive the dataset and process the data. We also accept RData because it is the native way R has to manage data, so if somebody happens to have a dataset saved in an RData file we can also process it more easily than a csv file.

This workflow will consist of three parts, each one of them is increasingly complex, slow and accurate, all of it ending in a final result with the last selected features.

Figure 2.1: Diagram of the workflow

The workflow parts are the following:

Fisher scoring

First we have Fisher scoring. This is a very fast algorithm, but it is inaccurate because the feature evaluation it performs is univariate, which means that it does not take into consideration the relationships that might be present between variables, which are actually characteristic of omics data. It also

does not handle feature redundancy, which means that if two variables provide the same information, either relevant or not, this algorithm will not give any of them a special treatment and will be always executed the same way. There is a more advanced algorithm, called Generalized Fisher Scoring[22], which overcomes redundancy trying to jointly select features. However, we chose not to use this algorithm because, although it proves to be useful processing certain kinds of data, it still does not solve the relationships between variables problem, and adds more complexity to the step. Thus, in order to keep it simple and quick, we use the standard algorithm.

Fisher Scoring relies on the intuition that features with high quality should assign similar values to instances in the same class and different values to instances from different classes.

This way, the score of the i-th feature S_i will be calculated as:

$$S_i = \frac{|\overline{X}_i^{(0)} - \overline{X}_i^{(1)}|}{\sqrt{n_0 \times var(X_i)^{(0)} + n_1 \times var(X_i)^{(1)}}}$$

where $\overline{X}_i^{(y)}$ is the mean of the values of the variable i in the samples of the class y, n_y is the amount of samples of the class y and $var(X_i)^{(y)}$ is the variance of the values of the variable i among all the samples of the class y.

Right after calculating the score for every single one of the features in the dataset, the workflow will keep for the next steps only the amount of variables chosen by the user, expecting to be keeping the x more informative out of them. However, as explained before, it is an univariate algorithm and, for this reason, it needs to be the first step the data goes through, as a way to quickly get rid of the biggest amount of individually irrelevant data. And ease the execution of the next steps.

This step will always be executed, and the user will be able to keep any positive amount of variables for the next steps.

ReliefF

In the middle part of our workflow we have ReliefF. This one is also a filtering algorithm, but it is more complex as it introduces a series of improvements to the calculation of the variables quality. This is an optional step that the user can select if they want to perform or not. There are reasons why they would want to use it, explained in the next lines, and reasons why they would not, such as the extra execution time of the workflow. Either way, we let the user choose.

ReliefF belongs to the Relief family of algorithms. The majority of heuristic measures for estimating the quality of the attributes, such as Fisher Scoring, assume the conditional independence of the attributes. However, Relief algorithms do not make this assumption. This way, they are efficient, aware of the contextual information, and can correctly estimate the quality of attributes in problems with strong dependencies and interactions between attributes, such as ours[24].

The basic Relief algorithm is as follows: assuming that samples $I_1, I_2, ..., I_n$ in the instance space are described by a vector of attributes $A_i, i = 1, ..., a$, where a is the number of explanatory attributes, and are labeled with the target value τ_j

Algorithm 1 Basic Relief algorithm

1: Set all weights $W[A] := 0.0$;
2: **for** $i := 1$ **to** m **do**
3: randomly select an instance R_i;
4: find nearest hit H and nearest miss M;
5: **for** $A := 1$ **to** a **do**
6: $W[A] = W[A] - \text{diff}(A,R_i,H)/m + \text{diff}(A,R_i,M)/m$;
7: **end for**
8: **end for**

The key idea[25] is that, given a randomly selected instance R_i, Relief searches for the two nearest neighbors: one from the same class (H), and one from the different class (M). It updates the quality estimation $W[A]$ for all attributes A depending on their values for R_i, M and H. If instances R_i and H have different values of the attribute A then the attribute A separates two instances with the same class which is not desirable so we decrease the quality estimation $W[A]$. On the other hand if instances R_i and M have different values of the attribute A then the attribute A separates two instances with different class values which is desirable so we increase the quality estimation $W[A]$. Then, this whole process is repeated m times. Function diff(A, I_1, I_2) calculates the difference between the values of the attribute A for two instances I_1 and I_2. For numerical attributes, which is our case, it is calculated as:

$$\text{diff}(A, I_1, I_2) = \frac{|value(A, I_1) - value(A, I_2)|}{max(A) - min(A)}$$

Two years later, an extension called ReliefF is created based on the original Relief. It is similar to it, but adds a few improvements, some that do not matter in our context, such as not being limited to two class problems

or that can deal with incomplete data, since the workflow will not work with data with these features; and some that do improve it for our problem, such as being more robust and, most importantly, searching for k of its nearest neighbors instead of just one. This algorithm is as follows:

Algorithm 2 ReliefF algorithm

Set all weights $W[A] := 0.0$;
for $i := 1$ **to** m **do**
    randomly select an instance R_i;
    find k nearest hits H_j;
    for all class $C \neq class(R_i)$ **do**
        from class C find k nearest misses $M_j(C)$;
    end for
    for $A := 1$ **to** a **do**
        $W[A] = W[A] - \sum_{j=1}^{k} \text{diff}(A, R_i, H_j)/(m \cdot k) +$
$\sum_{C \neq class(R_i)} [\frac{P(C)}{1-P(class(R_i))} \sum_{j=1}^{k} \text{diff}(A, R_i, M_j(C))]/m \cdot k$;
    end for
end for

Similarly to Relief, it selects a random instance R_i, but then searches for k of its nearest neighbors of the same class and k nearest neighbors of each different class. It updates the quality estimation $W[A]$ depending on their values for R_i, hits H_j and misses $M_j(C)$. The update formula is similar to the one in Relief, except that ReliefF averages the contribution of all the hits and the misses. The contribution for each class of the misses is weighted with the prior probability of that class $P(C)$ (which will always be 1 for our case since we will only have two classes problems). Then again, the algorithm repeats itself m times. To deal with incomplete data, the *diff* changes and treats missing values probabilistically. However, we can leave this part out of

the explanation because our workflow does not accept datasets with missing data.

This way, using this algorithm, we can let the user customize the number k of nearest neighbors, and the number m of repetitions. Then, like in the first step, the user inputs the amount of variables that wants to keep after this one and discards the rest for the last step.

sPLS-DA

The last step is the most important one. In this step we will make use of sparse PLS-DA, a variant of PLS-DA that performs variable selection and classification in one step. This workflow is for data analysis and not prediction, and for this reason we do not care about the classification part, but we do want the variable selection part as a way to reduce even more the amount of variables returned by the entire process.

First we have to talk about the origins of it. Partial Least Squares (PLS) was not originally designed for classification and discrimination problems, but it has often been used for that purpose[26][27]. Basically, PLS will try to find the multidimensional direction in the X space that explains the maximum multidimensional variance direction in the Y space. PLS regression is particularly suited when the matrix of predictors has more variables than observations, which is the case in high-dimensional data, and when there is multicollinearity among X values, meaning that one variable in a multiple regression model can be linearly predicted from the others with a substantial degree of accuracy, which is also the case in omics data due to the interactions among the variables that characterize them. Then, more recently, Partial Least Squares-Discriminant Analysis (PLS-DA) was developed. This new approach is a variant of PLS that can be used when the response variable

Y is categorical. This technique is specially suited to deal with models with many more predictors than observations, and with multicollinearity[28][29], so this technique is perfect for our project. It's important to say, though, that PLS-DA performs much better when the dataset contains a small amount of significant variables in comparison to noisy ones, and the more clustered the classes are, the better PLS-DA will be able to select features[30].

PLS-DA works as follows: assume X is a $n \times m$ matrix and y is the class label vector, the principal components of PLS-DA can be formulated as the eigenvectors of the non-singular portion of the covariance matrix C, given by:

$$C = \frac{1}{(n-1)^2} X^T C_n y y^T C_n X$$

The iterative process computes the loading vectors $a_1, ..., a_d$, which give the importance of each feature in that component. In iteration h, PLS-DA has the following objective:

$$\max_{(a_h, b_h)} \text{cov}(X_h a_h, y_h b_h)$$

where b_h is the loading for the label vector y_h, $X_1 = X$ and X_h and y_h are the error matrices after transforming with the previous $h-1$ components.

Sparse PLS-DA (sPLS-DA) is just a variant of PLS-DA that makes a sparsity assumption, which means that assumes that only a subset of the variables are responsible for driving the biological event or effect that is being measured. sPLS-DA has been shown to be successful with applications where the number of features far outnumbers the number of samples[31]. Thus, sPLS-DA not only calculates the importance of each variable, but it is also able to select the subset of the variables that it believes are enough to correctly separate all the samples between all the classes they belong to.

This step will be mandatory and after executing it, with all the parameters inputted by the user, the workflow will return the last results, which will be the variables that sPLS-DA selected, along with their loadings vector. Also, it needs to return proof of that the variables selected and their loadings are actually good enough to discriminate between the data between the two classes they belong to.

2.2.2 Implementation

In this section, the implementation of each part of the project will be explained, allowing to have an understanding on why certain decisions were taken and what are the results that the workflow returns.

This workflow makes use of the R packages `CORElearn`[33] and `mixOmics`[34]

Before starting with the explanation of each part, it is worth mentioning that, in order to make the project clearer for me and other developers that might want to look deeper into the code, as well as to avoid repeated code, I encapsulated the implementation of each part into a different function. This way, in order to apply one of the parts, only the function needs to be executed. Also, to solve certain problems caused by the user interface, I added a `debug` parameter to each function, that allows us to see more data about the calculations made and certain plots that, unless exclusive to the workflow part, will be shown as examples in the next section where we detail the design and implementation of the user interface. Its use will be detailed when explaining each function.

The functionality of each function is:

- Applying PCA (for informative purposes only)
- Applying Fisher Scoring
- Applying ReliefF
- Applying tuning of sPLS-DA
- Applying sPLS-DA
- Applying the entire workflow

Applying PCA

The signature of this function is:
`apply_pca = function(dataset, classes, components = 10, debug = TRUE)`
`dataset` contains the data that will be analyzed by PCA.
`classes` contains the vector of class to sample of the dataset.
`components` is the maximum amount of components that PCA will try to obtain from `dataset`.
`debug`, if `TRUE`, will output in a different window a plot showing the calculated principal components, and the samples of the two first components plotted in a two dimensional space.

This step makes use of the `pca` function, from the package `mixOmics`, which takes `dataset` and `components` as parameters, and calculates the set amount of components. This is purely done for informational purposes, to show if there is by any change a configuration of the variables that maximizes their variance. This calculation of the PCA will be executed three times in the workflow:

- When the dataset is uploaded.
- When Fisher scoring is applied.
- When ReliefF is applied.

This way we will be able to see the effect of each step of the workflow, and quickly observe if there is actually a source of variance in the data.

As said, when `debug == TRUE`, this function shows two plots: the first one is a plot where we can see the principal components and their explained variance, and the second one shows the samples as clustered points placed according to their projection in the smaller subspace spanned by the components.

This function returns the result of the `pca` function, which contains all the information about the components.

Applying Fisher Scoring

The signature of this function is:

```
apply_fisher = function(dataset, positive, negative,
features_to_keep, debug = TRUE)
```

`dataset` contains the full dataset that will be analyzed by Fisher Scoring.

`positive` contains a list of the indices of the samples belonging to the positive class of the dataset

`negative` contains a list of the indices of the samples belonging to the negative class of the dataset.

`features_to_keep` is an integer specifying the amount of best features that will be selected from the original dataset after analyzing it with Fisher Scoring.

`debug`, if `TRUE`, will output in different windows plots showing information about the process that is being executed.

The Fisher Scoring algorithm was written manually, and it makes use of `positive` and `negative`. We apply it to every variable in `dataset`, then sort the variables in descending order according to their score, and return these scores and the best `features_to_keep` variables in `dataset`.

With `debug == TRUE`, it also outputs in a new window a plot with the scores from highest to lowest.

Applying ReliefF

The signature of this function is:

```
apply_relieff = function(dataset, classes, nearest_neighbors,
features_to_keep = 500, iterations = 0, estimator = 'ReliefFexpRank',
debug = TRUE)
```

`dataset` contains the data the user wants to analyze with ReliefF.

`classes` contains a list with the class of each sample with each index matching the index of that sample in `dataset`.

`nearest_neighbors` is an integer that specifies the amount of nearest neighbors of each variable ReliefF will calculate.

`features_to_keep` is an integer that specifies the amount of best features that will be selected from `dataset` after analyzing it with ReliefF.

`iterations` integer that specifies the amount of iterations that the ReliefF algorithm will perform. If it contains a positive number, that number will be the amount of iterations that will be performed. On the other hand, 0, -1 and -2 can be chosen, those meaning the number of samples of `dataset`, the natural logarithm of the number of samples, and the squared root of the number of samples, respectively.

`estimator` is a string that specifies the name of the evaluation method used. There are many that could be used, all of them specified in the manual of CORElearn[35], but here we will be using only two: ReliefFequalK and ReliefFexpRank.
`debug`, if `TRUE`, will output in different windows plots showing information about the process that is being executed.

This step makes use of the `CORElearn` library. It provides us with the function `attrEval`, which works as an interface to multiple methods of variables evaluation. It has multiple ways to work, but I will not explain all of them, instead I will explain the way it is implemented in this project. In this case, we give it the `classes` list, `dataset` having been previously casted to a data frame, the `estimator` name, the amount of `iterations` to perform, and the amount of `nearest_neighbors`. This returns us a list of the scores of each variable in the dataset, which we will use to, again, keep only the `features_to_keep` best ones. Then, just like we did with the Fisher Scoring step, we return the scores of the variables and the new dataset with the selected variables.

As said, we use two different estimators:

- With **ReliefFequalK**, the weight of the k nearest instances are the same.
- With **ReliefFexpRank**, the weight of the k nearest instances decreases exponentially with increasing rank, which is good in case we want to take into account conditional dependences between attributes.

With `debug == TRUE`, this step will also show a plot with the scores of each variable in descending order, but in this case it also shows a heatmap of the expression of the variables selected by ReliefF.

Applying tuning of sPLS-DA

The signature of this function is as follows:

`apply_plsda_perf = function(dataset, classes, components = 10, cv_folds = 5, cv_repeats = 10, debug = TRUE)`

`dataset` contains the data to be used for tuning sPLS-DA.

`classes` contains a list with the class of each sample with each index matching the index of that sample in `dataset`.

`components` is an integer that specifies the maximum amount of principal components that this function will try to compute.

`cv_folds` is an integer specifying the amount of cross validation folds that will be performed.

`cv_repeats` is an integer that specifies the amount of cross validation repetitions that will be performed.

`debug`, if `TRUE`, will show several plots with information of the different steps that this function has been performing.

This step makes use of the `mixOmics` library too. It provides a lot of functionalities to analyze omics data, and the functions `plsda`, `perf` and `tune.splsda` will be used here.

`plsda`, as its name indicates, executes PLS-DA to analyze a certain set of data. It uses `dataset` as the data to analyze, `classes` to indicate the class of each sample, and `components` to know how many components to compute. This function is used in conjunction with `perf`, because this one is used to analyze the analysis performed by `plsda`. It uses the data returned by that function, plus `cv_folds` and `cv_repeats`. This function is just to test, by classifying the data with the results returned by `plsda`, how good the analysis was.

`tune.splsda` is a method that returns a configuration for the most op-

timal, fast and accurate analysis with sPLS-DA. It uses `dataset`, `classes`, `components`, `cv_folds` and `cv_repeats`. It also uses an internally fixed variable called `list_keepX`, which contain a list of numbers from 5 to 100 in steps of 5. This function is executed as follows:

- Compute a principal component c.
- For each amount of variables $vars$ in `list_keepX`, perform sPLS-DA with c and keeping $vars$.
- Test the performance using `cv_folds` and `cv_repeats`, and from that obtain the Balanced Error Rate (BER).
- Save c and the $vars$ that obtained the best BER.
- Repeat the above steps for `components` times, but using also all the previously saved c and $vars$.

This tuning function returns the most optimal amount of components and variables to consider in each component to correctly analyze the data, along with data that proves that this is actually the best configuration to analyze our data with sPLS-DA. It does this by doing classification steps while selecting the data, being able to observe which amount of variables and components classifies the inputted dataset with the lowest error and in the shortest time.

This step returns the data of the test performed before the tuning, the data of the tuning, the amount of components to use, and the amount of important features in each component.

If `debug == TRUE`, this step will show in separate windows a plot showing the BER of the PLS-DA, and a plot showing the BER for every amount of components and every amount of variables used in each component, highlighting the best performing choices.

Applying sPLS-DA

This function has the following signature:

```
apply_splsda = function(dataset, classes, variables_to_keep, components
= 10, cv_folds = 5, cv_repeats = 10, debug = TRUE)
```

`dataset` contains the data the user wants to analyze.

`classes` contains a list with the class each sample belongs to.

`variables_to_keep` contains a list with the amount of variables to use in each selected component.

`components` is an integer specifying the amount of components chosen to be analyzed with sPLS-DA.

`cv_folds` is an integer specifying the amount of cross validation folds to be executed for validating the analysis.

`cv_repeats` is an integer specifying the amount of cross validation repetitions to be executed for validating the analysis.

`debug`, if `TRUE`, will show plots and more information in different windows to the user.

In this last step of the workflow, we finally perform the last analysis, sPLS-DA. It uses the function `perf` and `splsda` (also from the package `mixOmics`) which, given `dataset`, `classes`, `components` and `variables_to_keep`, will do partially the same as PLS-DA, but it will also continually select variables from the components until, in the end, it has selected the amount of variables specified in `variables_to_keep` for each component respectively. Then, `perf`

is used again in order to validate the analysis just performed.

This function returns the results obtained by the execution of sPLS-DA and the results of the validation.

If `debug == TRUE`, this function will show several informational plots in different windows. First of all, it will show the samples as clustered points placed according to their projection in the smaller subspace spanned by the two first components. Then, it shows the Area Under the Receiver Operating Characteristics (AUROC) plot of the first component sPLS-DA returned. AUROC is a method to test how much the model is capable of distinguishing between classes. The result goes from 0 to 1, and the higher the result, the better the model is at classifying the data between two classes with only the components and variables chosen to perform sPLS-DA. Then, the last plot shown is the BER of the model, in this case after having used sPLS-DA.

Applying the entire workflow

The signature of this function is as follows:

```
apply_workflow = function(dataset, classes, nearest_neighbors,
fisher_variables = 5000, relieff_variables = 500, pca_components
= 10, cv_folds = 5, cv_repeats = 10, debug=TRUE)
```

`dataset` contains the full dataset to analyze.

`classes` contains a list with the class of each sample of the dataset.

`nearest_neighbors` is an integer that specifies the amount of nearest neighbors of each variable ReliefF will calculate.

`fisher_variables` is an integer that specifies the amount of best variables the workflow will keep after executing Fisher Scoring.

`relieff_variables` is an integer that specifies the amount of best variables the workflow will keep after executing ReliefF.

`pca_components` is an integer that specifies the amount of components that will be computed in every step that requires to compute them. That is, in PCA, PLS-DA, sPLS-DA and sPLS-DA tuning.
`cv_folds` is an integer that specifies the amount of cross validation folds that will be executed in every step that performs cross validation, which are PLS-DA, sPLS-DA and sPLS-DA tuning.
`cv_repeats` is an integer that specifies the amount of cross validation repetitions that will be executed in every step that performs cross validation. Those are PLS-DA, sPLS-DA and sPLS-DA tuning.
`debug` is a boolean that specifies if the workflow will be executed with debugging data or not. It is used in all the functions and its effects have already been explained in each one of them.

This step basically executes all the steps without stops, one after another. It first casts the `dataset` into a matrix. Then extracts from `classes` the sample indexes belonging to each class, and starts executing the functions.

This function returns the results of the sPLS-DA analysis, returned by `apply_splsda`. This is a function that is supposed to be used from R's interactive console, in order to find bugs or crashes, and for this reason it will not be used by a regular user.

2.3 User interface

The workflow previously described could simply be executed in a terminal emulator or a command line. However, one of the problems we wanted to solve is not having a friendly user interface to use the tool that has been made. There are many tools and frameworks with machine learning functionalities, but most of them provide a set of functions or methods to add to your code. In our case, we provide all of that, along with a user friendly interface to interact with the data that is being analyzed in real time.

2.3.1 Design

First of all we need the answer to two separate questions: what would the user want to customize, and what would the user want to see?

It is important, for a tool that aims to provide the most complete functionality, to allow the user to easily customize every parameter that is going to have an impact on the results they will get. In this project we allow the user to customize everything except for one thing: the list of amounts of variables that are candidates for being the most optimal for a certain component when tuning sPLS-DA (that is, a list named `list_keepX`, mentioned in the above section about `apply_plsda_perf`).

On the other hand, the user would also want to see as much information as possible about everything their data is going through. For each part of the workflow there are different kinds of information that it would be good to retrieve:

- **When applying Fisher Scoring**: The only important data to show here is the ordered scores of the variables, for the user to know how many of them approximately would be the best to select for the next steps.

- **When applying ReliefF**: Again, we want to know the scores of the variables in order to know how many of them would not be necessary for the last step of the analysis. Also, it might be interesting to visually see a plot with the expression of the selected variables, since after ReliefF there should be few enough remaining to start seeing some patterns, if there were any.

- **When applying PLS-DA tuning**: In this case we are basically interested in knowing how many components and variables per component have been selected.

- **When applying sPLS-DA**: This is the step in which we want to have the biggest amount of information. We want to know if there is a real differentiation between the two classes of samples that are being analyzed, we want proof of it, and we want to know which variables cause that.

- **When applying every step**: For the purpose of just having more information, we would want to know the principal components of all the variables remaining in each step of the workflow, in order to start seeing from the very beginning if there is some relationship between the features of a sample and its class.

2.3.2 Implementation

For this part, I chose to use the library `shiny`. Shiny provides an entire framework to build a user interface in the form of a web service. It makes the developer able to easily build it with functions that provide different layouts, inputs for the back end, informational outputs and even interactive plots. It also provides everything needed to automatically host the web page and allow connections from a local network. Shiny is the most popular and complete framework and, for all these reasons, it was a safe bet to choose it to make the interface.

This part was also, like the workflow, implemented by parts.

The explanation will be divided in two sections: the server (back end), and the ui (the front end). This is because the implementation of the code also requires to be divided between these two parts.

UI (front end)

Shiny calls the front end the UI part of the code, because everything coded in this part will be only related to the HTML code that, when the app is executed, will be generated and served to the user. Shiny also allows the edition of JavaScript and CSS code, but it is only used to show an animation when the workflow is executing any step.

In the UI you can define a layout for the web page. In this case, we want a part to input the data, and another part to see the results. Since the inputs are going to be mostly numbers and, in general, data that are represented in a small amount of space, while the results will be mostly bigger graphs and plots, we chose one called "sidebar layout". It shows a thinner section at a side of the page, in contrast to the other part of it (Figure 2.2).

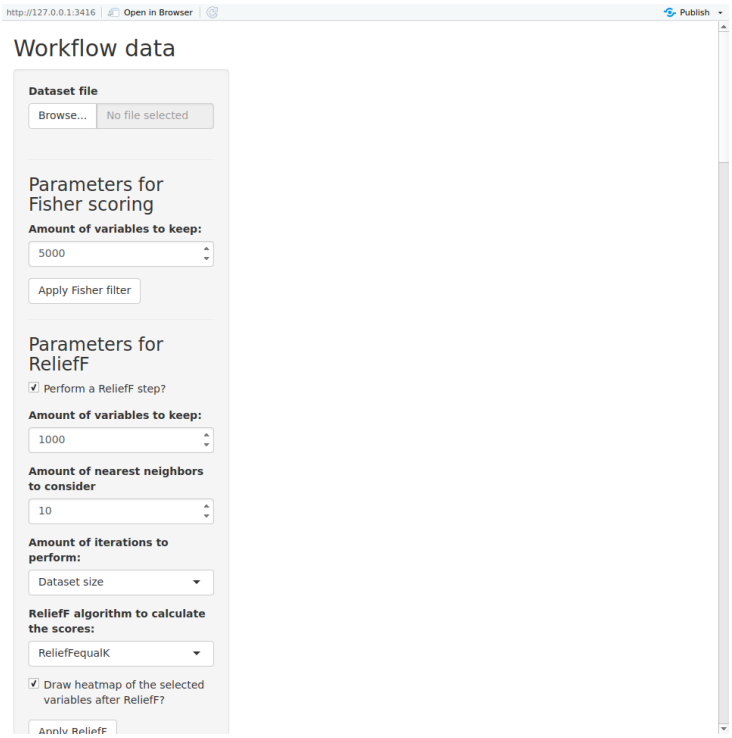

Figure 2.2: Layout of the UI

In the sidebar we can divide the different parts of the workflow. First of all we have an input for providing the data to the server. Then, each part of the workflow has some parameters to customize, the choice to not see some plots (except in Fisher Scoring), and a button to apply that part of the workflow and start the next one. Finally, there is a button to export the results in a pdf file and save it.

In the main panel, the different plots and messages thrown by the server will be shown in order. Starting from the top, the user will see the error messages thrown when a part of the workflow has not been well executed. Then, there will be the PCA plot, showing the principal components after applying every single step. After that, there are the plots of Fisher Scoring, then the plots of ReliefF, the plots of the PLS-DA tuning and the plots of

the results given by sPLS-DA. Due to particularities of Shiny, the plots that the user chooses not to see will still take some blank space in the main panel, because Shiny reserves that static space for the plot since the startup of the app.

Server (back end)

The first thing needed to be done by the back end is augmenting the maximum size of the data uploaded. The size that Shiny allows in its default configuration is too small to even upload the test datasets I used, so it had to be manually changed (I set it to 100MB, but it can be easily changed to any size0).

Then, before starting with the more in-depth explanations, there are two concepts that are important to talk about: observed events and reactive variables.

- First there are the **observed events**. An event is an interaction the user does with any element of the web page. In Shiny, using the function `observeEvent`, you can encapsulate a part of code which will only be executed when the user triggers an event targeting the element specified in the first parameter of that function. This encapsulated code is created in a context different from the rest of the code, and thus variables used in there that had been given a different value in other previous parts of the code will behave in unexpected ways. To solve this problem, Shiny provides reactive variables.

- **Reactive variables** are functions that store the data they receive as parameters, and return the data they contain when executed. They are special and essential because, contrary to regular variables, these actually change in all parts of the code, regardless of the encapsulation they are in, and for this reason they can be used to pass data from the code of one event to another without any loss of information.

At this point, all the necessary reactive variables are declared and initialized as `NULL`. After this, all the events need to be declared.

The first event we declare is triggered when the user uploads a file. It first checks that the uploaded file is a csv or RData file. If it is not, it outputs an error message in the UI and stops the execution of the rest of the code. If it is, it checks that the file contains the data necessary to be analyzed: in the case of a csv file, the whole dataset just needs a column called "class" with the class of each sample, and in the case of an RData file, it needs a data frame or matrix-like structure with the data and a vector-like structure, also called "class", which again contain the class of each sample. In both cases, there must be no missing data. After this checks have passed, the samples are divided between classes, and a plot showing the 10 principal components (Figure 2.3) and a plot showing the ordered Fisher scores (Figure 2.4) are shown, and the default values of the inputs indicating the amount of variables to keep after Fisher and ReliefF are automatically set to a 10% and a 1% of the total, respectively.

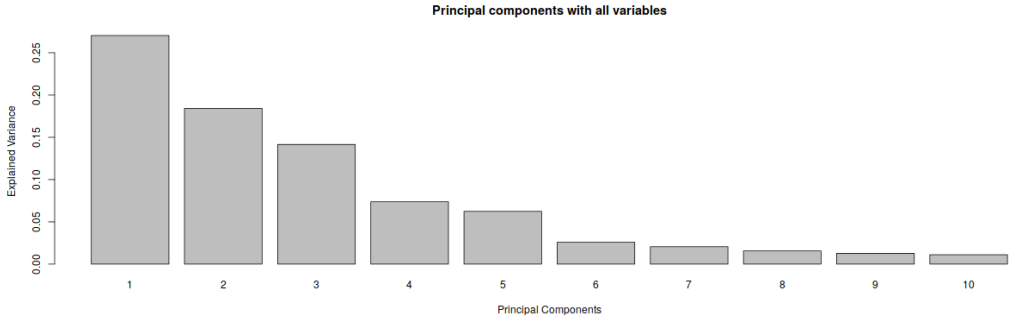

Figure 2.3: PCA plot

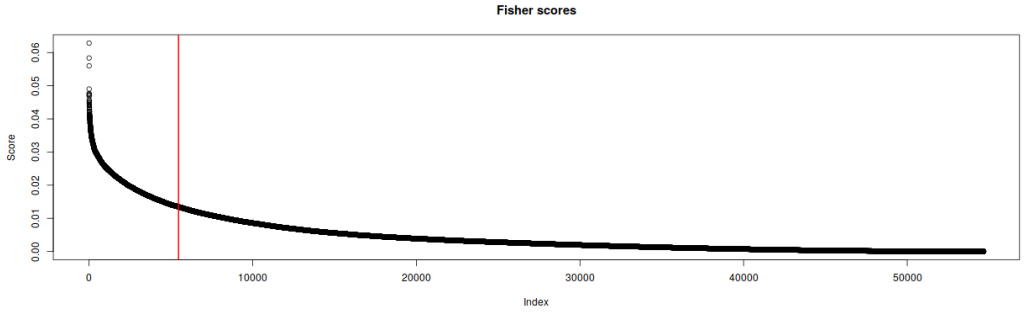

Figure 2.4: Fisher scores (the red line divides the kept and discarded features)

Then, pressing the button to apply Fisher filter, the specified amount of variables will be saved and passed to the next step of the workflow, and a new PCA plot will be shown, in this case using only the reduced dataset. In case the checkbox to apply ReliefF has been deactivated, nothing will happen with that data, the parameters set will be ignored, and pressing the button to apply it will leave it as it came from the Fisher filter. Otherwise, the data will be analyzed with ReliefF, using the parameters the user has set. Then, a plot showing the score of each variable (Figure 2.5) will be generated, and a plot showing a heatmap of the remaining variables after discarding the rest (Figure 2.6) will be shown in case the user marked the checkbox indicating so.

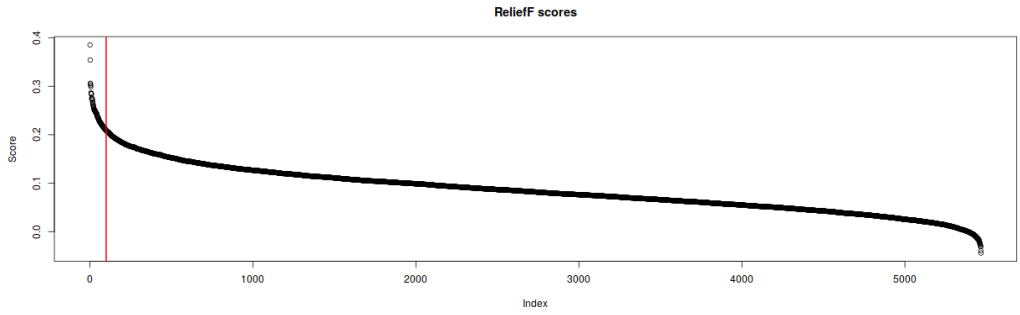

Figure 2.5: ReliefF scores

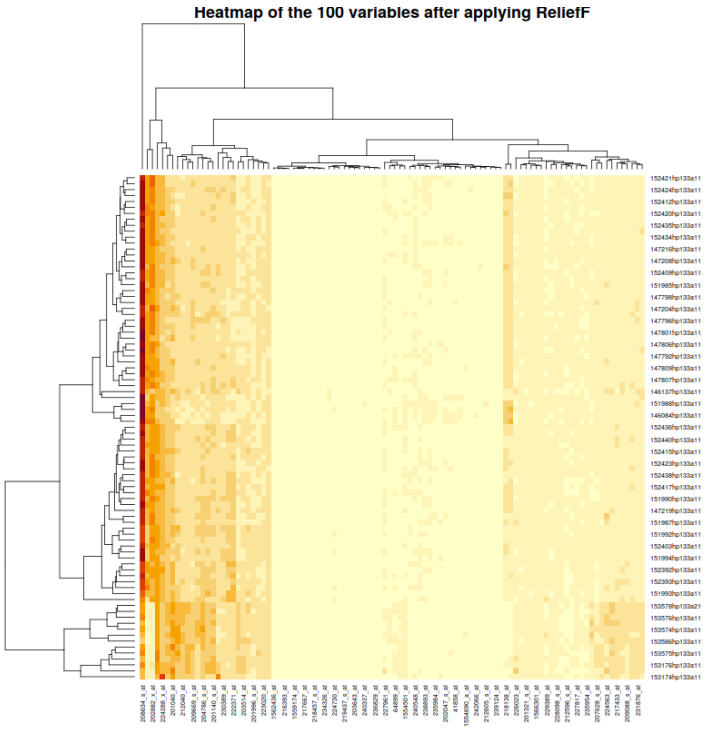

Figure 2.6: Heatmap of the remaining variables

After this, pressing the button to apply ReliefF will save the indicated amount of best variables (or just use the ones obtained by fisher, if ReliefF was deactivated), the PCA plot will be updated with only said variables, and the PLS-DA tuning will take place. It will be performed with the input amount of components, cross validation folds and cross validation repetitions. After this process, two plots will be shown: one showing the BER per principal component (Figure 2.7), and one indicating the error rate per every amount of variables possible per principal component (Figure 2.8). This is the slowest step and might take some time to obtain the results.

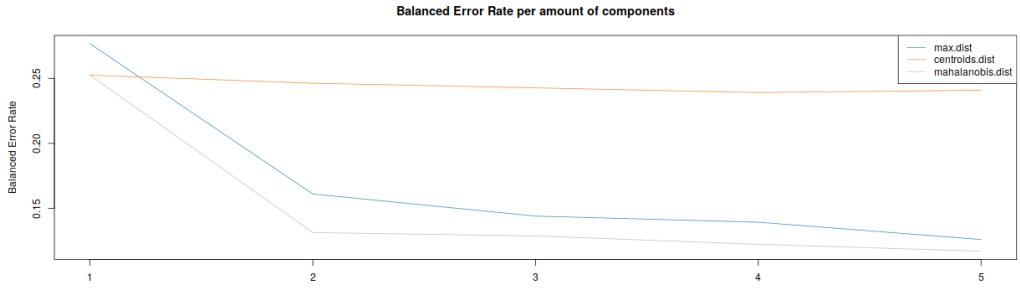

Figure 2.7: BER per component

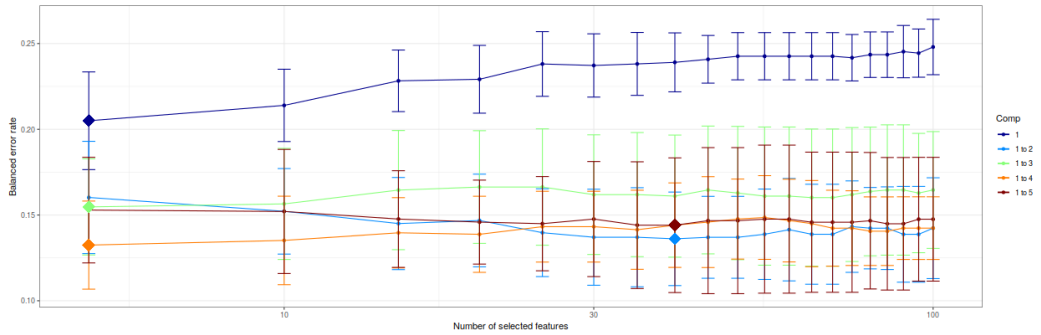

Figure 2.8: Error rate per amount of variables selected per component

Then, pressing the button to apply sPLS-DA will run it with the settings already tuned, and the last exhaustive analysis will be performed. When the analysis has been completed, four new plots, all of them being optional, will appear. The first one depicts a clustered bidimensional representation of the samples projected to the subspace of the two principal components that explain most of the variance between them (Figure 2.9); the second one shows the AUROC of the first component (Figure 2.10); the third one shows a Clustered Image Map (CIM) of all the selected variables (Figure 2.11), which is basically a heatmap showing more information about the class each sample belongs to and clustering the samples and variables according to their correlation; and the last one shows the loadings of the first component (Figure

2.12), also showing a number indicating their relevance in said component. Additionally, a table will show up with the variables from the first component that have been selected.

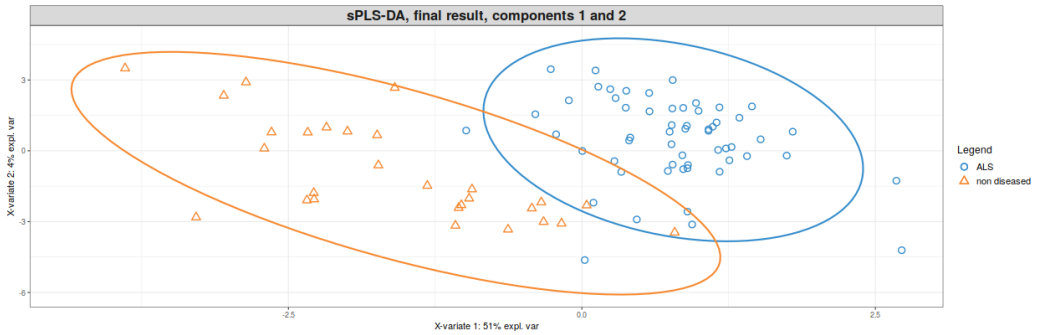

Figure 2.9: Projection of the samples to the subspace of the two best principal components

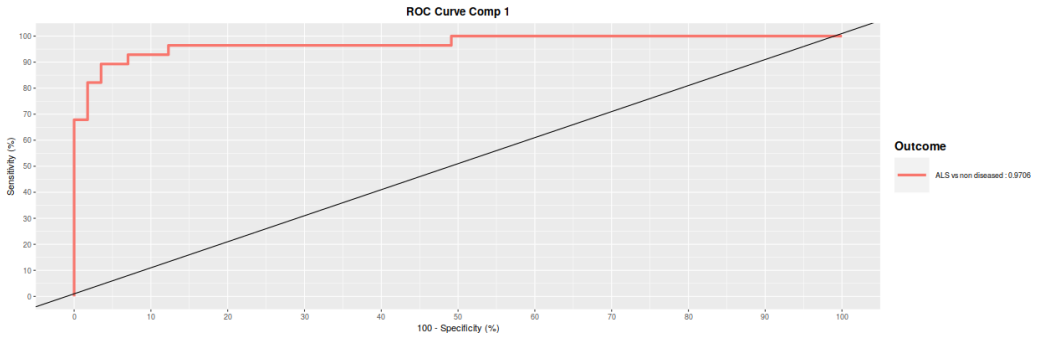

Figure 2.10: AUROC plot of the first component

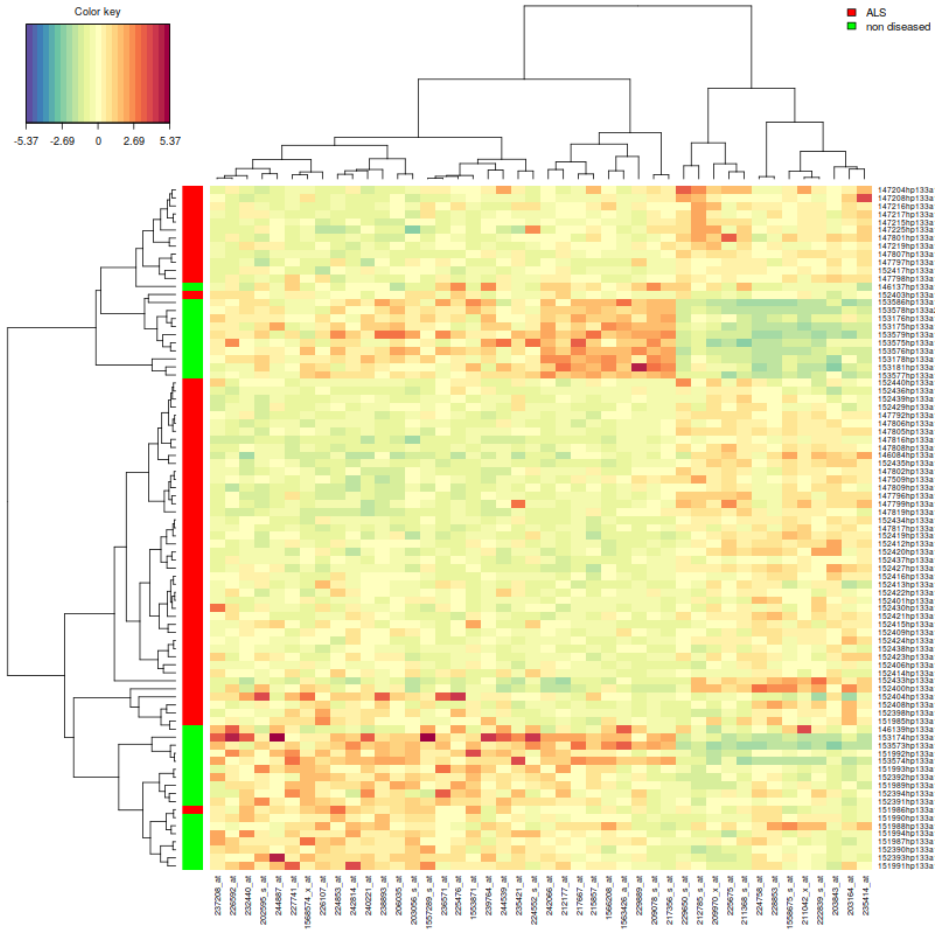

Figure 2.11: CIM of all the variables and samples

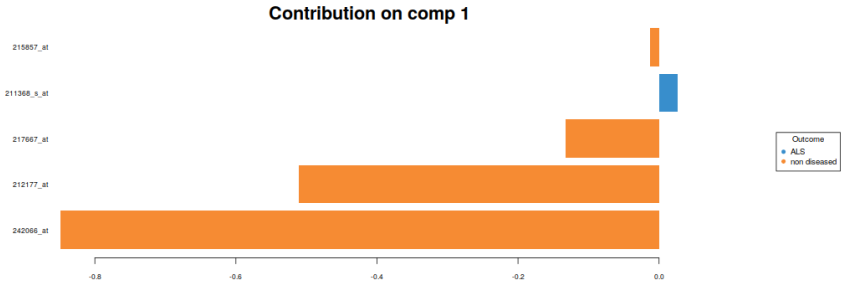

Figure 2.12: Loadings and relevance of the selected variables in the first component

Finally, if the user presses the button to download the pdf report, he will receive in a matter of seconds a full pdf file with all the information about the data they uploaded, the changes it had gone through in the different parts of the workflow, all the plots they decided to get, and finally a list with all the selected variables. This not only allows the user to use the results of the analysis in an offline environment, but it also gives them more information about the analysis that has just been performed. Shiny does not allow multiple plots to be rendered in the place of one, and for this reason some of them do not appear in the webpage (for example, AUROC or the loadings of all the selected components), but R can draw as many plots as desired in the pdf, so the user will see all the information that was hidden before.

2.4 Experimentation

2.4.1 An experiment with real data (Golub et al. 1999)

To test and experiment with our workflow, we will pretend to be researchers who want to know if the genes of a person are relevant to the kind of leukemia that person is a patient of. We will use the data of *Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring*[1], and we will validate our results against the ones that study obtained.

To start, we downloaded and formatted as csv an improved version of the dataset they used as necessary to be inputted to our software (that means each row is a sample, each column is a feature, and there must be a column called "class" which distinguishes the class each sample belongs to). This dataset consists in the original 38 bone marrow samples (27 ALL, 11 AML) of the original dataset plus 34 more samples (20 ALL, 14 AML), for a total of

72 samples (47 ALL, 25 AML). Also, the amount of features is also improved: the initial dataset contained 6817 genes, but the one we use contains 7129.

First of all, we observe the following plots, given by PCA and Fisher Scoring (Figure 2.13).

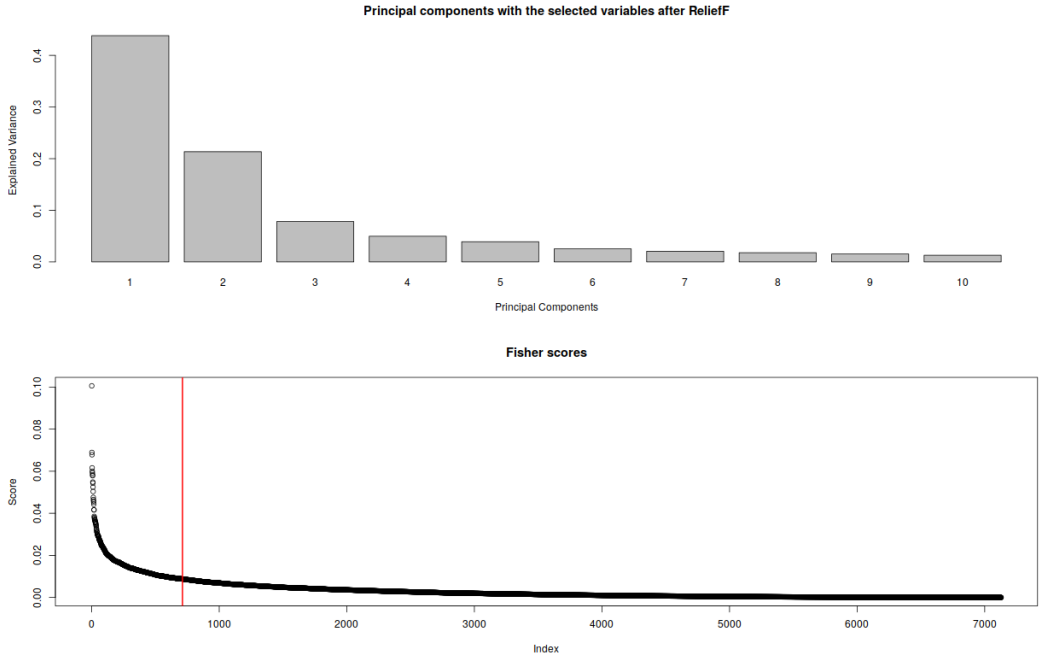

Figure 2.13: PCA and Fisher Scores of Golub et al.

From what we can see, PCA is detecting that some principal components explain much of the variance of the data we have, which is a good indicator of that the genes might give might have a relevance to which type of cancer the person they belong to has. As we can see in the Fisher plot, there are also some genes with a high score, but it rapidly decreases and stabilizes. As said in previous sections, the software automatically sets the amount of variables to keep as a 10% of the total, 712 in this case, so we will leave it like that.

Pressing the button to apply Fisher will get us another PCA plot, and one with the ReliefF scores (Figure 2.14).

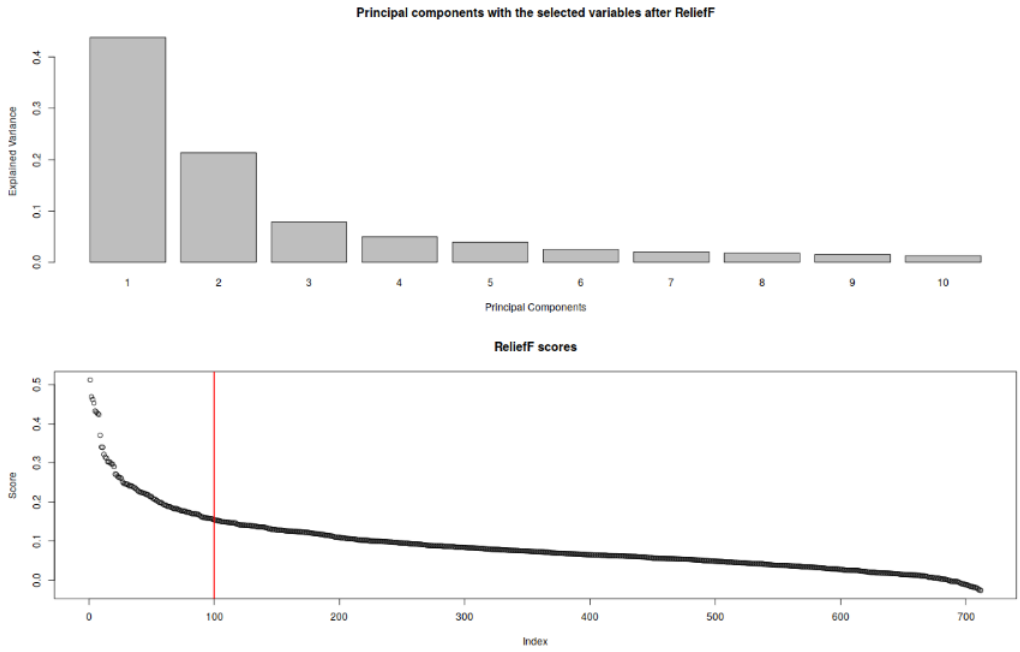

Figure 2.14: PCA and ReliefF Scores of Golub et al. (remaining 712 genes)

The ReliefF analysis was performed with these data: 50 nearest neighbors, the amount of iterations performed was equal to the amount of variables, and the estimator was ReliefFexpRank.

As we can see, the principal components have not changed much, which is good because it means that the almost 6500 variables Fisher Scoring left out were mostly not relevant to wether a sample is of one class of the other. Observing the ReliefF scores, we can see again that only a few have higher scores than the rest. Since the 10% of 712 is smaller than 100 (the minimum amount of variables possible to be kept after ReliefF), we can keep all those 100 variables and, according to the plot, we would not be leaving out any of the most relevant ones.

Also, for informational purposes, we can take a look at the heatmap we chose to be shown (Figure 2.15).

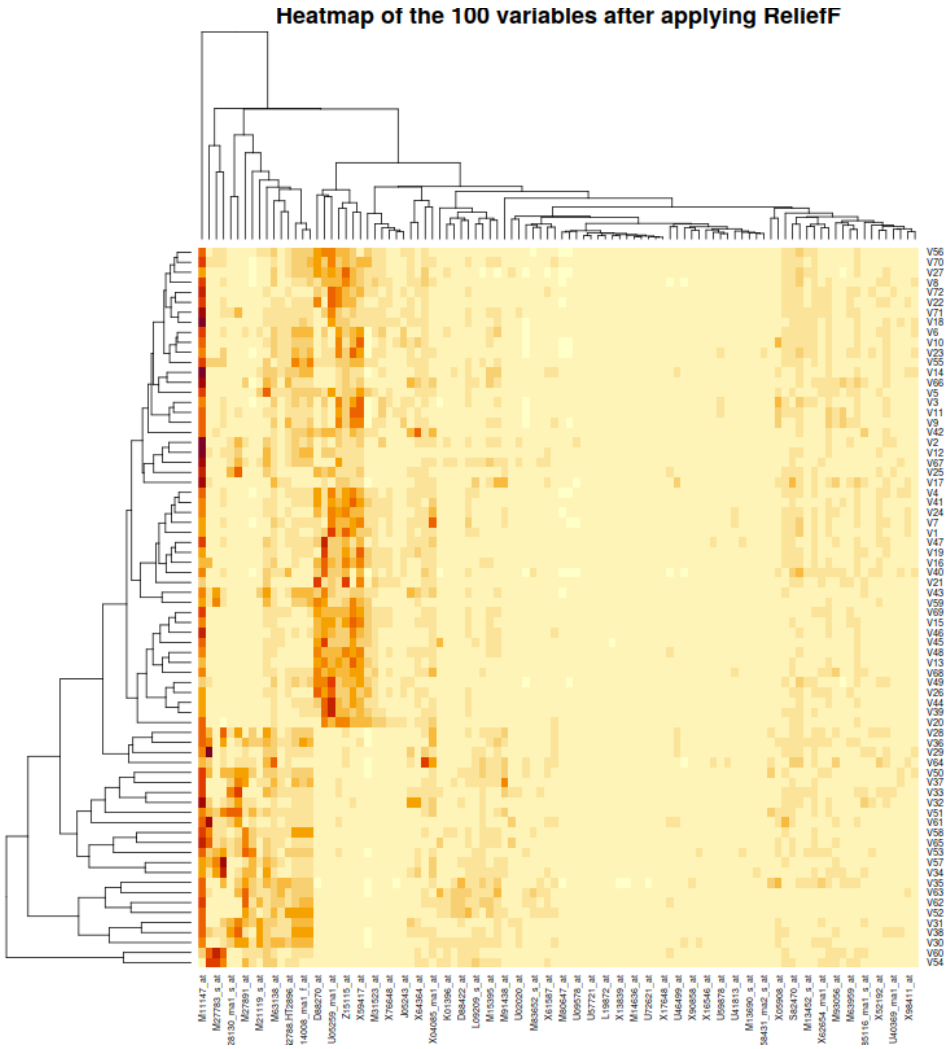

Figure 2.15: Heatmap of the 100 remaining genes after ReliefF

Although we do not see information about the classes of the samples at this point, we see how a pattern is starting to be generated, clearly separating the genes at the most left part of the heatmap, which have a higher expression in the samples clustered at the bottom part of it, with the rest of them, giving us the impression that genes may have a clear relevance to determining which

kind of cancer, between the two that are being studied, a patient will suffer of.

Clicking the button to apply ReliefF, the PLS-DA tuning will start. The tuning is executed with the default parameters: 5 principal components, 5 cross validation folds and 10 cross validation repetitions. This step shows us these plots:

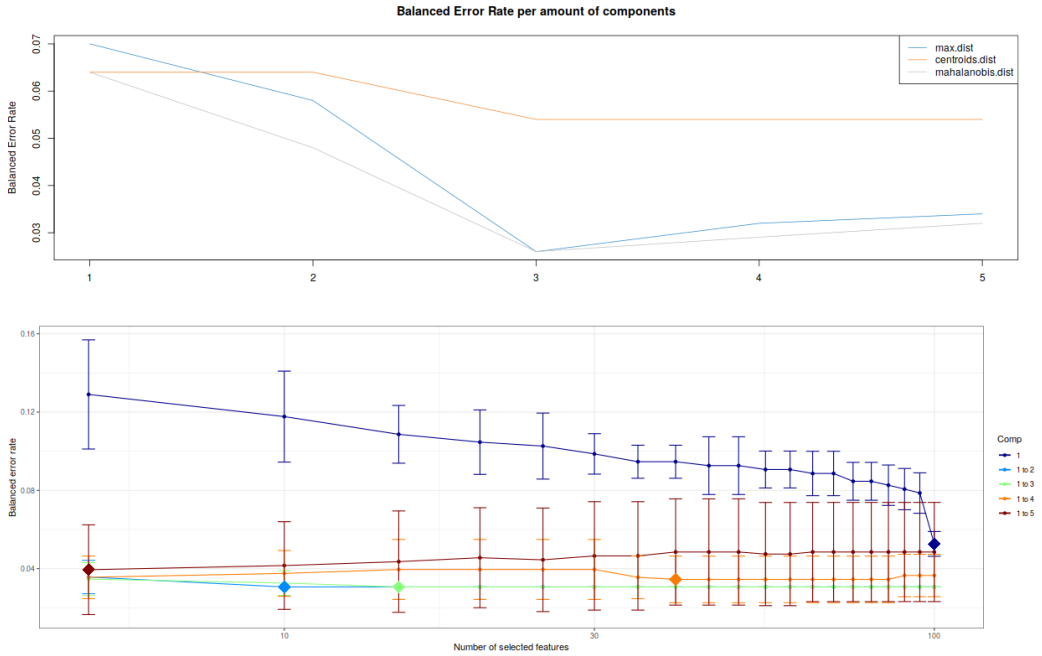

Figure 2.16: Error rates and selected variables per component for the remaining 100 genes

As we can see, the error rates with one component are really low and don't change much with the next components. The second plot shows the reason: the maximum amount of variables possible to select per component is 100, but we already only have 100 genes, so selecting all of them in the first variable is the most optimal option. We can see, by the results of the next components, that this is indeed true: selecting more or less variables does not

change the error rate, represented more or less by a flat line, because all the variables are the best choice to divide the two classes. However, the tuning selected two components, since, apparently, the second component with 10 variables improves a little the error of the classification.

If we press the button to apply this tuning and execute sPLS-DA, we will get the final results. Going by parts, we first see the projection of the samples in a two dimensional plot (Figure 2.17):

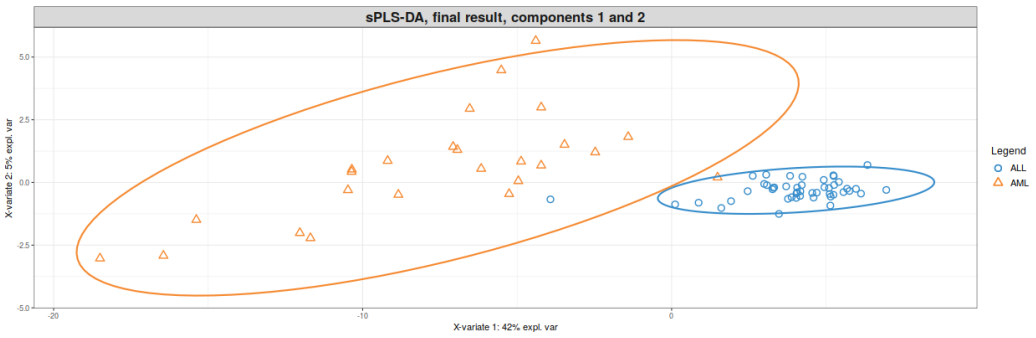

Figure 2.17: Projection of the samples in the subspace of the two components

Being the X axis the projection in the first component, and the Y axis the projection in the second, we clearly see that the samples are almost perfectly clustered, except for two cases, one of each class, that are close to the opposite cluster.

Next let's observe the AUROC of the first component:

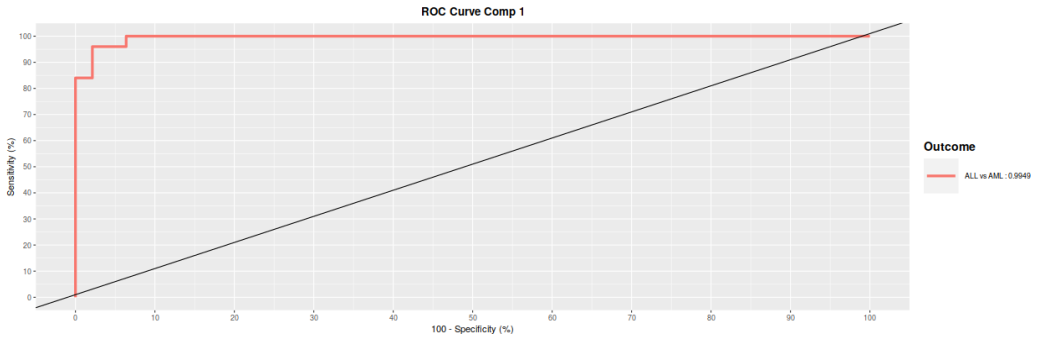

Figure 2.18: AUROC of the first component

We can see that the predictive model is very accurate. In fact, the outcome, indicated by the legend, is 0.9949 which confirms that, using our 72 samples, there is a clear and very well defined difference between our classes according to the gene expression of the samples.

Let's observe now the CIM:

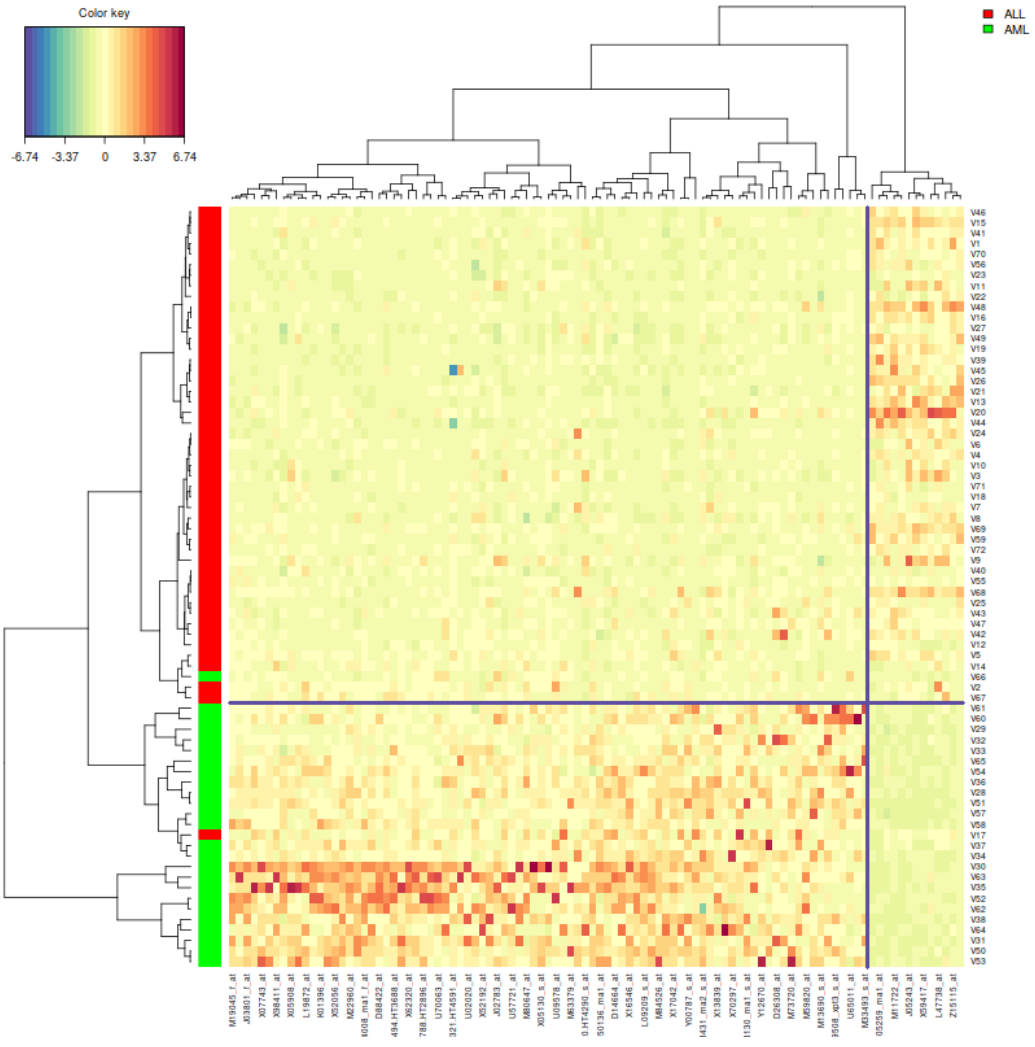

Figure 2.19: CIM of the final selected variables and samples

It is clear now that there is a difference between the gene expression of people of one class and the other. The purple lines (drawn manually, not automatically obtained) divide the map in clusters of closely related genes and samples. We can see that the samples, like in the above plot (Figure 2.17) are armost perfectly separated, except for two of them that ended up in the opposite cluster, as the color in the leftmost part of the plot indicates. We can also see that the genes are divided in two groups, being the ones at the left the genes for which a higher expression means pertainance to the AML class, and a lower expression means belonging to the opposite class, and the other way around.

Finally, let's take a look at the selected variables and their loadings:

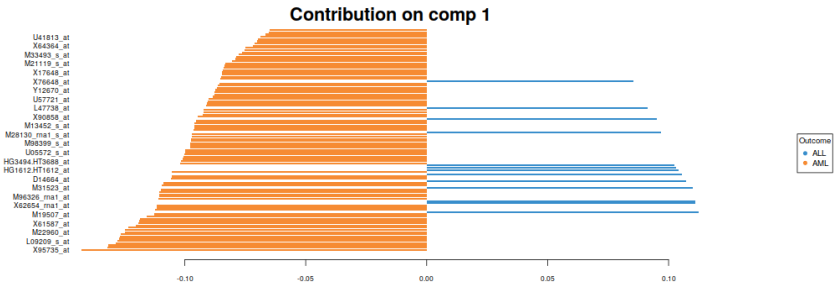

Figure 2.20: Loadings of the first component, the color indicating the class each gene contributes to

Selected variables

	value.var
X95735_at	-0.14
X17042_at	-0.13
M23197_at	-0.13
M84526_at	-0.13
L09209_s_at	-0.13
U46499_at	-0.13
M27891_at	-0.13
M16038_at	-0.13
M22960_at	-0.12
M63138_at	-0.12

Figure 2.21: Loadings of the 10 most relevant genes

These 10 genes are the first ones of the 100 that have been selected as the most relevant ones out of the entire set of variables. We will use them to draw some conclusions. We can consider that these are the ones that are the most relevant to separate the samples in two classes. Also, according to the loadings plot (Figure 2.20), these are useful to define the samples of the AML class.

Now, if we take a look at the results of Golub et al.'s study:

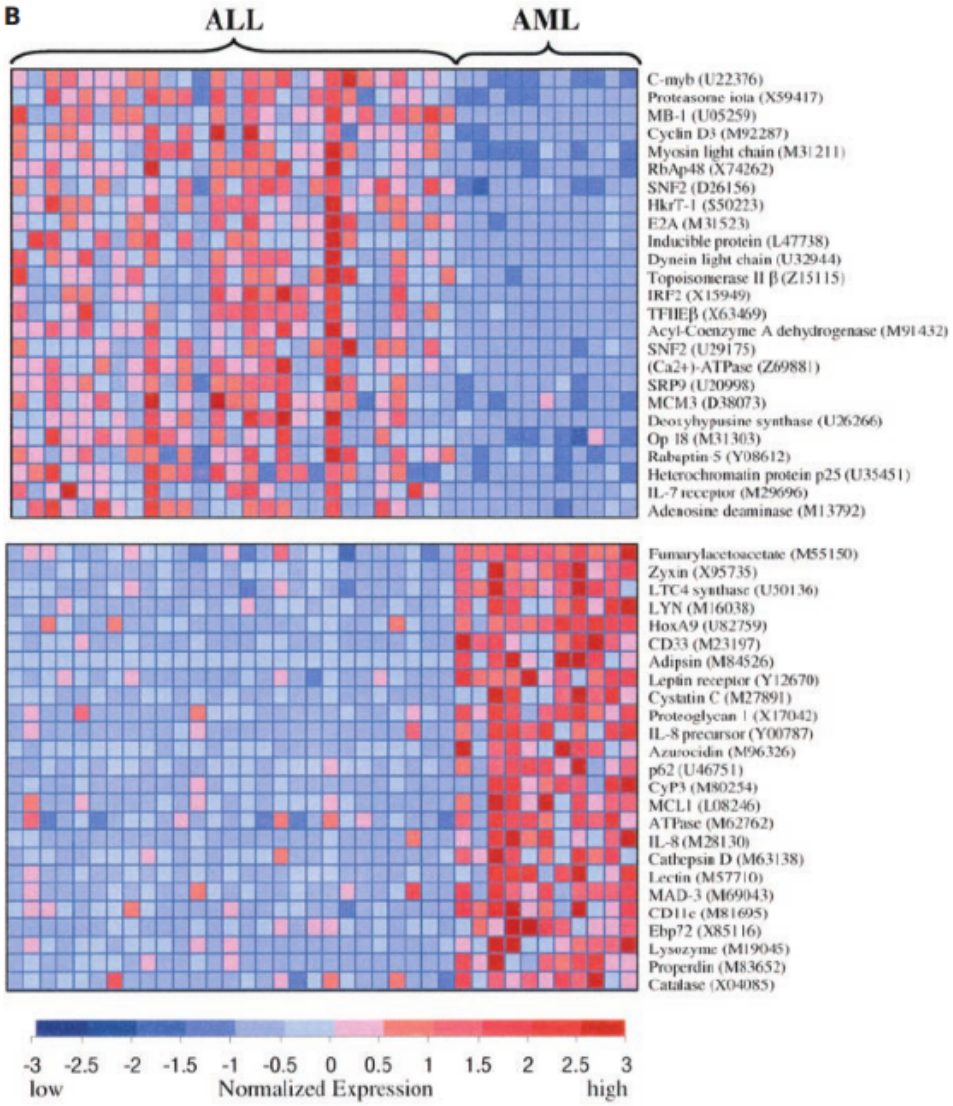

Figure 2.22: Gene expression for the 25 more relevant genes of each class[1]

As we can see, from the selected 10 ones some appear in the plot of the original study (Figure 2.21), such as Zyxin (X95735), CD33 (M23197) and Adipsin (M84526). A total of 6 of them appear, all of them in the bottom half of the figure, which means that, indeed, they are relevant when determining that a sample belongs to the AML class. However, there are 4 that still do not seem to match any of the 25 most relevant genes for Golub et al. There are some reasons for this to happen, but the three main ones could be: we have more genes, and those 4 could belong to the extra ones the original analysis could not analyze; we have twice as many samples, so the analysis now performed could have considered that those genes that for Golub et al. were not important, are indeed relevant to the problem with the new data we possess; and finally, the algorithms and techniques have evolved since 1999, and it is very possible that these new techniques that our workflow makes use of could make new discoveries on this field. If we take a look at more recent studies[36][37], in which they used the same data as I did, we can see that indeed, among the top ranked genes there are X17042, U46499 and L09209, which were not in Golub's study but appear in our analysis, and the first two of them do not just appear as interesting genes, but they are also among the ones of the highest interest for the problem.

Taking all of this into consideration, we can say that this software has provided for a robust and accurate analysis, arriving to very correct results with a high rate of precision, and a good selection of genes that indeed matches the ones of previous studies, reinforcing the hypothesis of that the genes of a person will define the kind of leukemia they will suffer of. Also, the entire process, from loading the web page to dowloading the pdf report, including all human input and change of parameters, has been performed in roughly 1 minute and 10 seconds, becoming really fast and having saved

a lot of time that otherwise should have been spent in implementing the different steps, testing the workflow, adding the representation in plots and performing it with the data to analyze.

2.4.2 An example with random data

In this case, I will use a dataset without any relevant or important data. For this purpose I made a quick script in python, which will be uploaded to the repository along with the rest of the project files, that generates a csv file processable by the workflow. This file contains 100 samples (50 of the "AAA" class, and 50 of the "BBB" class) with 20000 genes each, for a total of 2 million genes. The gene expression is a random value from -100 to 100, with no relationship of dependencies with other genes or samples. The purpose is to show that, indeed, the software will return results proving that in this dataset there is not a clear indicator of the fact that the separation of the classes is caused by the genes.

Starting by the same way, the data will be uploaded and the first plots will be shown (Figure 2.23).

These plots should already be explanatory enough: the extremely low fisher scores and explained variance of the principal components is a great indicator of the fact that these genes and classes have no relationship among them whatsoever. So our hypothesis was true: the genes of this dataset do not have any relevance in the definition of the class of their sample.

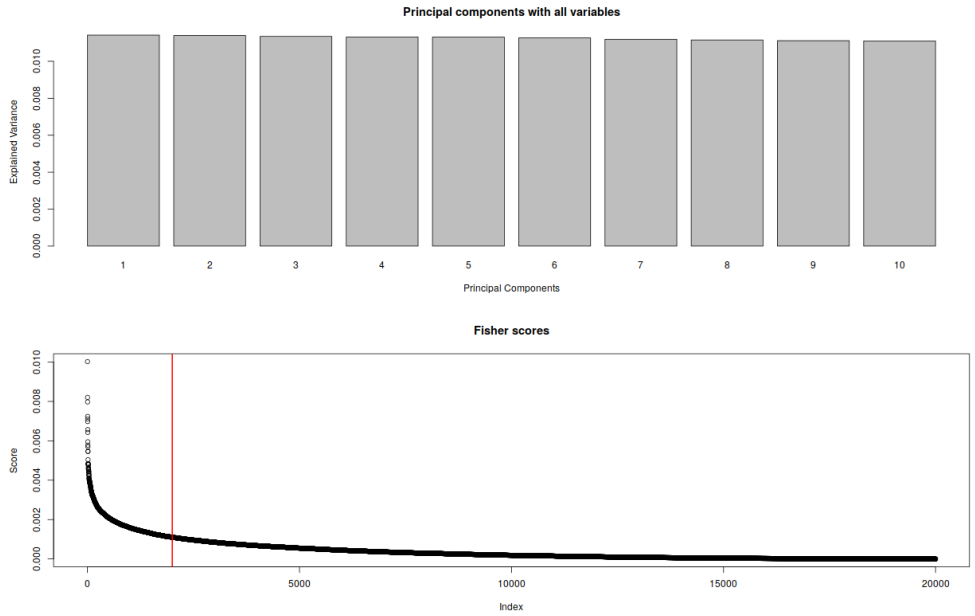

Figure 2.23: PCA and fisher scores of the random dataset

However, for curiosity, I will continue the workflow. After applying Fisher Scoring and ReliefF, keeping a 10% of the variables in each step, we are left with 200 variables. The PLS-DA tuning has considered that, with those 200 variables, the best choice is to use 195 for the lowest error rate (Figure 2.24), which is expected for random data because of not following any pattern.

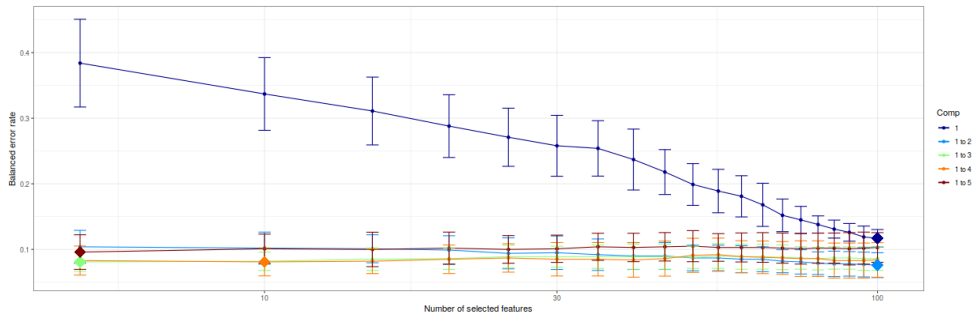

Figure 2.24: Error rates and selected variables per component for the remaining 200 genes

What is really interesting is the AUROC and CIM of these selected variables (Figures 2.25 and 2.26). Not only does it find certain relationships between the variables and classes in the dataset, but it is also able to perfectly classify the genes in their classes with a 100% accuracy, and if we look at the CIM, although at first sight it appears to be nonsensical, we can actually see that the genes of higher expression of the AAA class are clustered at the right side of the map and the others at the left, while the genes of the BBB class look the opposite. Of course, this does not mean that the selected genes are relevant to the problem, all the data was randomly generated after all, and as said after the first step, the low variance of scores and components meant that there can not be a clear indicator of separation between classes. However, this experiment can tell us about the power of this software, since it has been shown to be able to extract even the smallest amount of important information hidden in the middle of a big amount of completely irrelevant data, and all of that done in roughly 2 minutes and 30 seconds.

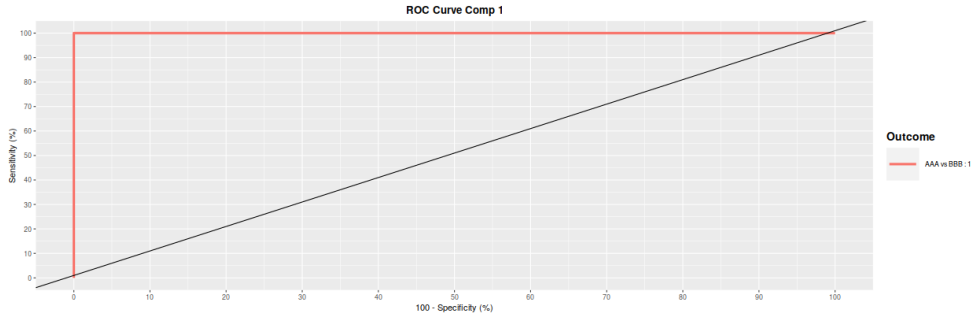

Figure 2.25: AUROC of the first component of the random dataset

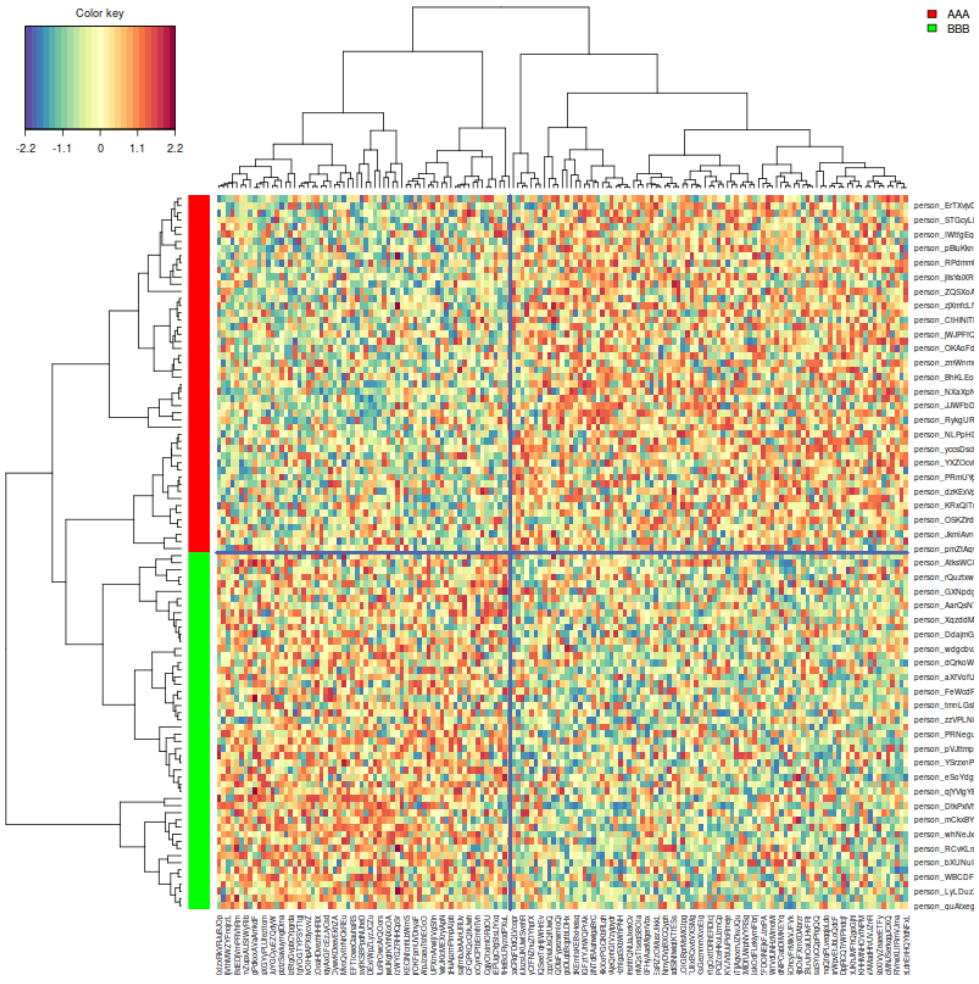

Figure 2.26: CIM of the final selected variables and samples from the random dataset

Chapter 3

Temporal planning

3.1 Planning and scheduling

This project will take approximately 5 months, from January 11th, 2019 to June 16th, 2019, a few days before starting the presentations.

It is worth mentioning that there is a margin of error in the planning of the whole project, and revision and changes might and probably will have to be made in order to correctly achieve the objectives. However, since the project is planned to be finished days before the presentation there will be time to correct and solve all the inconvenients that could be found.

3.2 Task descriptions

3.2.1 Acquire background in genomics and machine learning

The first task to be made is learn about the subjects I will be working on. From the first day I have been reading literature on the usage of machine

learning in the analysis of omics data. Starting with *Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring*[1], which is considered one of the most important researches on applying genomic tools, such as DNA microarrays, to cancer research, leading to important discoveries about leukemia. Because of this, it could be considered the base of everything I will be working on later so reading and understanding it was very important. Also, I read some more papers[**?**][**?**][**?**], all about the usage of machine learning techniques to analyze and profile the gene expression of different illnesses. Finally, I read about the state-of-the-art on microarray datasets and feature selection methods[**?**] in order to have an understanding of the different methods and techniques developed in the last years and their correctness and precision when being used with certain datasets.

3.2.2 Learn to use the tools

The language I will be using to develop this project will be R, and the IDE will be RStudio. It is not a new language, but a long time has passed since I used it for the last time. Also, I did not use it very thoroughly nor I used it for machine learning. Because of this, a certain period of time is needed in order to become familiar with it once again. To achieve this, I will be learning the basics and then developing a little project with help from a book called "Data Analysis and Graphics Using R"[**?**]. This task should not take longer than 2 or 3 weeks to complete. However, it is very important tto understand everything that has been done in order to easily apply it to the features of the actual project.

3.2.3 Develop the FS workflow

This is one of the two core tasks of this project: to develop the feature selection workflow. First of all it is important to know and understand the format of the data I will be working with. It will probably be not very different in comparison to the data I would have used in the previous task, but it is still worth taking a look at. Then, a study of different methods and algorithms will have to be made. There are many different ways to develop a workflow and not all of them work the same. Actually, the results can be very different just by using a different algorithm in a certain step of it, according to the data shown in [**?**]. It is also very important to test the correctness of the results but also the time it takes the workflow to complete the execution, since not only does it have to be accurate, but also relatively fast, so optimizing, if necessary, the critical parts of the code becomes a very important subtask to perform. This task should take around 4 weeks to be completed, spending most of the time on the implementation of the workflow and the rest on analyzing and testing it.

3.2.4 Develop the representation

This is the second core task of this project. In order to present the results and analyze them in a more visual way, it is necessary to graphically represent them. In this task I will first analyze the format of the results. This is crucial to develop the correct way to represent them in a humanly readable form. Of course, testing that the outputted graphs and results are correct is also required. I expect this task to take shorter to perform than the previous task since the data to use will be much smaller. However, after having it done it will need to be integrated with the FS workflow in order to have

an entire project capable of performing both tasks one after the other. So, the expected amount of time needed to do all of this is also no more than 4 weeks.

3.2.5 Final complete test

Once the code is done, right after finishing the integration of the graphic representation with the FS workflow, it will be necessary to test everything together, so as to find out if something is not going as expected when executing both parts of the software at the same time. Part of this task is also solving the problems found while testing and optimizing the last inefficient parts of the code, and the plan is not to change anything else after this has been done. This task should be pretty short and it is expected to take no longer than one week to be done.

3.2.6 Final steps

These final steps are basically collecting information. First of all, the documentation about the project needs to be done. This is something that will be done at the same time as the code is being written, in order not to lose any information about the process, but final reviews of it all and some updates will have to be made afterwards. Secondly, I will do the slides for the presentation and the preparation of it. This will be the last task to be done before the end of it. It should not take longer than two weeks, but I could take as long as I needed until the days to present it.

3.3 Estimated time

The times the tasks will take to be done are calculated according to my available working time, which is approximately 4 hours a day (actual times may vary), and shown in the following table.

Task	**Time (in hours)**
Acquire background in genomics and machine learning	140
Learn to use the tools	84
Develop the FS workflow	140
Develop the representation	132
Final complete test	28
Final steps	56
Total	**580**

Table 3.1: Estimated development time of the different parts of the project

3.4 Gantt chart

The following figure shows the Gantt chart of the project.

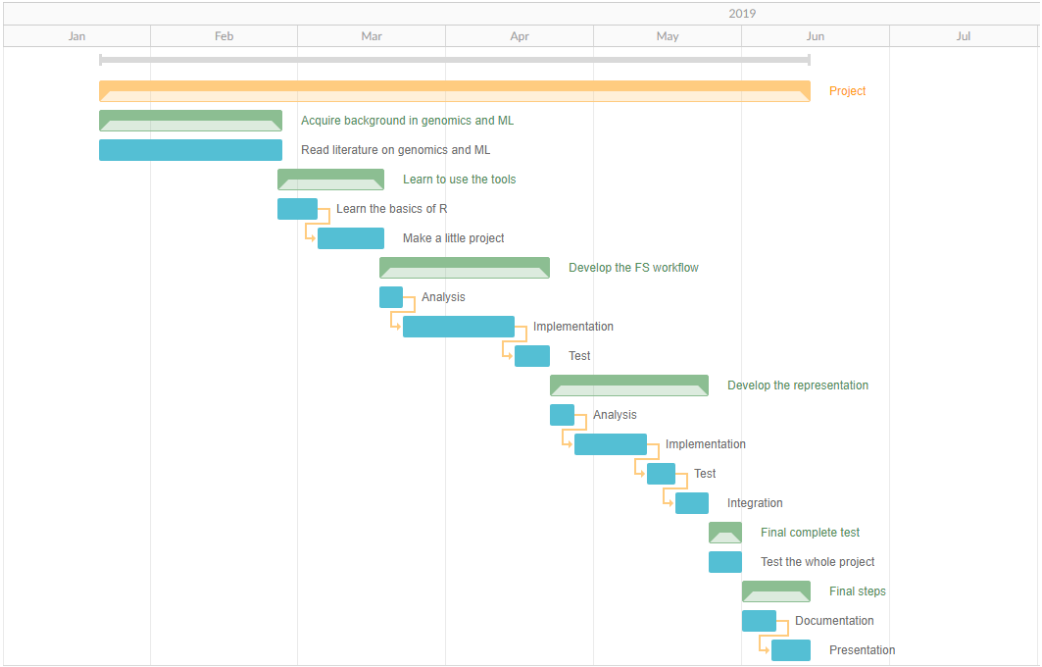

Figure 3.1: Gantt chart of the project, generated with `https://app.ganttpro.com`

3.5 Alternatives and action plan

The biweekly meetings and continuous communication via email with the professor will allow us to make as many changes as needed to the original planning of the project. However, the duration of this project has been set with a margin of a few days between the planned end and the start of the presentations, giving us time to spend on changes we could possibly need or just time to recover because of possible external problems. All the steps of this project are important so we can not skip a single one of them if a

previous time is taking us too long to finish, which is also a reason why these extra days may become of good use. Despite this, as said before, the communication with the professor will be very important to overcome the problems we might encounter during the development as soon as possible.

In conclusion, the project is feasible in the 580 planned hours of development.

Some of the problems we might find to do everything in the planned span of time will be explained below.

3.5.1 Complexity of the workflow

It is not a trivial task to find a workflow that returns us precise and correct results relatively fast, which is why research on the different algorithms and techniques will have to be made apart from the first readings of the project, since there will be tools that are already made and uploaded in packages ready to download and use, but the information about them will still be very important to know and understand so as to know what are we actually going to do with the data. We don't expect a lack of documentation because all of these techniques and algorithms should have been made according to a research that hopefully will be easily found online. However, if this is not the case, we will be forced to find a different path for the workflow because of the importance of the knowledge about everything it is doing from start to end.

3.5.2 Bugs

Since, as said before, R is a sort of new language to me, it is very probable that bugs will appear in the code and changes will have to be made. However, these are generally easily solvable problems and will not delay us more than a few hours.

3.5.3 Optimizations

Once the code is complete some parts may execute slower than expected. Notice that a "fast" execution will take minutes, and if not properly coded, parts of the code as simple as the data import may take even hours to finish. This is not acceptable because of the time it would take to test it just once, and even if the results were correct enough, the time needed to execute it at least once would make users lose a lot of time waiting for them. This could cause delays, but it is something we would have been working on since the beginning in order to avoid this kind of bit problems in the end, because of the fact that we would have found them very early in the development process anyway.

3.6 Changes regarding the original planning

In Figure 3.1 we can see the original planned schedule, specifying the tasks that had to be performed and the time they originally needed to be fully done. Originally, the project was planned to take no more than 5 months. However, a big adjustment had to be made, doubling the planned time up to 10 months. The reasons for this are:

- Having to attend more university classes.
- Working part-time until July, then starting a full-time job.
- An increase in the complexity of the originally planned project and its features.
- Other personal reasons.

Thus, a readjustment was made, allowing me to have twice the time to perform all the tasks and deliver the project completely finished. However, the objectives of the project are not only the same as the originally planned, but they go further: in the beginning, the planned project was supposed to analyze data related to the diagnose of Amyotrophic Lateral Sclerosis (ALS). However, we decided that the project could have a better and wider scope, and thus it was redesigned, allowing the analysis of any data, which is related to the third item in the previous list. The tasks, however, still remain the same and have the same duration, except for developing the workflow and the representation, which would take all the extended time the project was given.

In the original planning, the time schedule planned produced the times specified in Table 3.1. However, having re-planned the project, we need to recalculate the time spent on the third and fourth task. The dedicated time was of one hour from Monday to Friday, and five hours on Saturdays, Sundays and festive days. As said, the only modified tasks were developing the FS workflow and the representation. These tasks were supposed to take around one month to be finished each, from March the 18th to May the 19th, but with the new schedule they were finished on around August the 18th. Thus, recalculating the time now, we get that it took $(1 \times 5 + 4 \times 2) \times 22 = 286h$. If we add four hours for every festive day and holidays, we can add $10 \times 4 = 40$ more hours, for a total of 326 hours spent. This also makes 54 more hours than initially planned, for a total of 634 for the entire project.

Chapter 4

Economic planning

In this part we explain the details of the cost of all the resources we will need in order to develop the project, including hardware, software and human resources.

To calculate the amortizations, we will use the following formula:

$$UH * \frac{P}{Y * 365 * H}$$

UH = Usage Hours

P = Price of the item

Y = Years of useful life

H = Hours of usage a day

4.1 Direct Costs

4.1.1 Hardware

Here we show a table with the different costs of the hardware we will need for the development process. Since no material resources are needed apart from a computer, that is the only item we will put in the table.

Product	Price	Units	Useful life	Amortization
Lenovo ThinkPad X260	850 €	3	5 years	67.53 €
Total	**2550 €**			**202.60 €**

Table 4.1: Costs of the hardware

4.1.2 Software

Most of the resources we will use are digital, and we will try to make use of as many open source and free resources as possible. Here we show them.

Product	Price	Units	Useful life	Amortization
Kubuntu 18.10	0 €	3	∞	0 €
R 3.5.2	0 €	2	∞	0 €
RStudio Desktop 1.1.463	0 €	2	∞	0 €
LaTeX	0 €	3	∞	0 €
GitHub	0 €	2	∞	0 €
LibreOffice 6.2	0 €	3	∞	0 €
Total	**0 €**			**0 €**

Table 4.2: Costs of the software

4.1.3 Human Resources

A project is not developed by itself, so we will need people of different profiles to make it in time who, of course, will be paid for it. We will have a project manager, but since the main tasks of the entire development process are coding and analyzing data, we will assume that the manager will work less hours than the rest of the people involved.

In the following charts we will break down the costs of the human resources, showing the cost of each task and then the totals.

All the salary data has been obtained from `https://www.payscale.com/`, which shows us the average salaries for different jobs according to the city, experience and sector.

Profile	**Salary**	**Units**	**Estimated work time**	**Cost**
Project manager	15.65 €/h	1	290 h	4538.50 €
Software engineer	12 €/h	1	580 h	6960 €
Data scientist	13.50 €/h	1	580 h	7830 €
Total				**19328.50 €**

Table 4.3: Costs of the human resources

So, if we assume each hour of work will cost us $19328.50/580 = 33.325$ €, and that all members of the team will work equally on every task:

Task	Estimated duration	Cost
Acquire background in genomics and ML	140 h	4665.50 €
Learn to use the tools	84 h	2799.30 €
Develop the FS workflow	140 h	4665.50 €
Develop the representation	132 h	4398.90 €
Final complete test	28 h	933.10 €
Final steps	56 h	1866.20 €
Total	**580 h**	**19328.50 €**

Table 4.4: Costs of the human resources per task

4.1.4 Total Direct Costs

Type	Cost
Hardware	2550 €
Software	0 €
Human Resources	19328.50 €
Total	**21878.50 €**

Table 4.5: Total direct costs

4.2 Indirect Costs

The main indirect costs we will have are electricity and internet. For the electricity, we will calculate it considering these constants:

Price of electricity in Spain = 0.1255€/kWh

Hours of usage of the laptops = 580 h

Power of the laptop = 3 * 45 W = 135 W = 0.135 kW

$$0.1255\ €/\text{kWh} * 580\ \text{h} * 0.135\ \text{kW} \approx 9.83€$$

About the internet, a 50Mb internet connection would be more than enough to satisfy all of our requirements. This costs 31 € a month, and since the duration of the project is approximately 5 months, we would have to pay 31€/month * 5 months = 155 €.

The total of our indirect costs is in the following table:

Item	**Cost**
Electricity	9.83 €
Internet	155 €
Total	**164.83 €**

Table 4.6: Total indirect costs

4.3 Total direct and indirect costs

Type	**Cost**
Direct	21878.50 €
Indirect	164.83 €
Total	**22043.33 €**

Table 4.7: Total direct and indirect costs

4.4 Contingency Costs

We tried to plan the project as accurately as possible. However, as stated in previous reports, some inconveniences could happen that would delay the project some days. Since the presentations start the first week of July, there could only be two additional weeks of development. This does not affect the price of the hardware and the software, but it does add to the cost of the human resources, the electricity and the internet. Here are the additional costs for everything:

Type	**Cost**
Human resouces	2304.40 €
Electricity	0.95 €
Internet	31 €
Total	**2336.35 €**

Table 4.8: Contingency costs

4.5 Total costs

To calculate the total costs for this project, we need to also add a 21% VAT. This leaves us with these costs:

Type	**Cost**
Direct	26472.99 €
Indirect	199.44 €
Contingency	2826.98 €
Total	**29499.41 €**

Table 4.9: Total costs

4.6 Budget tracking

In order to keep track of the money we will be having to pay, as well as the variations we might have to make and the extra expenses we could have, we will check the amount of worked hours at the end of each task and calculate a deviation of the cost. This deviation will be calculated as follows:

$$Costdeviation = (EC - RC) * RH$$

EC = Estimated cost of the task

RC = Real cost of the task

RH = Real hours the task took to perform

This way, we will be able to see how well estimated the cost and duration of each task were and if we are delaying the project too much. As explained before, we will have extra time to finish it in case we need it, but with this deviation we will be able to see if the time we are taking is close to the

expected or inside the margin given, or if we are going to have to perform some task faster than estimated in order to stay in it.

4.7 Economic deviations

There were also changes in the economic planning of the project too.

First of all, the redefinition of the temporal planning in the previous section makes obvious that there would be changes in the economic planning too. In the original planning, we set that the human resources costs (Table 4.3).

However, with the new planning, we have these costs:

Profile	**Salary**	**Units**	**Estimated work time**	**Cost**
Project manager	15.65 €/h	1	317 h	4961.05 €
Software engineer	12 €/h	1	634 h	7608 €
Data scientist	13.50 €/h	1	634 h	8559 €
Total				**21128.05 €**

Table 4.10: Costs of the human resources: re-planned

Not only that, but we also need to modify the costs of electricity accordingly. With the new planning, though, the costs are these:

$$\text{Price of electricity in Spain} = 0.1255€/\text{kWh}$$
$$\text{Hours of usage of the laptops} = 634 \text{ h}$$
$$\text{Power of the laptop} = 3 * 45 \text{ W} = 135 \text{ W} = 0.135 \text{ kW}$$
$$0.1255 €/\text{kWh} * 634 \text{ h} * 0.135 \text{ kW} \approx 10.74€$$

And of course, the cost of internet also grows. Initially we had planned that internet would be used for 5 months, for a total of 155€. Knowing that the months spent on the project would be twice the initial ones, the internet costs grow to 310€.

Because of the new planning of the project, the costs have grown noticeably. However, we already took this into consideration and added contingency costs to the initial planning to avoid this (Table 4.8). In the following table we calculate the difference between the initial costs and the new costs, and between this difference and the contingency costs:

Type	**Initial costs**	**New costs**	**Rise of costs**	**Remaining**
Human resouces	19328.50 €	21128.05 €	1799.55 €	504.85 €
Electricity	9.83 €	10.74 €	0.91 €	0.04 €
Internet	155 €	310 €	155 €	-124 €
Total	**19328.50 €**	**21128.05 €**	**1955.46 €**	**380.89 €**

Table 4.11: Difference between initial and actual costs

As we can see, the contingency costs were more than enough to deal with the rescheduling of the project, so even though the costs of several parts have changed, we already had a plan to deal with them.

In conclusion, although the economic planning has also changed, a good initial planning regarding the contingency has allowed us to not change the initial costs.

Chapter 5

Sustainability

5.1 Environmental

In this project, the only material resources used have been the computer and the electricity.

According to Lenovo[38], their company has been given several social and environmental sustainability certifications, so we can consider using one of their laptops, although still requires the use of natural resources, does not become a threat to the nature and the environment.

In the case of electicity, since the project has been done at home, I have used the energy that we have contracted here. It does not come from renewable sources. However, Being the computer a laptop, it can be recharged and used without being connected to the power, which makes it consume less energy and become much more ecologic than a regular PC.

Once the project is done, no more energy nor computers will be needed from my part since the code and its documentation will be uploaded in a remote repository and storage.

5.2 Social

This project is not only a source of knowledge about computer science for the developer, but also a very good way to learn about some basics of biology. I have learned about the properties of genes, proteines, and in general any molecule that belong to the omics fields, the way they interact with each other, how important they are in the functions they perform in an organic body, and how to treat and analyze them using computer science.

It also provides a great tool for studying those data that can be used by both experienced and novice people, for it has a great interface and feedback every step of the workflow and, as proved in previous sections, is also very accurate and fast. For this reason, it is excellent to be used in the previous stages of a new study in order to have a first quick answer to the formulated hypothesis, and also have some guidance to understand which could be the more interesting parts of the data to look more into.

In this way, it is nothing but beneficial for all sorts of people, for the purpose of research and education, and could be helpful in the path leading to the solution of many biological problems and mysteries.

5.3 Economic

The cost of every task and resource of this project is clearly stated in previous sections. This project is not cheap, even less if we take into consideration that it is supposed to be open source and free to use by every single person in the world. However, the costs are mostly human, since the costs of electricity, internet and hardware are something that almost every person including me has to pay regardless of what they are working on, and the software is all free.

Also, since this was a project for university supposed to be done by only me, the human costs in reality are also null, so economically the making of this project has been absolutely free. Additionally, it will be free software, thus being available for every person that is interested in using it for any purpose, even modifying it if desired, and becoming not an economical problem at all.

5.4 Sustainability Matrix

Here we present the sustainability matrix in a scale from 0 to 10. In case of the risks, instead we will show negative points, that will be the amount of points to subtract from the other sections.

	PPP	**Exploitation**	**Risks**
Environmental	8	10	-1
Social	10	10	0
Economic	10	10	0

Table 5.1: Sustainability matrix

Chapter 6

Conclusion

During the development of this project I have been in touch with many things completely new for me. The language R was the only thing I was a little familiar with, and it still became a struggle to change my mindset to work with something so different from what I usually use. Along the year, I have been in a constant process of learning, understanding and coding everything from mathematical formulas to even the user interface of a web page.

The result has been a tool that efficiently solves all the problems proposed at the very beginning:

- I learned a lot about the state of the art in the field of Machine Learning. I had to research through many scientific documents, journals, code... To find inspiration in what I wanted to do, and to find the tools and ways to achieve my proposed objectives. Also, the subject majority of documents I went through was biology and computer science together, so I also learned about the ways molecules interact with each other and how and why they are important in all functions of organic bodies.

- I built a complete workflow, with a big rate of customization, that was able to reduce the initial amount of data to a very small part of it losing a very small part of relevant information in the process.

- It was tested using it to analyze the famous ALL/AML leukemia dataset, comparing the results to the ones of many other studies, and successfully asserting that they were, indeed, very correct, and more than that, they all were generated and returned very quickly.

- I developed a web user interface, allowing the upload of the data in multiple formats, customization of multiple parameters of the workflow, and exhaustive but quickly obtained feedback as plots and graphs depicting the analysis and transformations the data goes through.

- Apart from that, an additional objective accomplished that was not in the initial ones, but that I still developed, was the generation of a final downloadable report of all the workflow the data has gone through, including all the plots and graphs generated in the way.

In the end we can see that this project successfully accomplished all the initially set objectives, and even more, although the initial planning had to be remade: we have a complete, fast, accurate and user-friendly tool to analyze and study biological data in amounts as big as required. This project is stored in its own repository, which can be found in the following URL: `https://github.com/Jrryy/TFG`

6.1 Future work

I have been working on the development of this project for almost a year, and although it has ended up being very robust, I am aware of its weak spots. There is a list of certain parts of it that could be improved, and as a plan for future work, that should be the first of all to work on:

- First of all, it accepts data in different formats, but those formats are very specific and only a slight difference would make the system not recognize the data and throw an error message. It would be great to accept data files that are represented in the most common formats around the scientific world.

- Sometimes, the plots are not 100% correctly drawn. The title, for example, appears cut out in several occasions, but most commonly in the pdf report, which might cause certain difficulties to understand what those graphs represent.

- The code went through a lot of trial and error writings, and for this reason some parts of it are difficult to read and some add execution time that is probably unnecessary. Writing clean code is the first rule for a programmer, for other people and for himself to have an easier time understanding and improving the code, so a refactor of it all would be also very important.

- There are many checks for errors throughout the entire code, making sure that the user does not try to break the software or make it crash at some point, but there probably are more parts and ways to make it crash that I have not taken into consideration. Thus, the project would need a process of quality assurance, trying to exploit every possible part

and adding error checks in every single one of them, to make it 100% robust.

These being the weakest points of the project to work on in the future, there is also a list of other features and upgrades that it could go through, not being errors or limitations, but more of new additions for the users to ,in general, improve their experience:

- Make some plots interactive. For example, it would be great if we, as users, instead of selecting the number of variables writing a number we could click on the plot whever we want the limit to be.
- Allow the user to select the amount of variables per component that the PLS-DA tuning tests.
- Allow the user to deactivate the tuning and select the components and variables to analyze with sPLS-DA by himself.
- Parallelize the different parts of the code. While it is true that I already added a parallelism option to the ReliefF step (basically, it will check if there are enough cores available to parallelize that whole process), it does not seem to work properly. Either way, the Fisher Scoring and ReliefF steps can both be parallelized, which would improve the execution time of the workflow, being overall useful with bigger datasets.
- Deploy the project to an online server. So far, this project can be downloaded and used locally by any person. However, it would also be an improvement if people could use it through their browser, without having to download the project and all the dependencies it has, and it could be used by multiple users simultaneously.
- Allow analysis of data with more than two classes.

Bibliography

[1] Golub TR, Slonim DK, Tamayo P, Huard C, Gaasenbeek M, Mesirov JP, Coller H, Loh ML, Downing JR, Caligiuri MA, Bloomfield CD. "Molecular classification of cancer: class discovery and class prediction by gene expression monitoring." Science. 1999; 286:531-7.
https://doi.org/10.1126/science.286.5439.531

[2] Duong CN, Quach KG, Le N, Nguyen N, Luu K. "MobiFace: A Lightweight Deep Learning Face Recognition on Mobile Devices", Nov 2018
https://arxiv.org/pdf/1811.11080.pdf

[3] Bhowmick A, Hazarika SM. "Machine Learning for E-mail Spam Filtering: Review, Techniques and Trends", Jun 2016
https://arxiv.org/pdf/1606.01042.pdf

[4] Li D, Zhao D, Zhang Q, Chen Y. "Reinforcement Learning and Deep Learning based Lateral Control for Autonomous Driving", Oct 2018
https://arxiv.org/pdf/1810.12778.pdf

[5] Lynch C. "Big data: How do your data grow?" Nature. 2008; 455(7209):28-9.
https://doi.org/10.1038/455028a PMID: 18769419

[6] Perez-Riverol Y, Bai M, da Veiga Leprevost F, Squizzato S, Park YM, Haug K, et al. "Discovering and linking public omics data sets using the Onics Discovery Index." Nature biotechnology. 2017; 35(5):406-9. https://doi.org/10.1038/nbt.3790 PMID: 28486464

[7] Saeys Y, Inza I, Larranaga P. "A review of feature selection techniques in bioinformatics." Bioinformatics. 2007; 23(19):2507-17. https://doi.org/10.1093/bioinformatics/btm344 PMID: 17720704

[8] Perez-Riverol Y, Kuhn M, Vizcaíno JA, Hitz M-P, Audain E "Accurate and fast feature selection workflow for high-dimensional omics data." PLoS ONE. 2017; 12(12): e0189875. https://doi.org/01.1371/journal.pone.0189875

[9] Bellman R. "Dynamic programming and Lagrange multipliers." Proceedings of the National Academy of Sciences. 1956; 42(10):767-9

[10] Hughes, G.F. "On the mean accuracy of statistical pattern recognizers". IEEE Transactions on Information Theory. 1968; 14 (1): 55–63. https://doi.org/10.1109/TIT.1968.1054102

[11] Michalak K, Kwasnicka H. "Correlation-based feature selection strategy in classification problems. International Journal of Applied Mathematics and Computer Science." 2006; 16:503-11.

[12] Wang Y, Tetko IV, Hall MA, Frank E, Facius A, Mayer KE, et al. "Gene selection from microarray data for cancer classification-a machine learning approach." Computational biology and chemistry. 2005; 29(1):37-46. https://doi.org/10.1016/j.compbiolchem.2004.11.001 PMID: 15680584.

[13] Ringner M. "What is principal component analysis?" Nature biotechnology. 2008; 26(3):303-4.
http://doi.org/10.1038/nbt0308-303 PMID: 18327243.

[14] Phuong TM, Lin Z, Altman RB. "Choosing SNPs using feature selection." Proceedings / IEEE Computational Systems Bioinformatics Conference, CSB. IEEE Computational Systems Bioinformatics Conference. 2005; 301-309.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.511.7735&rep=rep1&type=pdf PMID 16447987.

[15] Lal TN, Chapelle O, Weston J, Elisseeff A. "Embedded Methods." Feature Extraction. Studies in Fuzziness and Soft Computing. 2005; 207.
https://doi.org/10.1007/978-3-540-35488-8_6

[16] Song Q, Ni J, Wang G. "A fast clustering-based feature subset selection algorithm for high-dimensional data." IEEE Trans. Knowl. Data Engineering. 2013; 25(1):1-14.

[17] Boulesteix AL and Strimmer K. "Partial least squares: a versatile tool for the analysis of high-dimensional genomic data." Briefings in bioinformatics, 2006; 8(1):32-44.

[18] Nguyen DV and Rocke DM. "Tumor classification by partial least squares using microarray gene expression data." Bioinformatics, 2002; 18(1):39–50.
https://doi.org/10.1093/bioinformatics/18.1.39

[19] Barker M and Rayens W. "Partial least squares for discrimination." J. Chemometrics, 2003; 17:166-173.
https://doi.org/10.1002/cem.785

[20] Lê Cao KA, Boitard S, Besse P. "Sparse PLS discriminant analysis: biologically relevant feature selection and graphical displays for multiclass problems." BMC Bioinformatics, 2011; 12(1).
https://doi.org/10.1186/1471-2105-12-253

[21] Shafranovich Y. "Common Format and MIME Type for Comma-Separated Values (CSV) Files." IETF Tools, 2005.
https://tools.ietf.org/html/rfc4180#section-2

[22] Q. Gu, Z. Li, J. Han. "Generalized Fisher score for feature selection." arXiv preprint, 2012.
https://arxiv.org/abs/1202.3725

[23] Charu CA. "Data Classification: Algorithms and Applications." Chapman & Hall/CRC, 2014.

[24] Sikonja MR, Kononenko I. "Theoretical and empirical analysis of ReliefF and RReliefF." Machine Learning, 2003; 53(1-2):23–69.
http://lkm.fri.uni-lj.si/rmarko/papers/robnik03-mlj.pdf

[25] Kira K, Rendell LA. 1992. "A practical approach to feature selection." Proceedings of the ninth international workshop on Machine learning (ML92), 1992; 249-256.
https://sci2s.ugr.es/keel/pdf/algorithm/congreso/kira1992.pdf

[26] Tan Y, Shi L, Tong W, Gene Hwang G, Wang C. "Multi-class tumor classification by discriminant partial least squares using microarray gene expression data and assessment of classification models." Computational Biology and Chemistry, 2004; 28(3):235-243.
https://doi.org/10.1016/j.compbiolchem.2004.05.002

[27] Nguyen D, Rocke D. "Tumor classification by partial least squares using microarray gene expression data." Bioinformatics, 2002; 18:39.
http://dx.doi.org/10.1093/bioinformatics/18.1.39

[28] Fordellone M, Bellincontro A, Mencarelli F. "Partial least squares discriminant analysis: A dimensionality reduction method to classify hyperspectral data" 2018.
https://arxiv.org/pdf/1806.09347.pdf

[29] Brereton RG, Lloyd GR. "Partial least squares discriminant analysis: taking the magic away." Journal of Chemometrics, 2014; 28(4):213–225.
https://doi.org/10.1002/cem.2609

[30] Ruiz-Perez D, Narasimhan G. "So you think you can PLS-DA?" Bioinformatics, 2017.
https://doi.org/10.1101/207225

[31] Chung D, Keles S. "Sparse partial least squares classification for high dimensional data." Statistical applications in genetics and molecular biology, 2010; 9(1)

[32] Pearson K. "On lines and planes of closest fit to systems of points in space" The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 1901; 2(11):559-572
https://doi.org/10.1080/14786440109462720

[33] Robnik-Sikonja M. "CORElearn v1.53.1" 2018.
http://lkm.fri.uni-lj.si/rmarko/software/

[34] Déjean S, Lê Cao KA, González I, et al. "mixOmics v3.9" 2019.
http://mixomics.org/

[35] Robnik-Sikonja M. "CORElearn v1.53.1: attrEval method details" 2018.
https://www.rdocumentation.org/packages/CORElearn/versions/1.53.1/topics/attrEval#l_details

[36] Bø TH, Jonassen I. "New feature subset selection procedures for classification of expression profiles." Genome Biology, 2002; 3(4):research0017
https://doi.org/10.1186/gb-2002-3-4-research0017

[37] Mahmoud AM, Maher BA. "A Hybrid Reduction Approach for Enhancing Cancer Classification of Microarray Data." International Journal of Advanced Research in Artificial Intelligence (IJARAI), 2014; 3(10)
http://dx.doi.org/10.14569/IJARAI.2014.031001

[38] Lenovo's sustainability report.
https://www.lenovo.com/us/en/social_responsibility/sustainability_reports/