ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA DE LA SALUD
MENCIÓN BIOINFORMÁTICA

Estudio y búsqueda de marcadores genéticos mediante el uso de Deep Neural Networks

Deep Neural Networks to find genetics signatures

Realizado por
Fernando Moreno Jabato
Tutorizado por
José Manuel Jerez Aragonés
Departamento
Dpto. Lenguajes y Ciencias de la Comunicación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Septiembre 2016

Fecha defensa:
El Secretario del Tribunal

Keywords: omics, Machine Learning, Deep Learning, DNN, ANN, cancer, microarray, data mining, Big Data.

Abstract

This document contais the final dissertation ot the degree student Fernando Moreno Jabato for the studies Grade in Health Engeneering, speciality on Bioinformatics, of University of Málaga. This dissertation have been performed with the supervision of Dr. José Manuel Jerez Aragonés from the Departament of Lenguajes y Ciencias de la Comunicación.

The project title is **Deep Neural Networks to find genetics signatures** and is focused on the development of a bioinformatic tool oriented to identification of relationships between an attribute set and concret factor of interest on medicine. To do this, a tool was designed with the capacity of handle data sets from microarrays of different types. Microarrays was selected as preferent technology because it's the most extended and accessible techonologie on health and biology fields nowadays.

Once implemented the tool, an experiment was performed to evaluate the efficiency of this tool. The experiment uses prostate cancer related datasets from trascriptomics microarrays containing patients of prostate cancer and some normal individues.

The results obtained in the experiment shows an improvement offered by the new Deep Learning algoritms (specifically, Deep Neural Networks) to analyze and obtain knowledgement from microarrays data. Besides, has been observed an improvement of efficiency and the beat of computational barriers that traditional Artifical Neural Networks suffered allowing apply this bioinformatics tools of new generation to masiva data sets.

Palabras clave: ciencias ómicas, Aprendizaje Computacional, Deep Learning, DNN, ANN, cáncer, microarray, minería de datos, Big Data.

Resumen

Este documento contiene el Trabajo de Fin de Grado del alumno Fernando Moreno Jabato, estudiante del Grade in Health Engeneering, speciality on Bioinformatics, en la University of Málaga. Dicho proyecto se ha realizado con la tutorización de Dr. José Manuel Jerez Aragonés, profesor perteneciente al Departament of Lenguajes y Ciencias de la Comunicación.

El proyecto recive el título de **Estudio y búsqueda de marcadores genéticos mediante el uso de Deep Neural Networks** y se centra en el desarrollo de una herramienta bioinformática orientada a la identificación de relaciones entre un set de atributos y un factor concreto de interés en la medicina. Para ello se ha disaño una herramienta capaz de manejar datos procedentes de microarrays de diferentes tipos ya que es la tecnología más expandida y accesible en la actualidad para este campo del conocimiento.

Una vez implementada la herramienta se ha realizado un experimento para probar la eficacia de la misma. El experimento ha utlizado los resultados obtenidos de un microarray del ámbito de la transcriptómica y el set de datos en cuestión correspondía a un grupo de estudio con individuos normales y otros individuos que padecen de tumores de cancer de prostata.

Los resultados obtenidos en el experimento muestran una clara mejoría de los nuevos algoritmos de Deep Learning, en concreto, de las Deep Neural Networks, para analizar y obtener conocimiento de datos obtenidos de microarrays. Además se ha observado una mejora de la eficaciencia y la rotura de las barreras computacionales que los algoritmos tradicionales (Artificial Neural Networks, ANNs) padecían, permitiendo poder aplicar estas herramientas bioinformáticas de nueva generación a conjuntos de datos masivos.

Contents

1 - Introduction

This document is a dissertation of the degree student Fernando Moreno Jabato for the studies in Grade in Health Engeneering, speciality on Bioinformatics, of University of Málaga. The project has the title "Deep Neural Networks to find genetics signatures" wich wants prove the viability of use Deep Neural Networks (DNN) to identify relationships between genes and clinical symptoms and create new reliable diagnosis and prediction tools. To do this, in this project a specifical software has been implemented that can handle all needed variables for identifying genes relationship with a spcific issue and generate DNN models to make predictions using those genes. The entire project has been implement on R language.

1.1 Motivation

Nowadays masive amounts of data are generated and stored constantly in all society fields. It's known that this data can be stored on different variables and be studied to improve and obtain knowledgment in several fields like economy, sociology or medicine. Currently, this way of masive minning of information is known as Big Data and it's being implemented on several business and social purposes companies.
In Healt sector, data is being generated massively each day. On this information we can find diseases presences by location, blood type, metabolic activity registers of a specific patient and, increasingly, genomes, genotypes and other genomics data from specific patients. This new use of Omics on medicine are part of the new trend to personalized medicine that are increasing on the recent years. This personalized studies also can offer information useful for global health creating realtionships between data with symptoms, treatments or any other useful information to clinical medicine and society.
Is on this point where Bioinformatics can offer tools for both medicine ways (personal and global). In first place, bioinformatics generate and contribute to diagnosis and decision making with biological data analysis tools. Nowadays the use of tools as electronic phonendoscopes or electrocardiographs with analysis software integrated, on hospitals is habitual. These two examples are a sample of the current tools that automate common process on clinical medicine that give the traditional information and several analysis of this information which generate more knowledgement that the sanitary can can use to making decisions.
In other way, link this generated knowledgement is a newflanged concept that is being

implemented on other fields using ontologies which generate more knowledgement using reasoners already implemented.
My project proposal is about the first way. I've observed a huge potential on genetic data to relate symptoms, inmunities or whatever other classifiable factor. That are the reasons because I propose implement a bioinformatical tool which can handle genetical data and relate it with our interest factors. This tool target will be the identification of high related variables on these genetic data sets to generate predictors and diagnostic tools which could be applied on medicine. To implement this tool I'll use already known statistical methods to select and filter variables. To improve the current process I also will use some new machine learning algorythms known as *Deep Learning* algorythms. Particulary I will use Deep Neural Networks (DNN) because their excellent results in other fields like Automatic Speech Recognition (ASR)[1] or image identification[2]. Traditional Artificial Neural Networks (ANN) are already used to generate cancer predictors[3] or other medicine interest factors, that's the reason because I look for improve the current tools using DNNs to generate new more precise predictors.
If reults are good enough, this application would be a new tool to integrate bioinformatic on medicine and, also, to increase the knowledgement that Big Data and personal medicine can contribute to this one and to global health.

1.2 State of the art

Currently, Deep Neural Networks have been used in several areas as speed automatic recognition(ASR)[4] or in image recognition[5,6] as more remarkable examples.
It's difficult find publications of real applications of DNN in omics field.
More easy is find publications about real applications of traditional artificial neural net-

[1]Geoffrey Hinton, et.al. " Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups ". I EEE Signal Processing Magazine (Volume:29 , Issue: 6),Pages 8297,doi: 10.1109/MSP.2012.2205597

[2]Alex Krizhevsky, et.al. "ImageNet Classification with Deep Convolutional Neural Networks"

[3]José M. Jerez Aragonés, et.al. "A combined neural network and decision trees model for prognosis of breast cancer relapse". Artificial Intelligence in Medicine 27 (2003) 45–63.

[4]Geoffrey Hinton, et.al. " Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups ". I EEE Signal Processing Magazine (Volume:29 , Issue: 6),Pages 8297,doi: 10.1109/MSP.2012.2205597

[5]Alex Krizhevsky, et.al. "ImageNet Classification with Deep Convolutional Neural Networks"

[6]Dan Ciregan, et.al. Multi-column deep neural networks for image classiffcation. Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on 16-21 June. doi 10.1109/CVPR.2012.6248110.

works (ANN) in omic fields[7,8] or medicine fields[9,10,11].
Thats the reason because we look for improve the current results obtained in these studies using the new DNN technology as has been used in other fields.

1.3 Objectives

This project objectives are:

- Implement a support software to identify genetics signatures using DNNs.
  - Filtering and identification of genetics variables.
  - Generate DNN models.
  - GeneraTe preodictions using DNN model.
- Research the reliability and effectiveness of DNN models generated to be used with genetic data.
- Research effectiveness differences between Artificial Neural Networks (ANNs) and DNNs to find genetic signatures.

Besides these particular objectives, there're generals objectives for all dissertations. These are:

- Interrelate different concepts learned during grade studies.
- Perform a project related with one of the specialties of the grade studied.
- Perform a project autonomously.

1.4 Metodology

THe metodology that will be used on this project to implement the software and study the results will be the **waterfall model**. This model is a sequential (non-iterative) model where each project fase is not performed until the precedent is finished.
Our project phases will be:

1.- Identify the project needs.
2.- Research about the State of the art.
3.- Design functional and non-function requirement.
4.- Logical design of the software.
5.- Software implementation.
6.- Software evaluation.
7.- Generating results.

[7]José M. Jerez Aragonés, et.al. "A combined neural network and decision trees model for prognosis of breast cancer relapse". Artificial Intelligence in Medicine 27 (2003) 45–63.

[8]Mateo Seregni, et.al. Real-time tumor tracking with an artffcial neural networksbased method: A feasibility study. Acta Medica Iranica, 2013

[9]Farin Soleimani, et.al. Predicting Developmental Disorder in Infants Using an Artiffcial Neural Network. Acta Medica Iranica51.6 (2013): 347-52.

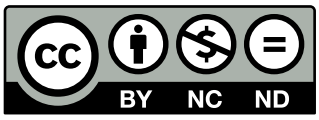
[10]Hon-Yi Shi, et.al. Comparison of Artffcial Neural Network and Logistic Regression Models for Predicting In-Hospital Mortality after Primary Liver Cancer Surgery.

[11]Filippo Amato, et.al. Artiffcial neural networks in medical diagnosis. Journal of Applied Biomedicine, vol.11, issue 2, pages 47-58, 2013

8.- Analysis of results.

These are the project phases to whitch is added in parallel the documentation development.

1.5 Licence



2 - Problem study

This is the technical study of the problem. Here, the functionalities that must be implemented are going to be analized and the software and strategies to implement it are going to be selected.

2.1 Functional requirements

The software must implement functionalities that satisfies the following requirements:

1.- Import data in different formats.
2.- Filter genetic variables.
3.- Generation of formulas using a set of variables.
4.- Divide a data set un random subsets (KFold strategy).
5.- Generate a Deep Neural Network using a formula.

2.2 Non-functional requirements

The software must satisfies the following non-functional requirements:

1.- The software must can load microarrays data sets to expedite the tool integration (microarray is the most already implemented on clinics and hospitals).
2.- The software must be implemented on R language to take advantage of his high capacities on Big Data, Estatistiscs and Machine Learning.
3.- The software must handle exceptions generated to avoid non-controlated program abortion.
4.- The software must work on version 3.1.1 (R).
5.- The software must atomize the functionalities and implement it as function to subserve parallelization.
6.- The software must atomize the functionalities and implement it as function to subserve the code maitenance without affect the final user interface.
7.- The software must offer the higher amount of attributes to subserve the configuration of the pogram.

2.3 Software

The languaje used will be R. The external software that will be used must be implemented also in R. The external software that will be used is:

- RStudio: IDE to work with R.
- Sublime Text 3: plain text files editor.
- R packages:
  - **FSelector:** filtering and variable selection.
  - **Caret:** data set divider (kfold strategy).
  - **H2O:** DNNs package.
  - **R.oo:** Oriented Object Programming on R.
  - **nnet:** ANNs already implemented package.
  - **AUC:** ROC and AUC implementations.
- Tex Live: LaTeXcompiler.
- Git: version manager.

3 - Design

Now I study and perform the software logical design using as reference the functional and non-functional requirements. If we observe these requirements we can indentify the following functionality groups:

- **Data management (inputs and partitions):** corresponds to function requirements 1 and 4.
- **Variable handling (filtering and selection):** corresponds to functional requirements 2 and 3.
- **Deep Neural Networks handling:** corresponds to functional requirement 5.

For implement these functional groups I'm ging to take into acount the non-functional requirements. The two first requirements dictates the software and strategy to ease the tool implantation, these don't alter our design just in the way of the language imposition. The two following corresponds to implementation requirements and will be reflected on the use cases. The last three requirements suggests the best way to implement and organize the functional groups using, also, the biggest amount of relevant parameters posible.
Based on this, each functional group will be implemented on one or serveral functions and, taking advantage of this functionality groups, these functions will be stored on different files grouping by functionality. Following this guidelines I'll desing the described functionalities.

3.1 Data management

This section includes the data sets management logical design. The functional requirements that must be implemented are:

1.- Import data in different formats.
2.- Divide a data set un random subsets (KFold strategy).

Each requisites will be studied separately.

3.1.I Inputs

The programs must handle microarray files data and several R structures to use it. The data sets that will be used contains complex structures, for this reason the main data structure that will be used is the *data frame*. It means that our functions must handle this R structures.

Currently, R basic language implements functions with trivial use complexity that handle data leaded on data frames, for that reason specific software will not be implemented for this functionality.
Our first callenge is load tables from files. This tables nees be read, loaded and parsed to wanted format. To do this an specific function will be implemented to load and handle tables stored on files.
Also is usual that a data sets is composed by several files, at least a training data set and a testing data set, sometimes a third file with attributes information are included too. This means that a function that loads the two main data sets and relate the information of a possible third file with these two data sets must be implemented.

Data loading

This function must read a data table from an external file and parse int in a data frame. Necessary parameters to do this activity are:

- **Directory:** file wanted directory path.
- **File:** wanted file name (with extension).
- **Store:** variable where data frame will be stored.
- **Directories separator:** directories separator used by the operating system.
- **File separator:** data entries separator used on the data file.

Functions must return the following values:

- FINALIZATION WITHOUT ERRORS.
- ERROR IN DIRECTORY PARAMETER.
- ERROR IN FILE PARAMETER.

Function use case will be:

1.- If directory parameter is null: abort process and throw ERROR IN DIRECTORY PARAMETER signal.
2.- If file parameter is null: abort process and throw ERROR IN FILE PARAMETER signal.
3.- Read and load table from file.
4.- Parse table to data frame structure.
5.- Store data frame on given store variable.
6.- End with FINALIZATION WITHOUR ERRORS signal.

Atributes loading

The current tool version will not handle metadata included on attribute files, the only information that will be considered is the attribute names to be associated to data sets for a better understanding.
The function will load the row or column of attribute names and will return it in a vector. Necessary parameters to do this activity are:

- **Directory:** file wanted directory path.
- **File:** wanted file name (with extension).
- **Column:** boolean that indicates the order of the attribute names (row or column. TRUE indicates that are in column and FALSE in a row. (OPTIONAL)

- **Store:** variable where vector will be stored.
- **Directories separator:** directories separator used by the operating system.
- **File separator:** data entries separator used on the data file.

Functios must return the following values:

- FINALIZATION WITHOUT ERRORS.
- ERROR IN DIRECTORY PARAMETER.
- ERROR IN FILE PARAMETER.
- ERROR IN COLUMN PARAMETER.

Function use case will be:

1.- If directory parameter is null: abort process and throw ERROR IN DIRECTORY PARAMETER signal.
2.- If file parameter is null: abort process and throw ERROR IN FILE PARAMETER signal.
3.- If column parameter isn't a boolean or is null: abort process and throw ERRO IN COLUMN PARAMETER signal.
4.- Read and load col/row of attribute names from file.
5.- Parse col/row to vector structure.
6.- Store vector on given store variable.
7.- End with FINALIZATION WITHOUR ERRORS signal.

Data sets loading

This function uses the two precious functions to load a full dataset composed by, at least, a training and testing data files and, optionally, an attribute file.
Necessary parameter to do this activity are:

- **Directory:** data set files wanted directory path.
- **Training file:** wanted training file name (with extension).
- **Testing file:** wanted testing file name (with extension).
- **Attribute file:** wanted attribute file name (with extension). (OPTIONAL)
- **Training store:** variable where training data frame will be stored.
- **Testing store:** variable where testing data frame will be stored.
- **Attribute store:** variable where attributes vector will be stored.
- **Directories separator:** directories separator used by the operating system.
- **Training file separator:** data entries separator used on the training data file.
- **Testing file separator:** data entries separator used on the testing data file.
- **Attribute file separator:** data entries separator used on the attribute data file. (OPTIONAL)
- **Column:** boolean that indicates the order of the attribute names (row or column. TRUE indicates that are in column and FALSE in a row.

Functios must return the following values:

- FINALIZATION WITHOUT ERRORS.
- ERROR IN DIRECTORY PARAMETER.
- ERROR IN TRAINING FILE PARAMETER.
- ERROR IN TESTING FILE PARAMETER.
- ERROR IN ATTRIBUTE FILE PARAMETER.

- ERROR IN COLUMN PARAMETER.

Function use case will be:

1.- If directory parameter is null: abort process and throw ERROR IN DIRECTORY PARAMETER signal.
2.- If training file parameter is null: abort process and throw ERROR IN TRAINING FILE PARAMETER signal.
3.- If testing file parameter is null: abort process and throw ERROR IN TESTING FILE PARAMETER signal.
4.- If attribute file parameter is null: avoid 5,6,8,9,10 and 13 process points.
5.- If column parameter isn't a boolean or is null: abort process and throw ERRO IN COLUMN PARAMETER signal.
6.- Load training file.
- If any exception is thrown: abort and throw error to upper level.

7.- Load testing file.
- If any exception is thrown: abort and throw error to upper level.

8.- Load attribute file.
- If any exception is thrown: abort and throw error to upper level.

9.- Link attributes to training data frame.
10.- Link attributes to testing data frame.
11.- Store training data frame on given training store variable.
12.- Store testing data frame on given testing store variable.
13.- Store attribute vector on given attribute store variable.
14.- End with FINALIZATION WITHOUR ERRORS signal.

3.1.II Data division

The software must can divide data sets on random subsets to apply cross-validation. The number of division must be configurable. For that, a unic function that do a random division (kfold strategy) will be implemented.

KFold division

Necessary parameters to do this activity are:

- **Data:** data set that will be divided.
- **Divisions:** number of divisions that will be applied. Default: 10. (OPTIONAL)

Functios must return the following values:

- *Vector of subsets generated.*
- ERROR IN DATA PARAMETER.
- ERROR IN DIVISIONS PARAMETER.
- ERROR DIVSIONS BIGGER THAN SET.

Function use case will be:

1.- If data parameter is null: abort process and throw ERROR IN DATA PARAMETER signal.
2.- If divisions parameter is less or equal to zero: abort process and throw ERROR IN DIVSIONS PARAMETER signal.
3.- Apply random kfold division.
4.- Return the divisions vector generated.

3.2 Variable handling

This section includes the variable handling functionalities. The functional requirements related to these funcionalities are:

1.- Filter genetic variables. (Filtering)

2.- Generation of formulas using a set of variables. (Formula generation)

Each requisites will be studied separately.

3.2.I Filtering

The software must can filter and select variables from a massive variable set. To do this a statistical filtering and selection function must be implemented. This function must return a subset with a configurable minimum and maximum size that satisfies some requirements.

Variable filtering and selection

This function must can filter variables using different statistical methods to calculate correlation between variables and the main factor.

Necessary parameters to do this activity are:

- **Data:** data set to be filtered and used to train DNN models.
- **Attribute set:** attribute names set used to select and filter variables from data set. Default: data set given names. (OPTIONAL)
- **Filtering method:** statistical filtering method to be used. (OtCIONAL)
- **Maximum size:** maximum size of the returned attributes subset.
- **Minimum size:** minimum size of the returned attributes subset. Default: zero. (OPTIONAL)
- **Group:** indicates if the best minimum or best maximum subset is wanted. (OPTIONAL)
- **Store:** variable where generated vector will be stored.
- **DNN store:** variable where generated best DNN model will be stored. (OPTIONAL)
- **AUC:** variable where AUC obtained on predictions over the main data set. (OtCIONAL)
- **Testing AUC:** variable where AUC obtained on predictions over the testing data set. (OPTIONAL)
- **Collapse ratio:** value used to decide if the subset model accuracy have collapsed. Default: 0.03. (OPTIONAL)
- **Correlation minimum:** minimum correlation value with the main factor to accept an attribute. Default: 0.5. (OPTIONAL)
- **Testing data:** testing data set. (OPTIONAL)
- **Activation:** activation function that will be used in DNNs training. Default: *Tanh*. Possible values: "Tanh", "TanhWithDropout", "Rectifier", "RectifierWithDropout", "Maxout" or "MaxoutWithDropout". (OPTIONAL)
- **Neurons:** hidden neural layers that will be used on DNNs.

- **Iterations:** iterations that will be done over training data set on DNN train process. Default: 100. (OPTIONAL)
- **Seed:** seed used to start the DNN training process. (OPTIONAL)
- **Rho:** factor of learning ratio adaptation on time. Used on DNN training. (OPTIONAL)
- **Epsilon:** adaptative learning ratio used on DNNs training process. (OPTIONAL)
- **Threads:** number of CPU threads that will be used. Default: 2. (OPTIONAL)

Functios must return the following values:

- *Subset generated size.*
- ERROR IN DATA PARAMETER.
- ERROR IN TRAINING PARAMETER.
- ERROR IN FILTERING PARAMETER.
- ERROR IN GROUP PARAMETER.
- ERROR IN MAXIMUM PARAMETER.
- ERROR IN MINIMUM PARAMETER.
- ERROR IN ATTRIBUTE PARAMETER.
- ERROR MAXIMUM LESS THAN MINIMUM.
- IMPOSIBLE TO OBTAIN SATISFACTORY SUBSET.
- *Signals thrown by DNNs function..*

Function use case will be:

1.- If data parameter is null: abort process and throw ERROR IN DATA PARAMETER signal.

2.- If filtering parameter is not allowed (unless it's null): abort process and throw ERROR IN FILTERING PARAMETER.

3.- If group parameter is not allowed (unless it's null): abort process and throw ERROR IN GROUP PARAMETER signal.

4.- If maximum attribute is less than one: abort process and throw ERROR IN MAXIMUM PARAMETER signal.

5.- If minimum parameter is negative: abort process and throw ERROR IN MINIMUM PARAMETER signal.

6.- If minimum parameter is greater than maximum parameter: abort process and throw ERROR MAXIMUM LESS THAN MINIMUM signal.

7.- If attributes parameter is null: abort process and throw ERROR ATTRIBUTE PARAMETER signal.

8.- Filter variable set using selected statistical filtering method.

9.- Sort obtained results by correlation value.

10.- Select attributes with a correlation value greater than minimum given.

11.- If subset size is greater than maximum given: obtein maximum subset possible.

12.- Check group type selected:

- Case Best Minimum:

12.1.- Obtain minimum subset.

12.2.- Generate model with minimum subset.

12.2.1.- If any exception is thrown: throw exception to upper level.

12.3.- Evaluate generated model.

12.4.- Store evaluation value on current best model value variable.

12.5.- Add next best attribute to formula.

12.6.- Generate model with current subset.

12.6.1.- If any exception is thrown: throw exception to upper level.

12.7.- Evaluate generated model.

12.8.- If:

- Upgrading greater than collapse ratio: go to 12.5.
- Upgradin decrease or less than collapse ratio: continue.

12.9.- Delete last attribute added.

12.10.- Store results on best result variables.

- Case Best Maximum:

12.1.- Obtain maximum subset.

12.2.- Generate model with maximum subset.

12.2.1.- If any exception is thrown: throw exception to upper level.

12.3.- Evaluate generated model.

12.4.- Store evaluation value on current best model value variable.

12.5.- Delete worst attribute.

12.6.- Generate model with current subset.

12.6.1.- If any exception is thrown: throw exception to upper level.

12.7.- Evaluate generated model.

12.8.- If:

- Upgrading greater than collapse ratio: go to 12.5.
- Upgradin decrease or less than collapse ratio: continue.

12.9.- Add last deleted attribute.

12.10.- Store results on best result variables.

13.- Store subset selected on store variable given.

14.- Store best DNN model generated on given store variable.

15.- Store AUC values obtained on store variables given.

16.- Return generated subset size.

3.2.II Formula generation

The software mus can generate formula instances using an attribute set given.
An R formula can have a high complexity, in the current version only addtion an deletion formula types will be implemented.

Simple formula

This function must generate instances of formula using a given attribute set. This formulas only will have addition factors. The necessary parameters to do this activity are:

- **Main factor:** main attribute of the formula.

- **Attributes:** attribute set.

Functios must return the following values:

- *Generated formula.*
- ERROR IN MAIN FACTOR PARAMETER.
- ERROR IN ATTRIBUTES PARAMETER.

Function use case will be:

1.- If main factor paremeter is null: abort process and throw ERROR IN MAIN FACTOR PARAMETER signal.
2.- If main factor isn't text: abort process and throw ERROR IN MAIN FACTOR PARAMETER signal.
3.- If attributes parameter is null: abort process and throw ERROR IN ATTRIBUTES PARAMETER.
4.- If attributes aprameter isn't a string vector: abort process and throw ERROR IN ATTRIBUTES PARAMETER.
5.- Generate string with main factor and attributes using the correct mathematical sign.
6.- Parse string to formula.
7.- Return generated formula.

Complex formula

This function must generate instances of formula using a given attribute set. This formulas only will have addition and sustraction factors. The necessary parameters to do this activity are:

- **Main factor:** main attribute of the formula.
- **Attributes:** attribute set.
- **Signs:** signs set used to do a addition or sustraction factor.

Functios must return the following values:

- *Generated formula.*
- ERROR IN MAIN FACTOR PARAMETER.
- ERROR IN ATTRIBUTES PARAMETER.
- ERROR IN SIGNS PARAMETER

Function use case will be:

1.- If main factor paremeter is null: abort process and throw ERROR IN MAIN FACTOR PARAMETER signal.
2.- If main factor isn't text: abort process and throw ERROR IN MAIN FACTOR PARAMETER signal.
3.- If attributes parameter is null: abort process and throw ERROR IN ATTRIBUTES PARAMETER.
4.- If attributes aprameter isn't a string vector: abort process and throw ERROR IN ATTRIBUTES PARAMETER.
5.- If signs parameter is null: abort process and throw ERROR IN SIGNS PARAMETER signal.
6.- If signs parameter isn't an integer array: abort process and throw ERROR IN SIGNS PARAMETER signal.

7.- Generate string with main factor and attributes using the correct mathematical sign.
8.- Parse string to formula.
9.- Return generated formula.

3.3 DNN handling

This section includes the DNN handling functionalities. The functional requirements related to these funcionalities are:

1.- Generate a Deep Neural Network using a formula.

To do this, the function must train a DNN model using a given formula and training data set using the bigger relevant parameters set for DNN learning process configuration. Also must evalueate predicitons done over the training data set and a testing data set if it's possible returning these evaluations.
The function must follow this model:

Fit Deep Neural Network

The necessary parameters to do this activity are:

- **Formula:** formula used to generate the DNN model.
- **Data:** training data set.
- **Testing data:** testing data set. (OPTIONAL)
- **Activation:** activation function that will be used in DNNs training. Default: *Tanh*. Possible values: "Tanh", "TanhWithDropout", "Rectifier", "RectifierWithDropout", "Maxout" or "MaxoutWithDropout". (OPTIONAL)
- **Neurons:** hidden neural layers that will be used on DNNs.
- **Iterations:** iterations that will be done over training data set on DNN train process. Default: 100. (OPTIONAL)
- **Seed:** seed used to start the DNN training process. (OPTIONAL)
- **Rho:** factor of learning ratio adaptation on time. Used on DNN training. (OPTIONAL)
- **Epsilon:** adaptative learning ratio used on DNNs training process. (OPTIONAL)
- **Threads:** number of CPU threads that will be used. Default: 2. (OPTIONAL)
- **AUC:** variable where AUC obtained on predictions over training data set. (OtCIONAL)
- **Testing AUC:** variable where AUC obtained on predictions over the testing data set. (OPTIONAL)

Functios must return the following values:

- *DNN generated model.*
- ERROR IN FORMULA PARAMETER.
- ERROR IN DATA PARAMETER.
- ERROR IN TEST PARAMETER.
- ERROR IN ACTIVATION PARAMETER.
- ERROR IN NEURONS PARAMETER.

- ERROR IN ITERATIONS PARAMETER.
- ERROR IN SEED PARAMETER.
- ERROR IN RHO PARAMETER.
- ERROR IN EPSILON PARAMETER.
- ERROR IN THREADS PARAMETER.
- ERROR GENERATING MODEL.

Function use case will be:

1.- If formula parameter isn't a formula instance: abort process and throw ERROR IN FORMULA PARAMETER signal.

2.- If formula parameter doesn't contains the main factor: abort process and throw ERROR IN FORMULA PARAMETER signal.

3.- If data parameter is null: abort process and throe ERRO IN DATA PARAMETER signal.

4.- If main factor isn't on training data set: abort process and throe ERRO IN DATA PARAMETER signal.

5.- If formula attributes aren't on training data set: abort process and throe ERRO IN DATA PARAMETER signal.

6.- If testing parameter is null: abort process and throw ERROR IN TEST PARAMETER signal.

7.- If main factor isn't on testing data set: abort process and throw ERROR IN TEST PARAMETER signal.

8.- If formula attributes aren't on testing data set: abort process and throw ERROR IN TEST PARAMETER signal.

9.- If neurons parameter is less than one or null: abort process and throw ERROR IN NEURONS PARAMETER signal.

10.- If iterations parameter is less than one: abort process and throw ERROR IN ITERATIONS PARAMETER signal.

11.- If seed parameter is negative: abort process and throw ERROR IN SEED PARAMETER signal.

12.- If rho parameter is negative: abort process and throw ERROR IN RHO PARAMETER signal.

13.- If epsilon parameter is negative: abort process and throw ERROR IN EPSILON PARAMETER signal.

14.- If threads parameter is lees than 1: abort process and throw ERROR IN THREADS PARAMETER signal.

15.- Generate model using the training data set.

  15.1.- If any exception is thrown: abort process and throw ERROR GENERATING MODEL signal.

16.- Evaluate model making predictions over training set.

17.- Store AUC obtained on store variable given.

18.- If testing and testing AUC parameters are not null:

  18.1.- Evaluate model making predictions over testing set.

  18.2.- Store testing AUC obtained on testing AUC store variable given.

19.- Return generated model.

4 - Implementation

This chapter includes the software implementation based on the logical design described in the previous chapter. The full implementatios have been done using R language and grouping functions in files by functionality.
The implementation order is the same used on design chapter unless for functions with dependencies of other functions.
The order will be:

- Data management:
  - Inputs.
  - Data division.
- Variable handling:
  - Formula generation.
- DNN handling:
  - Fit Deep Neural Network.
- Variable handling (2):
  - Variable filtering (Dependencies: formula generation and DNN handling).

Remark: it's important to know that all functions have the package dependecy of *R.oo*.

4.1 Data management

The name of the file used to store the data mangement implementation is *funcs_data_management.R* and will includes the following functions:

4.1.I Inputs

Data loading

To implement data loading function, the designed interface will be implemented. The error signals will be implemented as exceptions that will be thrown and the FINALIZATION WITHOUT ERRORS signal will be just implemented as a normal function without return anything.
The function name will be *read.dataset.table* and the implementation is:

```
1 read.dataset.table <- function(dir = "",
2                                file,
3                                store = NULL,
```

```
                                dir.sep = "/",
                                file.sep = ","){
  # CHECK ATTRIBUTES
  if(is.null(dir)){
    throw('DIR is not specified (NULL)')
  }

  if(is.null(file)){
    throw('FILE is not specified (NULL)')
  }

  # VARIABLES
  data.table <- ""
  data <- ""

  # READ AND PARSE
  data.table <- read.table(file=paste(dir,file,sep=dir.sep),sep=file.sep)
  data <- data.frame(data.table) # Store data

  # FREE UNNECESSARY SPACE
  rm(data.table)

  # STORE
  eval.parent(substitute(store <- data))
}
```

Attributes loading

To implement attributes loading function, the designed interface will be implemented. The error signals will be implemented as exceptions that will be thrown and the FINALIZATION WITHOUT ERRORS signal will be just implemented as a normal function without return anything.
The function name will be *read.dataset.attr* and the implementation is:

```
read.dataset.attr <- function(dir = "",
                              file,
                              store = NULL,
                              column = TRUE,
                              dir.sep = "/",
                              file.sep = ","){
  # CHECK ATTRIBUTES
  if(is.null(dir)){ # Check directory
    throw('DIR is not specified (NULL)')
  }

  if(is.null(file)){ # Check file
    throw('FILE is not specified (NULL)')
  }

  if(!is.logical(column)){ # Check column
    throw('COLUMN isn\'t a boolean value')
  }

  # VARIABLES
  attr.table <- ""
  attr <- ""

  # READ AND PARSE
  attr.table <- read.table(file=paste(dir,file,sep=dir.sep),sep=file.sep)
  if(column)
    attr <- rownames(attr.table) # Store attribute names
  else
```

```
    attr <- colnames(attr.table) # Store attribute names

  # FREE UNNECESSARY SPACE
  rm(attr.table)

  # STORE
  eval.parent(substitute(store <- attr))
}
```

Data set loading

To implement data sets loading function, the designed interface will be implemented. The error signals will be implemented as exceptions that will be thrown and the FINALIZATION WITHOUT ERRORS signal will be just implemented as a normal function without return anything.
The function name will be *read.dataset* and the implementation is:

```
read.dataset <- function(dir.data,
                         file.test,
                         file.train,
                         file.attr = NULL,
                         store.test = NULL,
                         store.train = NULL,
                         store.attr = NULL,
                         dir.sep = "/",
                         file.test.sep = ",",
                         file.train.sep = ",",
                         file.attr.sep = ":",
                         file.attr.column = TRUE){

  # CHECK ATTRIBUTES
  if(is.null(dir)){ # Check directory
    throw('DIR is not specified (NULL)')
  }

  if(is.null(file.test) | is.null(file.train)){
    throw('TEST or TRAINING files are not correct files path')
  }

  if(!is.logical(file.attr.column)){ # Check column
    throw('COLUMN isn\'t a boolean value')
  }

  # VARIABLES
  data.test <- ""
  data.train <- ""
  data.attr <- ""


  # READ FILES
  # Training file
  read.dataset.table(dir=dir.data,
                     file=file.train,
                     store=data.train,
                     dir.sep=dir.sep,
                     file.sep=file.train.sep)


  # Test file
  read.dataset.table(dir=dir.data,
                     file=file.test,
                     store=data.test,
```

```
                      dir.sep=dir.sep,
                      file.sep=file.test.sep)

  if(!is.null(file.attr)){
    read.dataset.attr(dir=dir.data,
                      file=file.attr,
                      store=data.attr,
                      dir.sep=dir.sep,
                      file.sep=file.attr.sep,
                      column=file.attr.column)

    names(data.train) <- data.attr # Give attribute names
    names(data.test) <- data.attr # Give attribute names
  }


  # Set read values
  eval.parent(substitute(store.test <- data.test))

  eval.parent(substitute(store.train <- data.train))

  if(!is.null(data.attr))
    eval.parent(substitute(store.attr <- data.attr))
}
```

4.1.II Data division

KFold division

To implement kfold division function, the designed interface will be implemented. The error signals will be implemented as exceptions that will be thrown and returning the subsets vector generated in case the program end without errors.
The function name will be *kfold* and the implementation is:

```
kfold <- function(data,
                  folds,
                  k = 10){
  # CHECK ATTRIBUTES
  if(is.null(data)){
    throw('Data is null')
  }

  if(k<=0){
    throw('K is negative or zero')
  }

  if(is.null(folds)){
    throw('Store variable given is null')
  }

  # GENERATE PARTITIONS
  generatedFolds <- createFolds(data, k=k)
  eval.parent(substitute(folds <- generatedFolds))
}
```

This functios has a package dependecy: **caret**.

4.2 Variable handling

The name of the file used to store the variable handling implementation is *funcs_variable_handling.R* and will includes the following functions:

4.2.I Formula generation

Simple formula

To implement simple formula generation function, the designed interface will be implemented. The error signals will be implemented as exceptions that will be thrown and returning the formula generated in case the program end without errors.
The function name will be *variable.formula.generator* and the implementation is:

```
variable.formula.generator <- function(main,attr){
  # CHECK ARGUMENTS
  if(is.null(main)){
    throw('MAIN is not specified (NULL)')
  }

  if(!is.character(main)){
    throw('MAIN is not character type')
  }

  if(is.null(attr)){
    throw('ATTR set is not specified (NULL)')
  }

  # GENERATE FORMULA
  formula <- paste(main," ~ ",paste(attr,collapse = "+"),sep = "")

  # END
  return(as.formula(formula))
}
```

Complex formula

To implement complex formula generation function, the designed interface will be implemented. The error signals will be implemented as exceptions that will be thrown and returning the formula generated in case the program end without errors.
The function name will be *variable.formula.generator.complex* and the implementation is:

```
variable.formula.generator.complex <- function(main,attr,signs){
  # CHECK ARGUMENTS
  if(is.null(main)){
    throw('MAIN is not specified (NULL)')
  }

  if(!is.character(main)){
    throw('MAIN is not character type')
  }

  if(is.null(attr)){
    throw('ATTR set is not specified (NULL)')
  }

```

```
  if(is.null(signs)){
    throw('SIGNS set is not specified (NULL)')
  }

  if(length(attr) != length(signs)){
    throw('ATTR set and SIGNS set have different dimensions'))
  }

  # GENERATE FORMULA
  formula <- paste(main," ~ ",sep = "")
  formula.attr <- ""
  for(i in 1:length(attr)){ # Add attributes
    if(signs[i]>=0)
      formula.attr <- paste(formula.attr,attr[i],sep="+")
    else
      formula.attr <- paste(formula.attr,attr[i],sep="-")
  }
  formula.attr <- substr(formula.attr,2,length(formula.attr)) # Delete first simbol

  formula <- paste(formula,formula.attr,sep="") # End formula

  # END
  return(as.formula(formula))
}
```

4.3 DNN handling

The name of the file used to store the DNNs handling implementation is *funcs_dnn.R* and will includes the following functions:

Fit Deep Neural Network

To implement Fit DNN process function, the designed interface will be implemented. The error signals will be implemented as exceptions that will be thrown and returning the DNN model generated in case the program end without errors.
For implementations needs and H20 (used package) requirements, a new parameter will be added to the interface. This parameter is *Connect* that indicates if H2O server connection is needed or not.
The function name will be *dnn* and the implementation is:

```
dnn <- function(formula,
                data,
                data.test = NULL,
                activation = c("Tanh","TanhWithDropout","Rectifier","RectifierWithDropout
                    ","Maxout","MaxoutWithDropout"),
                hidden, # three layers of 50 nodes
                epochs = 100, # max. no. of epochs
                seed = 1200,
                rho = 1,
                epsilon = 1.0e-10,
                h2o.threads = NULL,
                store.auc.train = NULL,
                store.auc.test = NULL,
                connect = TRUE){
  # CHECK ARGUMENTS
  strF <- as.character(formula)
  if(is.na(match('~',strF)) | match('~',strF)!= 1){
    throw("Wrong formula expression.")
```

```
  }else
    attr.main <- strF[2]

  if(is.null(data))
    throw("DATA attribute is NULL")

  names <- names(data)
  if(length(names[names %in% attr.main])==0)
    throw(paste("Main attribute of formula isn't a data attribute. (",attr.main,")"))

  if(!is.null(data.test)){
    names <- names(data.test)
    if(length(names[names %in% attr.main])==0)
      throw("Main attribute of formula isn't a test data attribute.")
  }

  if(!is.null(activation)){
    activation <- "Tanh"
  }else if(length(activation) > 1){
    activation <- "Tanh"
  }

  if(epochs <= 0)
    throw("Iterations can't be less than 1")

  if(seed < 0)
    throw("Seed can't be negative")

  if(rho < 0)
    throw("Rho can't be negative")

  if(epsilon < 0)
    throw("Epislon can't be negatives")

  if(!is.null(h2o.threads)){
    if(h2o.threads < 1)
      throw("Can't use an H2O service with less than 1 thread of your CPU")
  }

  # VARIABLES
  attr.set <- all.vars(formula) # Take all variables
  attr.set <- attr.set[! attr.set %in% attr.main] # Remove main attr

  # Initialize h2O environment
  if(connect){
    if(is.null(h2o.threads))
      localH2O <- h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
    else
      localH2O <- h2o.init(ip = "localhost", port = 54321, startH2O = TRUE,nthreads = h2o
          .threads)
  }

  # Transform dataset
  data.h2o <- as.h2o(data)

  # Generate model
  model <- h2o.deeplearning(x = attr.set,  # Predictors
                            y = attr.main,   # Out attr
                            training_frame = data.h2o, # Training data
                            activation = activation,
                            hidden = hidden, # three layers of 50 nodes
                            epochs = epochs, # max. no. of epochs
                            seed = seed,
                            rho = rho,
                            epsilon = epsilon)

```

```
  # EVALUATE
  dnn.perf <- h2o.performance(model,data.h2o)
  DNN.AUC.tr <- h2o.auc(dnn.perf)
  eval.parent(substitute(store.auc.train <- DNN.AUC.tr)) # Store

  if(!is.null(data.test)){
    data.test.h2o <- as.h2o(data.test)
    dnn.perf.test <- h2o.performance(model, data.test.h2o)
    DNN.AUC.test <- h2o.auc(dnn.perf.test)
    eval.parent(substitute(store.auc.test <- DNN.AUC.test))
  }

  if(connect)
    h2o.shutdown(prompt = FALSE) # Close H2O environment

  return(model)
}
```

4.4 Variable handling (2)

Not implemented functions, because dependecies, of variable handling are now implemented. The file to store the implementation continue being *funcs_variable_handling.R*.

Variable filtering

To implement Fit DNN process function, the designed interface will be implemented. The error signals will be implemented as exceptions that will be thrown and returning the attributes subset filtered size in case the program end without errors.
The function name will be *variable.selector* and the implementation is:

```
variable.selector <- function(data,
                              variables = names(data),
                              main.class,
                              extra.eval = c("Sensible","Specific"),
                              filter.method = c("ChiSquared"),
                              max.size,
                              min.size = 0,
                              group = c("BestMin","BestMax"),
                              store,
                              dnn.store = NULL,
                              AUC.store = NULL,
                              AUC.test.store = NULL,
                              collapse = 0.3,
                              correlation.threshold = 0.5,
                              extra.min = 0.8,
                              testingset = NULL,
                              activation = c("Tanh",, "TanhWithDropout","Rectifier", "
                                  RectifierWithDropout", "Maxout", "MaxoutWithDropout"),
                              hidden, # three layers of 50 nodes
                              epochs = 100, # max. no. of epochs
                              seed = 1200,
                              rho = 1,
                              epsilon = 1.0e-10,
                              h2o.threads=NULL){
  ## CHECK ARGUMENTS
  if(is.null(data)){
    throw("Dataset given is NULL.")
  }

```

```
  if(is.null(variables)){
    throw("Varaible names given is NULL.")
  }

  if(!is.null(extra.eval)){
    if((extra.eval)!=8 | (!("Sensible" %in% extra.eval)
                          & (!"Specific" %in% extra.eval))){
      throw("Extra evaluation method is not allowed.")
    }
  }

  if(max.size <= 0){
    throw("Max size must be a positive number")
  }

  if(min.size > max.size){
    throw("Minimum size can not be less than maximum size.")
  }

  if(is.null(group)){
    group <- "BestMax"
  }else if(length(group)>1){
    group <- "BestMax"
  }

  if(nchar(group)!=7){
    throw("Group slection is not allowed")
  }else if(!("BestMin" %in% group)&(!"BestMax" %in% group)){
    throw("Group selection is not allowed.")
  }

  if(is.null(activation)){
      activation <- "Tanh"
    }else if(length(activation)>1){
      activation <- "Tanh"
    }

  if(is.null(store)){
    throw("Store variable is NULL.")
  }


  # VARIABLES
  SubGroups <- length(variables)/max.size # Num of kfolds variable (for huge amount of
      variables)
  correlation.selection <- c()
  varFolds <- variables[!variables %in% main.class]
  finalSelection <- c()
  auxFormula <- ""

  # CORRELATION EVALUATION
  if(SubGroups > 1){
    kfold(varFolds,varFolds,SubGroups) # For huge dataset divide in subgroups
  }

  # FIRST SELECTION
  for(i in c(1:length(varFolds))){ # For each subgroup, calculate correlation
    subformula <- variable.formula.generator(main.class,variables[rapply(varFolds[i],c)])
    subweight <- chi.squared(subformula,data)
    correlation.selection <- c(correlation.selection,rownames(subweight)[which(subweight
        >= correlation.threshold)])
  }

  # FINAL STATISTICAL SELECTION
  auxFormula <- variable.formula.generator(main.class,correlation.selection)
  weights <- chi.squared(auxFormula,data)
```

```
93  correlation.selection <- unlist(weights[weights >= correlation.threshold])
94  names(correlation.selection) <- rownames(weights)[which(weights >= correlation.
        threshold)]
95
96  # SORT CURRENT SELECTION
97  correlation.selection <- sort(correlation.selection,decreasing=TRUE)
98
99  # CHECK SIZE CONSTRAINTS
100 if(length(correlation.selection) < min.size){
101   return(0)
102 }else if(length(correlation.selection > max.size)){
103   correlation.selection <- correlation.selection[c(1:max.size)]
104 }
105
106 BEST <- "" # Final selection
107 BEST.dnn <- ""
108 current.AUC.train <- ""
109 current.AUC.test <- ""
110
111 # Open H2O connection
112 if(is.null(h2o.threads))
113   localH2O <- h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
114 else
115   localH2O <- h2o.init(ip = "localhost", port = 54321, startH2O = TRUE,nthreads = h2o.
        threads)
116
117 # SELECT WANTED GROUP
118 if(group %in% "BestMin"){ # Best minimum group
119   # Select initial minimum set
120   if(min.size >= 1){
121     bestmin = correlation.selection[c(1:min.size)]
122   }else if(length(correlation.selection)>=10){
123     bestmin = correlation.selection[c(1:10)]
124   }else{
125     bestmin = correlation.selection[1]
126   }
127
128   # Generate DNN and calculate AUC (firsttime, initialize)
129   current.formula <- variable.formula.generator(main.class,names(bestmin))
130   current.dnn <- dnn(formula = current.formula,
131                      data = data,
132                      data.test = testingset,
133                      hidden = hidden,
134                      store.auc.test = current.AUC.test,
135                      store.auc.train = current.AUC.train,
136                      connect=FALSE)
137
138   current.best <- bestmin
139   new.AUC.train <- ""
140   new.AUC.test <- ""
141   new.dnn <- ""
142   # Start to search
143   while(length(current.best) < length(correlation.selection)){
144     if(length(current.best) >= max.size){ # Check size constraints
145       bestmin <- current.best
146       break;
147     }
148     current.best <- correlation.selection[c(1:(length(current.best)+1))] # Add new
          variable
149     current.formula <- variable.formula.generator(main.class,names(current.best)) #
          Generate formula
150     new.dnn <- dnn(formula = current.formula, # Generate DNN
151                    data = data,
152                    data.test = testingset,
153                    hidden = hidden,
154                    store.auc.test = new.AUC.test,
```

```
                        store.auc.train = new.AUC.train,
                        connect=FALSE)
      if(new.AUC.train - current.AUC.train < collapse){ # Collapsed
        bestmin <- current.best[c(1:(length(current.best)-1))]
        break;
      }else{ # Update values
        current.AUC.train <- new.AUC.train
        current.AUC.test <- new.AUC.test
        current.dnn <- new.dnn
      }
    }

    BEST <- bestmin # STORE
    BEST.dnn <- current.dnn
  }else{ # BestMax
    # Select initial minimum set
    if(max.size <= length(correlation.selection)){
      bestmax = correlation.selection[c(1:max.size)]
    }else{
      bestmax = correlation.selection
    }

    # Generate DNN and calculate AUC (firsttime, initialize)
    current.formula <- variable.formula.generator(main.class,names(bestmax))
    current.dnn <- dnn(formula = current.formula,
                       data = data,
                       data.test = testingset,
                       hidden = hidden,
                       store.auc.test = current.AUC.test,
                       store.auc.train = current.AUC.train,
                       connect=FALSE)

    current.best <- bestmax
    new.AUC.train <- ""
    new.AUC.test <- ""
    new.dnn <- ""
    # Start to search
    while(length(current.best) > min.size & length(current.best) > 1){
      current.best <- correlation.selection[c(1:(length(current.best)-1))] # Delete a
          variable variable
      current.formula <- variable.formula.generator(main.class,names(current.best)) #
          Generate formula
      new.dnn <- dnn(formula = current.formula, # Generate DNN
                     data = data,
                     data.test = testingset,
                     hidden = hidden,
                     store.auc.test = new.AUC.test,
                     store.auc.train = new.AUC.train,
                     connect=FALSE)
      if(new.AUC.train - current.AUC.train < collapse){ # Collapsed
        bestmin <- correlation.selection[c(1:(length(current.best)+1))]
        break;
      }else{ # Update values
        current.AUC.train <- new.AUC.train
        current.AUC.test <- new.AUC.test
        current.dnn <- new.dnn
      }
    }

    BEST <- bestmax # STORE
    BEST.dnn <- current.dnn
  }

  # STORE
  eval.parent(substitute(store <- BEST))
  if(!is.null(dnn.store)){
```

```
    eval.parent(substitute(dnn.store <- BEST.dnn))
  }
  if(!is.null(AUC.store)){
    eval.parent(substitute(AUC.store <- current.AUC.train))
  }

  if(!is.null(testingset) & !is.null(AUC.test.store)){
    eval.parent(substitute(AUC.test.store <- current.AUC.test))
  }

  # Close H2O connection
  h2o.shutdown(prompt = FALSE) # Close H2O environment

  # RETURN FINAL SELECTION VARIABLE
  return(length(BEST))
}
```

5 - Guided experiment

This chapter includes a guiged experiment to test the implemented tool. Also, at the end, eficiency of DDNs and ANNs will be compared.

5.1 The data set

In this experiment several individues have been studied using microarrays technology. These individues have been catalogated as normal or tumor classes. This patients (tumor class) have developed a tumor related to prostate cancer.
We will understand *tumor* instances as patients that have developed a prostate cancer tumor and we will understand *normal* instances as people that haven't developed a prostate cancer tumor. Not other information about normal individues is known.
The microarray has been designed to study a group of candidates gen to be related with prostate cancer tumoring. The total amount of genes studied are 12600.
The transcript gene levels read from microarray have been ponderated and stored on data sets with several files.
The data set was generated to be used in machine learning experiments, for that reason the files included are:

- **prostate_TumorVSNormal_test.data:** includes a data set that must be used to test models (gold pattern). Contains 34 patients instances:
  - **Normal:** 34.
  - **Tumor:** 9.
- **prostate_TumorVSNormal_train.data:** includes a data set that must be used to train models. Contains 102 patients instances:
  - **Normal:** 50.
  - **Tumor:** 52.
- **prostate_TumorVSNormal.names:** contains metadata about data set atributes. In this case, 12600 are transcripts levels and are of type *continuous*, there are one more that is the main class (tumor or normal) attribute, this is of type *factor*.

The first file will be used to evaluate the accuracy of generated predictors and models. The second one will be the main data set and will be used to filter, select, and train DNN models. The last one will be used to obtain the gene names and link it to the other two data sets.

5.2 Motivation y objectives

Prostate cancer is, nowadays, the most frequent cancer on males, except of skin cancer. Most of prostate cancer are glandular cancers (starts at mocusoa and other secreters cells) and commonly affect males with ages over 65 years.
The major prostate cancer diagnosis problem is that, usually, doen't show early symptoms and the evolution is slow. making harder the diagnosis and detection.
For that reason, in medicine, some known studies are used to classify the risk leves of patients. One of the main factors used to classify the prostate cancer risk (or other prostate alterations) is the Prostatic Specific Antigen (PSA) levels in blood.
PSA is a protein substance specific of prostate. The presence of PSA in blod is low for healthy patients but it encreases for prostate cancer patients or for other prostate alterations.
Besides, gene alteration have been observed for prostate cancer patients. Some examples are the lost of p53, amplification of MYC or lost of PTEN.
Our objectives ar focused on predict prostate cancer risk level using gene expression levels. Genes selected as candidates are those have shown alterations on prostate cancer patients.
Our target is identify and calculate relationships between these expression levels and tumor development risks. To do this, the tool implemented on this project will be used.
Besides, we will use the results obtained to generate DNN and ANN models and to compare their reliability on this kind of experiments.

5.3 Experiment execution

Our experiment workflow will be the following:

1.- Data sets loading.
2.- Filter&Select variables.
3.- Find best training parameters for train a DNN with our data set.
4.- Efficiency study of DNN in our experiment.
5.- Efficiency study of ANN in our experiment.

Before start, there are several variables that have been instantiated to avoid write big lines of code or to not repeat the same information several times. These variables are:

```
project.root.dir <- "~PERSONALPATH/tfg"
project.fun.dir <- "~PERSONALPATH/tfg/src/funcs/"
project.data.dir <- "~PERSONALPATH/tfg/data/ProstateCancer"
test.file <- "prostate_TumorVSNormal_test.data"
train.file <- "prostate_TumorVSNormal_train.data"
attr.file <- "prostate_TumorVSNormal.names"
```

To begin, our project path will be stablished as working directory on our R engine. Also, necessary packages will be loaded and a seed will be selected to permit experiment replications. To do this we will use these commands:

```
# Set directory
setwd(project.root.dir)
## Set a seed for experiment replications
```

```
set.seed(1200)

##IMPORT necessary libraries
library(caret) ## Friendly way to create boxes
library(FSelector) ## Var selection functions
library(h2o) ## Library for DNNs
library(R.oo) ## Oriented Objects
```

Now, implemented functions will be loaded. All functions files have *.R* extension, for that reason, load all of them is as easy as use this command:

```
## Load necessary external functions
sapply(list.files(pattern="[.]R$", path=project.fun.dir, full.names=TRUE), source)
```

With this, all functios are loaded and ready to be used.
The next step is instantiate and initialize necessary variables to load data sets. We have to do it because our functions overwrite this variables functions but, if they're not initilized, a null pointer exception is thrown.

```
# Declare data variables
test.data <- ""
train.data <- ""
attr.data <- ""
```

Now we can load our data set:

```
# Load data
read.dataset(dir.data = project.data.dir,
             file.test = test.file,
             file.train = train.file,
             file.attr =attr.file,
             store.test = test.data,
             store.train = train.data,
             store.attr = attr.data)
```

For our dataset there are a problem with gene names. Several gene names contains reserved characters or starts with a digit, for that reason a specific function has been implemented to swap this character for other characters allowed.
This function is called *variable.name.correcto* and have been implemented on file *funcs_variable_handling* to be used on other experiments. The implementatios is:

```
variable.names.corrector <- function(names){
  #VARIABLES
  names.aux <- ""
  # CORRECT TEXT
  names.aux <- gsub("-","__",names)
  names.aux <- gsub("/",".",names.aux)
  names.aux <- gsub(" ","",names.aux)
  # STORE
  eval.parent(substitute(names <- names.aux))
}
```

Now we can change special characters and add the prefix *GG_* to the gene names (not to Class name).
To do this we will launch the following set of commads:

```
# Correct posible errors
variable.names.corrector(attr.data)
for(i in 1:length(attr.data)) # Add GG_ to all genes
```

```
  if(attr.data[i] != "Class")
    attr.data[i] <- paste("GG_",attr.data[i],sep = "")

# Restore names
names(test.data) <- attr.data
names(train.data) <- attr.data
colnames(test.data) <- attr.data
colnames(train.data) <- attr.data
```

Until now, we have loaded the data sets and have overwritting special characters. If we want know more information about our dataset we can use the R function *summary()* but it's not recommendable because it will shown 12601 information column entries.
Now we can filter our attribute data set. To do this we ned instantiate and initialize some variables for the same reson than before.

```
# Result variables
var.selected.max <- ""
var.selected.min <- ""
AUC.train.max <- ""
AUC.train.min <- ""
AUC.test.max <- ""
AUC.test.min <- ""
```

The next step is call the selector function. To this experiment we will call to the selecto using the minimum set of parameters to observe the efficiency without adjust all parameters of DNN training. Also we will call the tool in his two modes (BestMin and BestMax). The commands to call selector in both modes are:

```
# FILTER AND SELECT
# Best max
variable.selector(data = train.data,
                  variables = attr.data,
                  main.class = "Class",
                  testingset = test.data,
                  extra.eval = NULL,
                  filter.method = "ChiSquared",
                  max.size = 200,
                  store = var.selected.max,
                  group = "BestMax",
                  AUC.store = AUC.train.max,
                  AUC.test.store = AUC.test.max)

# Best min
variable.selector(data = train.data,
                  variables = attr.data,
                  main.class = "Class",
                  testingset = test.data,
                  extra.eval = NULL,
                  filter.method = "ChiSquared",
                  max.size = 200,
                  store = var.selected.min,
                  group = "BestMin",
                  AUC.store = AUC.train.min,
                  AUC.test.store = AUC.test.min)
```

Results can be found on *Results* section, following this one.
As you can see, default values has been used in most of the configurable attributes letting H2O environment select hidden neurons layers size to our data set.
Finally, the minimum best gene list obtained is:

```
[1] "GG_37639_at"   "GG_32598_at"   "GG_38406_f_at" "GG_41288_at"   "GG_37720_at"   "GG_
    38634_at"   "GG_37366_at"
[8] "GG_40282_s_at" "GG_40856_at"   "GG_41468_at"   "GG_39031_at"
```

And the maximum best is:

```
[1] "GG_37639_at"   "GG_32598_at"   "GG_38406_f_at" "GG_41288_at"   "GG_37720_at"   "GG_
    38634_at"   "GG_37366_at"
[8] "GG_40282_s_at" "GG_40856_at"   "GG_41468_at"   "GG_39031_at"   "GG_556_s_at"   "GG_
    32243_g_at" "GG_31538_at"
[15] "GG_1767_s_at"  "GG_39315_at"   "GG_575_s_at"   "GG_36601_at"   "GG_36491_at"   "GG_
    38028_at"   "GG_39545_at"
[22] "GG_35702_at"   "GG_37068_at"   "GG_40436_g_at" "GG_40435_at"   "GG_36533_at"   "GG_
    34678_at"   "GG_33121_g_at"
[29] "GG_38044_at"   "GG_39939_at"   "GG_37251_s_at" "GG_31527_at"   "GG_36666_at"   "GG_
    31444_s_at" "GG_914_g_at"
[36] "GG_36589_at"   "GG_39756_g_at" "GG_37754_at"   "GG_41385_at"   "GG_39755_at"   "GG_
    35742_at"   "GG_34950_at"
[43] "GG_32206_at"   "GG_38042_at"   "GG_33198_at"   "GG_581_at"     "GG_34840_at"   "GG_
    36569_at"   "GG_216_at"
[50] "GG_33614_at"   "GG_33674_at"   "GG_41104_at"   "GG_39243_s_at" "GG_38291_at"   "GG_
    34820_at"   "GG_36495_at"
[57] "GG_34775_at"   "GG_38338_at"   "GG_33819_at"   "GG_769_s_at"   "GG_31568_at"   "GG_
    41755_at"   "GG_33668_at"
[64] "GG_2046_at"    "GG_32718_at"   "GG_39123_s_at" "GG_33396_at"   "GG_1664_at"    "GG_
    37005_at"   "GG_32076_at"
[71] "GG_31545_at"   "GG_35644_at"   "GG_34592_at"   "GG_38814_at"   "GG_36149_at"   "GG_
    38950_r_at" "GG_291_s_at"
[78] "GG_38827_at"   "GG_33412_at"   "GG_256_s_at"   "GG_37203_at"   "GG_36780_at"   "GG_
    35277_at"   "GG_40607_at"
[85] "GG_1740_g_at"  "GG_33904_at"   "GG_38642_at"   "GG_1676_s_at"  "GG_33820_g_at" "GG_
    40071_at"   "GG_38669_at"
[92] "GG_37716_at"   "GG_37000_at"   "GG_34407_at"   "GG_39054_at"   "GG_32695_at"   "GG_
    1736_at"    "GG_39341_at"
[99] "GG_36555_at"   "GG_38087_s_at" "GG_34369_at"   "GG_40567_at"   "GG_37406_at"   "GG_
    38057_at"   "GG_33362_at"
[106] "GG_40024_at"   "GG_37741_at"   "GG_36030_at"   "GG_34608_at"   "GG_41530_at"   "GG
    _37630_at"   "GG_38408_at"
[113] "GG_1513_at"    "GG_41504_s_at" "GG_38322_at"   "GG_32242_at"   "GG_40125_at"   "GG
    _33137_at"   "GG_41485_at"
[120] "GG_1276_g_at"  "GG_41178_at"   "GG_1521_at"    "GG_1897_at"    "GG_39830_at"   "GG
    _33408_at"   "GG_39366_at"
[127] "GG_37347_at"   "GG_34784_at"   "GG_41768_at"   "GG_37743_at"   "GG_39634_at"   "GG
    _40301_at"   "GG_41242_at"
[134] "GG_34791_at"   "GG_36624_at"   "GG_38051_at"   "GG_36587_at"   "GG_36786_at"   "GG
    _31907_at"   "GG_2041_i_at"
[141] "GG_38279_at"   "GG_36943_r_at" "GG_38429_at"   "GG_36095_at"   "GG_31385_at"   "GG
    _40060_r_at" "GG_1831_at"
[148] "GG_942_at"     "GG_39750_at"   "GG_37573_at"   "GG_38435_at"   "GG_35146_at"   "GG
    _38740_at"   "GG_32780_at"
[155] "GG_41013_at"   "GG_1980_s_at"  "GG_39551_at"   "GG_37929_at"   "GG_33355_at"   "GG
    _36918_at"   "GG_38780_at"
[162] "GG_38098_at"   "GG_496_s_at"   "GG_32435_at"   "GG_829_s_at"   "GG_34646_at"   "GG
    _36668_at"   "GG_254_at"
[169] "GG_41732_at"   "GG_37035_at"   "GG_33716_at"   "GG_32109_at"   "GG_37958_at"   "GG
    _37582_at"   "GG_36864_at"
[176] "GG_33405_at"   "GG_38033_at"   "GG_1257_s_at"  "GG_32315_at"   "GG_31583_at"   "GG
    _32123_at"   "GG_33328_at"
[183] "GG_35354_at"   "GG_39798_at"   "GG_38410_at"   "GG_31791_at"   "GG_1708_at"    "GG
    _38385_at"   "GG_33134_at"
[190] "GG_32667_at"   "GG_34304_s_at" "GG_41531_at"   "GG_34376_at"   "GG_38986_at"   "GG
    _32800_at"   "GG_41214_at"
[197] "GG_39070_at"   "GG_40063_at"   "GG_37708_r_at" "GG_32412_at"
```

Now, to compare ANN efficiency against DNNs, we will train a traditional artificial neural network.

To do this, we will use *nnet* package to generate ANN models and *AUC* package to calculate AUC of generated predictions. For that reason we have to load these packages:

```
# Import necessary packages
library(nnet) ## ANNs tools
library(AUC) ## AUC and ROC functions
```

Current step consists on generate a formula with the filtered attribute set and train the ANN model.
To do this we use *nnet* function from *nnet* loaded package. *Remark: several experiments have been performed to find the best configuration that is shown below.*

```
# Generate formula
best.formula.max <- variable.formula.generator("Class",var.selected.max)
best.formula.min <- variable.formula.generator("Class",var.selected.min)

# Fit ANN
nn.fit.min <- nnet(formula=best.formula.min,  # our model
                   data=train.data, # our training set
                   size=50, # number of hidden neurons
                   maxit=1000, # max number of iterances to try converge
                   decay=5e-8, # avoid overfitting (value: a little more than 0)
                   trace=FALSE) # don't print the process

nn.fit.max <- nnet(formula=best.formula.max,  # our model
                       data=train.data, # our training set
                       size=4, # number of hidden neurons
                       maxit=1000, # max number of iterances to try converge
                       decay=5e-4, # avoid overfitting (value: a little more than 0)
                       trace=FALSE) # don't print the process
```

In maximum set case we can boserve that neuron number is 4. That is because for higger values a *too high amount of weights* exception is thrown. That show the inability of the tool to handle big data sets.
Now we evaluate the AUC for predictions over training and testing data sets:

```
# Training set
nn.pred.min.train <- predict(nn.fit.min, train.data, type='raw')
nn.roc.min.train <- roc(as.numeric(nn.pred.min.train),train.data[,"Class"])
nn.auc.min.train <- auc(nn.roc.min.train)
nn.pred.max.train <- predict(nn.fit.max, train.data, type='raw')
nn.roc.max.train <- roc(as.numeric(nn.pred.max.train),train.data[,"Class"])
nn.auc.max.train <- auc(nn.roc.max.train)
# Testing set
nn.pred.min.test <- predict(nn.fit.min, test.data, type='raw')
nn.roc.min.test <- roc(as.numeric(nn.pred.min.test),train.data[,"Class"])
nn.auc.min.test <- auc(nn.roc.min.test)
nn.pred.max.test <- predict(nn.fit.max, test.data, type='raw')
nn.roc.max.test <- roc(as.numeric(nn.pred.max.test),train.data[,"Class"])
nn.auc.max.test <- auc(nn.roc.max.test)
```

Results obtained are:

- Grupo mínimo:
  - AUC training: 1
  - AUC testing: 0
- Grupo máximo:
  - AUC training: 1
  - AUC testing: 0

As you can see, in both cases we obtain an overfitting of the model and the predictors cannot classify correctly instances that not fit on training set.
This expose the low capacity of nnet ANNs model to adjust models on this kind of experiments (transcriptomic field) without perform an expensive study to adjust properly the ANN model. Unlike this, DNN models adjust offer excellent results without a exhaustive configuration and handle easily the attribute sets selected.
Conclusions and analysis are of these results are in the section *Results*.

5.4 Resultados

The results obtained table for filter and selection process with the implemented tool have been:

Tipo	Min	Max	Devuelto	AUC.test	AUC.train	Tiempo
BestMax	0	200	200	0,9689	0,7948	120,86
BestMax	0	200	200	1	0,9517	114,72
BestMax	0	200	200	0,9889	0,9092	117,72
BestMax	0	200	200	0,9689	0,9383	119,32
BestMax	0	200	200	0.5556	0.8742	120.11
BestMax	0	200	200	0.9978	0.945	113.5
BestMax	0	200	200	1	0.9571	112.31
BestMax	0	200	200	0.9956	0.9254	115.7
BestMax	0	200	200	0.5556	0.7975	114.78
BestMax	0	200	200	0.5	0.8156	113.52
BestMax	0	200	200	1	0.9402	115.27
BestMax	0	200	200	1	0.9413	112.73
BestMax	0	200	200	0.9956	0.9273	114.27
BestMax	0	200	200	1	0.9706	117.11
BestMax	0	200	200	0.9867	0.8917	115.11
BestMax	0	200	200	0.9822	0.8846	118.47
BestMax	0	200	200	0.9956	0.9438	123.8
BestMax	0	200	200	1	0.9288	119.87
BestMax	0	200	200	0.8333	0.9053	130.35
BestMax	0	200	200	0.9289	0.9386	115.79
BestMax	0	200	200	1	0.9242	116.22
BestMax	0	200	200	0.56	0.9467	116.82
BestMax	0	200	200	0.5	0.8246	116.87
BestMax	0	200	200	1	0.9231	121.25
BestMax	0	200	200	1	0.9096	119.69
BestMax	0	200	200	1	0.8901	112.67
BestMin	0	200	10	0.9444	0.9681	133.42
BestMin	0	200	10	0.8844	0.9804	113.87

BestMin	0	200	10	0.9911	0.9638	118.19
BestMin	0	200	10	0.8711	0.9685	126.25
BestMin	0	200	10	0.9911	0.9508	109.11
BestMin	0	200	11	0.96	0.9427	165.3
BestMin	0	200	10	0.9733	0.9563	115.3
BestMin	0	200	10	1	0.971	114.42
BestMin	0	200	10	0.9378	0.9479	127.49
BestMin	0	200	10	0.9956	0.9585	116.78
BestMin	0	200	10	0.9822	0.9685	122.98
BestMin	0	200	10	0.9911	0.9565	116.82
BestMin	0	200	10	0.9422	0.9662	118.17
BestMin	0	200	10	0.9956	0.9681	122.23
BestMin	0	200	10	0.9244	0.9746	114.36
BestMin	0	200	10	0.9867	0.9402	113.1
BestMin	0	200	10	1	0.9765	114.42
BestMin	0	200	10	0.9911	0.9577	123.22
BestMin	0	200	10	0.9867	0.9785	114.28
BestMin	0	200	10	0.9956	0.9585	126.38
BestMin	0	200	10	0.9956	0.9758	128.74
BestMin	0	200	10	0.8333	0.9831	124.58
BestMin	0	200	10	0.6111	0.9467	110.31
BestMin	0	200	10	1	0.9481	110.54
BestMin	0	200	10	0.9911	0.9731	109.81
BestMin	0	200	10	0.9822	0.9629	111.19
BestMin	50	200	50	0.9644	0.9769	111.5
BestMin	50	200	50	1	0.9852	121.75
BestMin	50	200	50	0.9667	0.9652	112.38
BestMin	50	200	50	1	0.9246	115.61
BestMin	50	200	50	1	0.9654	130.42
BestMin	50	200	50	0.9956	0.9759	113
BestMin	50	200	50	1	0.9802	118.1
BestMin	50	200	50	0.96	0.9738	112.08
BestMin	50	200	50	1	0.9654	116.08
BestMin	50	200	50	1	0.9823	124.58
BestMin	50	200	50	1	0.9869	137.26
BestMin	50	200	50	0.9778	0.9727	119.42
BestMin	50	200	50	1	0.9754	118.97
BestMin	50	200	50	0.9333	0.9181	118.08
BestMin	50	200	50	1	0.9669	110.72
BestMin	50	200	50	0.9956	0.9785	110.7
BestMin	50	200	50	0.8689	0.9594	120.99
BestMin	50	200	50	0.8689	0.94	117.12

BestMin	50	200	50	0.6111	0.9873	124.13
BestMin	50	200	50	1	0.8815	118.97
BestMin	50	200	50	0.9822	0.9673	119.61
BestMin	50	200	50	0.8888	0.9719	111.04
BestMin	50	200	50	0.9956	0.9665	114.68
BestMin	50	200	50	1	0.964	117.92
BestMin	50	200	50	1	0.9721	114.47
BestMin	50	200	50	0.9822	0.974	115.63
BestMin	50	200	50	1	0.9677	112.53

We have performed three experiments for obtain this results table:

- **Group 1:** mode BestMax and maximum size 200 without minimum.
- **Group 2:** mode BestMin and maximum size 200 without minimum.
- **Group 3:** mode BestMin and maximum size 200 with minimum size 50.

The results distribution obtained for each group and for training set predictions is shown in the following boxplot:

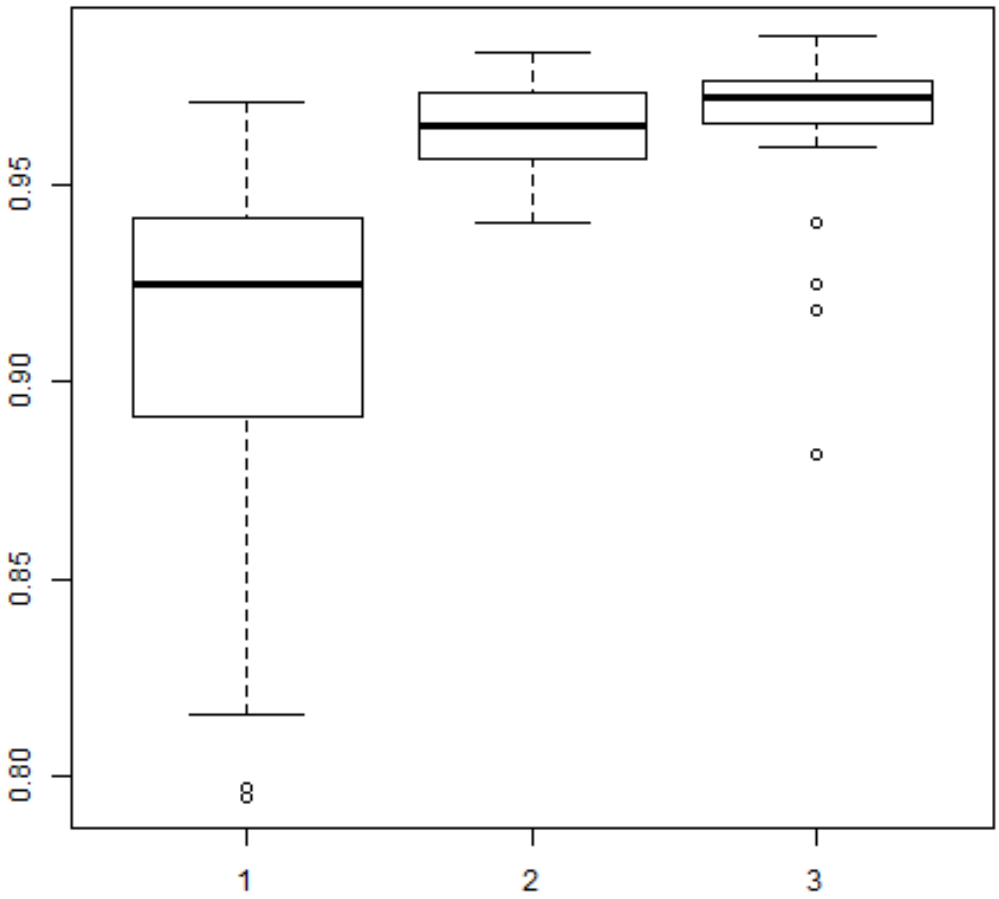


Figure 5.1: AUC Training

The boxplot shows a distribution with the most values of AUC above 0.9 for group 1 and above 0.95 for groups 2 and 3. This last one also have some outlayers values on below area.
These values are interesting but the really interesting results are the AUC for testing set, our gold pattern. These results distribution are:

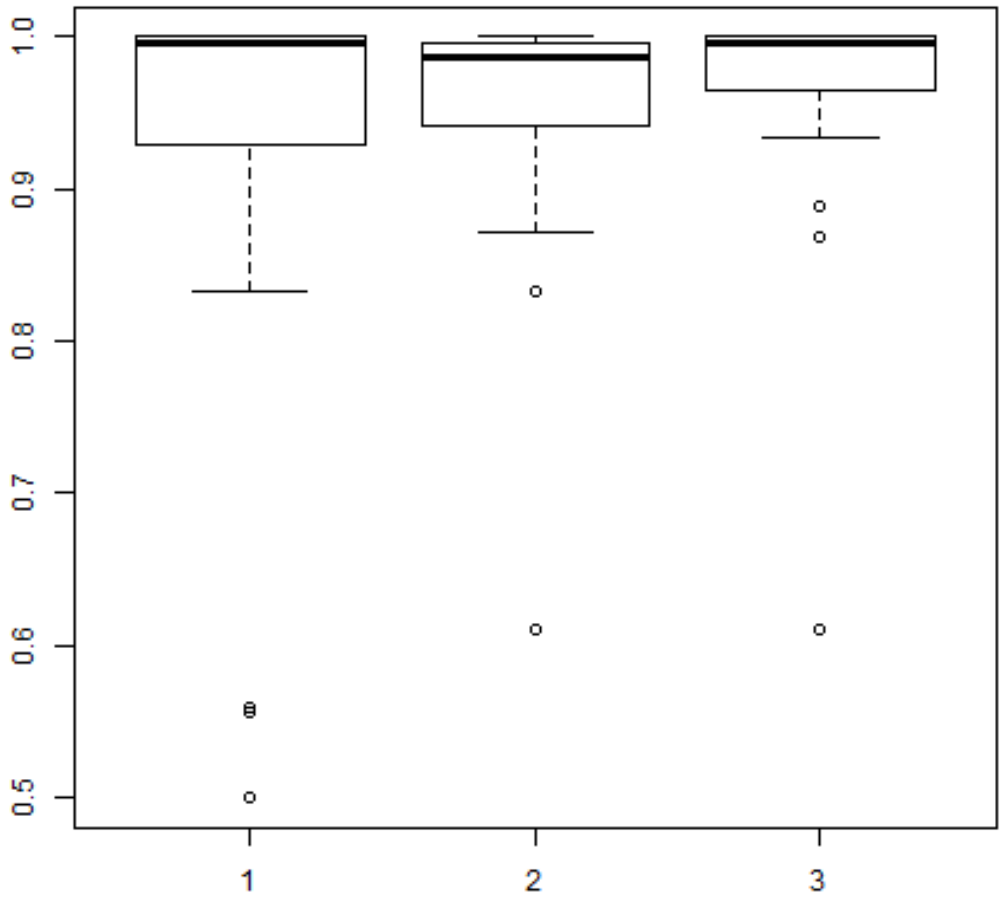


Figure 5.2: AUC Testing

We can observe that all groups have their values above 0.95. It means that our predictor can classify with a high accuracy the instances in our gold pattern.
Also we observe that the best group are not group 1 or 2, it's the third group which have the best distribution of results. It means that ask about the best maximum group or the best minimum group without the correct size constraints can return good results, but not the best ones and we must perform more experiment changing size constraints.
Finally, the time spent distributio to filter and generate the models and predictions is:

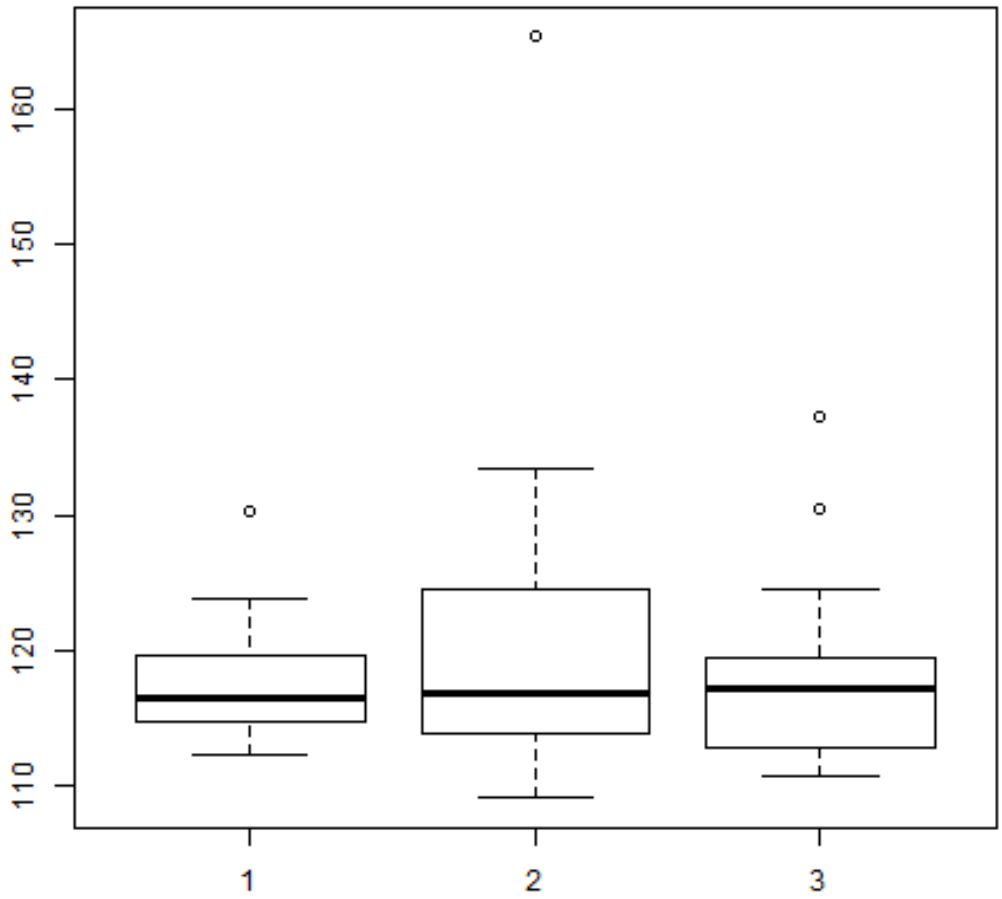


Figure 5.3: Time spent

We can se that for all groups, the time spent is around 2 minutes (120 secs) and rarely beats the 2 min 10 secs mark.
If we analyze the results obtained we can conclude that DNN aplied on this tool can offer excellent results in a reasonable time. Also can conclude that new DNN, and Deep Learning algorithms, can handle much better big dataset and more efficiently than traditional neural networks which couldn't handle correctly pur prostate cancer dataset.

6 - Project analysis and conclusions (EN)

To develope this project all normal software development project have been realized, some of them have been the problem study, design of a solution, logical design of software, implementation and testing of it. Furthemore, some research phases have been implented as results comparation with traditional technologies (ANN). In these named phases some bioinformatic important phases affecting parts as requirements selection, logical desing and some strategies of kind of data to handle to be useful for medics and biologists.
All of theses aspects will be studied one by one.

6.1 Review of requirements compliance

We will start checking the functional and non-functional requirements imposed at the beggining of the project.
The functional requirements were:

1.- Import data in different formats.
2.- Filter genetic variables.
3.- Generation of formulas using a set of variables.
4.- Divide a data set un random subsets (KFold strategy).
5.- Generate a Deep Neural Network using a formula.

All of functional requirements have been acomplished. Each one habe been implemented in one or several functions. The first and fourth requirements have been grouped by functionality and implemented on file *funcs_data_management.R*. The secon adn third requirements have been grouped too by functionaliti and have been implemented and stored on file *funcs_variable_handling.R*. Finally, the fifth requirement have been implemented in only one function and stored on file *funcs_dnn.R*.
Besides, during the guided experiment a new requirements, inherited from the R language, was identified. It was related to the reserved characters of the language. Some data set instances contains these reserved characters and must be swapped. For that reason a new function that substitute this special characters for others allowed was implemented and stored file *funcs_variable_handling.R* which contains the most related functionalities with this function functionality.
Now the non-functional requirements compliance will be studied. These requirements are:

1.- The software must can load microarrays data sets to expedite the tool integration (microarray is the most already implemented on clinics and hospitals).
2.- The software must be implemented on R language to take advantage of his high capacities on Big Data, Estatistiscs and Machine Learning.
3.- The software must handle exceptions generated to avoid non-controlated program abortion.
4.- The software must work on version 3.1.1 (R).
5.- The software must atomize the functionalities and implement it as function to subserve parallelization.
6.- The software must atomize the functionalities and implement it as function to subserve the code maitenance without affect the final user interface.
7.- The software must offer the higher amount of attributes to subserve the configuration of the pogram.

The first requirements is one of the most importants. If we have a llok to the current technologies used on sequencing and genetic research we can conclude that microarrays are one of the most studiend and expanded technologies. It's true that it's being eclipsed by the new RNAseq technology but we shouldn't forget that most part of operative biomedical and biologicar research facilities uses microarrays. For that reason, if we want a real applicability of the implemented tool we must handle the main type of data generated, it means, we must can handled numeric tables of attributes with continious variables and their related factors (diseases, malformations or other medical relevan factors).
The second and fourth requirements references the language and lower version that must be allowed by the tool. This decission was taken consciusly because the great capacity of R engine to handle huge amounts of information and for the big amount of already implemented tools offered by the community.
The third requirement have been implemented using the package *R.oo* which includes Oriented Object Programing aspects for R. in our case, the implemented tool handle exceptions and throw some exceptions when the functiones call have errors or missing attributes that wont let the tool work.
Fifth and sixth requirements references the strategies of code design to be used. The strategy used was atomize sll functionalities and implement them on different function to permit a future paralelization of the code and maintenance. This strategy can be observed on the logical design of the code.
Finally, the seventh requirement have been acomplished less in the DNN function. This was a consciusly decission, I decided reduce the amount of parameters to train a DNN because this algorithm, as the most of the new Deep Learning algoritmhs, has a huge amount of configurable paramaters that, at the end, don't significatly alter the results.

6.2 Review of objectives compliance

For this project some objectives have been imposed in base to the motivation of the project. This objectives were the guidelines to select the functional and non-functional requirements and, in consecuence, to the desing and implementation of the tool.
These objectives were:

- Implement a support software to identify genetics signatures using DNNs.
  - Filtering and identification of genetics variables.
  - Generate DNN models.
  - GeneraTe preodictions using DNN model.
- Research the reliability and effectiveness of DNN models generated to be used with genetic data.
- Research effectiveness differences between Artificial Neural Networks (ANNs) and DNNs to find genetic signatures.
- Interrelate different concepts learned during grade studies.
- Perform a project related with one of the specialties of the grade studied.
- Perform a project autonomously.

The first one was the main guideline of the functiona requirements. Has been acomplished and implemented in several functions that shape the tool.
The second objective has been acomplished with the guided experiment execution. The use of DNNs to analyze transcriptomic data have returned excellent results in acceptable times and with suitable resources consumptions. Their efficiency have been demonstrated, for the experiment performed, with the results obtained from predictions over our testing data set (gold pattern) identifying an attribute set higly related with the interest factor, in this case, the develop of prostate cancer tumors.
The third objective don't have been studiend in depth, understanding study in depth as the study of resources and time consumptions. This study have been evoided consciusly because the difficulties and computationals barriers observed in already implemented ANN tools that have been superseded by DNNs. This affirmation has been generated after see that these named ANN tools couldn't handled correctly a data set that, in omics sciences field, is small. DNNs didn't show this difficulties and computational barriers, for thath reasons DNNs are, definitely, a new potetntial tool in medicine and omics studies with huge data sets.
The last three objectives are the objectives associated to all final dissertations. The project has been selected an designed taking into account the studies cursed (Grade in Health Engeneering, speciality on Bioinformatics) and the knowledgement obtained during this, related to software, design, data mining, statistics, biology, algorithms and biotechnology. Furthemore, the project have been developed with the higgest autonomy by the author.

6.3 Enhancement opportunities

During the project development some concepts and functionalities non-implemented have been realized. These can offer a enhacement opportunities and must be taken in account. These are:

- **Specificity and sensibility evaluation implementation:** specificity and sensibility are concepts basics on medicine and must be taken in account if we want design a tool with applications in prognoses and diagnosis.
- **More statistical filtering methods:** currently there are several efficient statistical filtering and selection methods, but some less commons could be better in

specific situations, for that reason the amount of implemented filtering methods must be increased.
- **H20 exceptions handling:** it's unsual but, sometimes H2O environment can throw connection exceptions that kill the DNN training process. This rarely exceptions must be handled by the tool to avoid killing the process when connection fails.
- **H20 best configuration searcher:** H2O offers a random and grid search methods to find the best configuration for our DNN models. Coordinate this functionality with out tool could help to find better results or the best possible.
- **More complex formulas generator:** currently only addition and deletion formulas are generated by the tool. The improvement of the formula generator to allow more complex formulas could offer better results, but harder to understand.

6.4 Utilities and applicability

The main objective of this tool isn't generate research content, the main objective was and is apply this tool in a real medical envrionment. To favor this apllicability real economical conditions and technologies have been taken in account. Based on this, the following utilites can be performed with this tool:

- **Genetic signatures identification:** it's the main utility and the reason that impulses this project. This tool can be used to find relationships between genes and relevant medical factors increasing the current knowledgement about these factors.
- **Developing prognoses and diagnosis tools:** the knowledgement generated by this tool can promote the development of new diagnoses and prognosis tools that could accelerate the diseases identification and decrease the costs associated to prevention medicine.
- **Personalized medicine:** genetic signatures identification can improve the personalized medicine making easily the individual studies and helping to make strategic decission using population information.
- **Pharmacogenomic:** pharmacogenomic search accuracy of their drugs to alter specific genes or biological process. Identification of genetic signatures can offer information useful to design new drugs.
- **Autorregulatory gene networks enhancement:** autorregulatory gene networks is a fact and this tool, with this networks algorithms, can offer beter adjustments of these networks and could be applied on medicine, pharmacogenomic and in biological process monitoring.

For all these application we must take in account the bioethical implications and I suggest to study derived tools in a Bioethical Council.

6.5 Bioethical implications

The knowledgement in depth, althought it's an statistical value, about ourself, about our future or our offspring future, can generate personal conflicts of religious, moral,

existential, social or economical type. For this reason, it's important to highlight that conclusions derived by this project and their applications can provoke bioethical problems that must be resolved on specifics councils or advice groups once they arise.

6.6 Future lines of work

After projec analysis about requirements and objectives compliment and the study of the applicability of the impelemented tool, the project is considered ended having satisfied the bojectives imposed at the begining.
Now, as future objectives, we can think in several develoment lines, one of them have been already named on enhacemenet opprotunities section. Other development lines could be:

- Use of more and more diverse microarray data sets.
- Specificity and sensibilit evaluators implementation.
- Export functionalities module implementation.
- Include this tool in an R package.
- Implements a pipleine that search information in biological data bases using the calculated relationships.
- Study, on laboratory, of the results obtained with this tool.

7 - Análisis y conclusiones (ES)

Para este proyecto se han realizado todas las fases de proyecto normal de desarrollo de software, entre las que se encuentra el estudio del problema, diseño de la solución, diseño del software, implementación y testeo del mismo. Además se han realizado tareas referentes a la investigación con la comparación de los resultados con herramientas de la tecnología tradicional del área (ANN). En estas etapas nombradas también se han integrado aspectos y etapas referentes a un proyecto de bioinformática, viéndose afectada la parte de elección de requisitos y diseño lógico por factores de decisión biológicos o orientando los resultados de la herramienta a datos biológicos.
Todo esto lo vamos a analizar apartado por apartado.

7.1 Revision del cumplimiento de los requisitos

Comenzaremos revisando el cumplimiento de los requisitos funcionales y no funcionales impuestos al comienzo del proyecto. Los requisitos funcionales eran:

1.- Importación de datos en diferentes formatos.
2.- Filtrado de variables.
3.- Generación de fórmulas a partir de set de variables.
4.- División del set de datos en subconjuntos aleatorios (estrategia KFold).
5.- Generación de un modelo de Deep Neural Networks a partir de una fórmula.

Para el caso de los requisitos funcionales todos han sido cumplidos. Cada uno ha sido implementado en una o varias funciones. Las funciones que implementan el primer y cuarto requisito han sido codificadas en funciones y almacenadas en el fichero *funcs_data_management.R*.
El segundo y tercer requisito han sido implementados y almacenados en el fichero *funcs_variable_handling*
Por último, el quinto requisito ha sido implementado en una sóla función que se encuentra en el fichero *funcs_dnn.R*.
Además, durante la ejecución de la herramienta ya implementada se identificó un requisito propio del lenguaje utilizado, dicho requisito era derivado del impedimento de utilizar caracteres reservados en los nombres de las variables. Debido a que el set de datos utilizado para genes presentaba varios nombres de atributos que incluían dichos caracteres reservados, se implementó una función que sustituye dichos caracteres por otros no reservados. Ésta fue implementada y almacenada en el fichero *funcs_variable_handling.R* que agrupaba las funcionalidades más semejantes a la que esta función satisfacía.

Vistos los requisitos funcionales pasamos a revisar los requisitos no funcionales. Estos eran:

1.- El problema debe poder cargar sets de datos procedentes de microarrays (chips ADN) para favorecer su implantación (tecnología con mayor implantación en la actualidad).

2.- El programa debe implementarse en el lenguaje R para aprovechar su capacidad en áreas como Big Data, Estadística y Machine Learning.

3.- El programa debe gestionar las excepciones que se generen para evitar el aborto no controlado del workflow.

4.- El programa debe funcionar en cualquier versión de R igual o superior a la 3.3.1.

5.- El programa debe atomizar las funcionalidades e implementarlas como funciones para favorecer el paralelismo en futuras versiones.

6.- El programa debe atomizar las funcionalidades e implementarlas como funciones para favorecer la mantenibilidad del código sin afectar al usuario.

7.- El programa debe ofrecer funciones con la mayor cantidad de atributos posible para favorecer el absoluto control sobre la configuración del programa.

El primero de estos requisitos no funcionales es uno de los más importantes. Si se observa el estado actual de las tecnologías de secuenciación y estudios genéticos se puede concluir que los microarrays son una tecnología estudiada y expandida. Cierto es que está siendo eclipsada por la tecnología de RNAseq pero no hay que olvidar que la mayor parte de las instalaciones operativas en el medio en el que queremos incidir (Medicina) trabajan con la tecnología de microarrays. Es por esto que si queríamos que esta herramienta tuviera una aplicabilidad real en el entorno médico actual debía poder trabajar con los datos obtenidos de microarrays. Siguiendo este requisito la herramienta carga y modela utilizando datos de matrices numéricas (datos de microarrays) con variables de caracter contínuo (o acotado) que se relaciona a posteriori con variables de caracter finito (enfermedades, malformaciones u otros factores de interés médico).

Los requitos segundo y cuarto hacen referencia al lenguaje de implementación y versión mínima del mismo. Ambos se han cumplido al implementar la herramienta usando la versión de R 3.3.1. Esta decisión se tomó por el gran potencial que ofrece el motor de R para el manejo de grandes cantidades de información y por la cantidad de herramientas ya implementadas que ofrece la comunidad de R.

El tercero de los requisitos se ha implementado utilizando las funcionalidades del paquete para incluir características de programación Orientada a Objetos en R que es *R.oo*. En el caso de nuestra herramienta, se manejan excepciones y se lanzan excepciones propias para los casos en los que la llamada a la herramienta contiene errores o falta de argumentos que impiden el normal funcionamiento de la misma.

El quinto y sexto requisito hacen referencia a la estrategia de diseño del código a seguir, esta nos lleva a atomizar las funcionalidades implementándolas en funciones con el fin de favorecer su mantenibilidad y futura paralelización si conviene. Esta es la estrategia que se ha seguido y se puede ver reflejada en el diseño lógico del código.

Por último encontramos el séptimo requisito que busca favorecer el mayor control del usuario sobre la herramienta. Este requisito se ha cumplido a excepción de las funciones que manejan DNNs. Esto ha sido así por la decisión consciente de limitar el número de

parámetros de dichas funciones a un set de variables relevantes para el entrenamiento y configuración de una DNN ya que este algoritmo, como la gran mayoría de algoritmos pertenecientes a los conocidos como Deep Learning, tienen una cantidad muy grande de parámetros configurables, de los cuales no todos provocan cambios significativos en los resultados.

7.2 Revisión del cumplimiento de los objetivos

Para este proyecto se habían marcado unos objetivos en base a la motivación del proyecto. Dichos objetivos han guiado el estudio del problema, la selección de requisitos funcionales y no funcionales y, por ende, el diseño y la implementación de la herramienta.
Estos objetivos eran:

- Implementar un software de apoyo a la identificación de marcadores genéticos haciendo uso de DNNs.
  - Identificación y filtrado de variables genéticas.
  - Generación de modelos de DNNs.
  - Generación de predictores mediante el uso de modelos de DNNs.
- Estudiar la eficacia y fiabilidad de los modelos generados con DNNs para su uso con datos genéticos.
- Estudiar la diferencia de eficacia entre el uso de las Artificial Neural Networks (ANNs) y las Deep Neural Networks para la identificación de marcadores genéticos.
- Interelacionar los conocimientos adquiridos durante los estudios de grado.
- Realizar un proyecto relacionado con una de las menciones de los estudios del grado realizado.
- Realizar un proyecto de forma autónoma.

El primero de los objetivos es el que ha guiado los requisitos funcionales. Ha sido cumplido e implementado en diferentes funciones que conforman la herramienta en sí.
El segundo de los objetivos se ha cumplido con el experimento guiado realizado. El uso de DNNs para análisis de datos de transcriptómica ha dado resultados excelentes en tiempos razonables y con consumos de recursos adecuados. La eficacia queda demostrada, para el caso estudiado, con los resultados de predicción obtenidos para el set de testeo (patrón oro) consiguiendo identificar un set de variables fuertemente relacionadas con el factor de estudio, en este caso, la generación de tumores de cancer de próstata.
El tercero de los objetivos no ha sido estudiado en profundidad, entendiendo como estudio en profundidad un estudio del consumo de recursos y de tiempo para obtener un mismo resultado. Este estudio se ha evitado de forma consciente al observar las dificultades y barreras computacionales que padecían las ANNs ya implementadas en R que han sido desbancadas por las DNNs. Esta afirmación se genera al tener en cuenta que dichas herramientas no fueron capaces de manejar con soltura un set de datos que, dentro de las ciencias ómicas, es pequeño. Al no presentar estas dificultades y barreras el uso de DNNs las convierten, sin duda, en una nueva herramienta con gran potencial dentro del área de la medicina y los estudios de datos ómicos incluso a gran escala.
Los últimos tres objetivos son los referentes al propio objetivo de realizar un Trabajo Final de Grado. El trabajo se ha seleccionado y diseñado teniendo en cuenta los estudios

cursados (Grade in Health Engeneering, speciality on Bioinformatics) y los conocimientos adquiridos durante éstos interrelacionando conceptos de software, diseño, minería de datos, estadística, biología, algoritmia y biotecnología. Además el proyecto se ha realizado con la mayor autonomía posible por parte del alumno.

7.3 Oportunidades de mejora

Durante el desarrollo del proyecto se han observado algunos conceptos no implementados y algunas funcionalidades que ampliarían la calidad y utilidad de la herramientas. Estas cosas representan oportunidades de mejora de la herramieta y son:

- **Implementar evaluaciones de especificidad y sensibilidad:** los conceptos prueba sensible y específica son básicos en la medicina y deben ser tenido en cuenta si se desea diseñar una herramienta aplicable para la diagnosis o prognosis de pacientes.
- **Implementación de más métodos de filtración estadística:** en la actualidad existen varios métodos de filtrado y estudio de correlación entre variables. Aunque se implementen los más comunes y eficaces, es recomendable implementar otros que sean mejores para casos específicos que se puedan dar según el tipo de set de datos.
- **Gestión de excepciones de H2O:** aunque es inusual, el entorno de H2O puede lanzar excepciones de conexión que acaban con el proceso de entrenamiento de DNNs. Sería interesante implementar un proceso de gestión de excepciones que detecte cuál ha sido el error y lo subsane para no perder la iteración.
- **Implementación de la búsqueda del mejor de H2O:** el paquete H2O ofrece un sistema de búsqueda aleatoria y por grid del mejor conjutno de parámetros de configuración para un set de datos y su herramienta de entrenamiento de DNNs. Coordinar esta funcionalidad con nuestra herramienta permitiría reducir el conjunto de atributos obligatorios de las funciones de la herramienta y permitiría obtener el mejor resultado posible.
- **Generador de fórmulas más complejas:** actualmente solo formulas con adición y delección son generadas por la herramienta. La ampliación de las posibilidades en la generación de fórmulas podría ofrecer mejores resultados aunque más dificiles de entender.

7.4 Utilidades y aplicabilidad

Lo importante de esta herramienta no es sólo crear nuevo contenido de investigación sino tener aplicabilidad real en el entorno médico. Para favorecer esta aplicabilidad se han tenido en cuenta las condiciones reales y actuales de las tecnologías que se encuentras implementadas en los centros sanitarios. En base a esto se considera que las utilidades prácticas de esta herramienta serían:

- **Identificación de marcadores genéticos de factores de interés:** este es la utilidad principal para la que se ha diseñado la herramienta. Ésta puede servir para identificar la relación entre genes y factores de interés médico ampliando el conocimiento actual de dichos factores.

- **Creación de pruebas de diagnóstico y prognosis:** la generación de herramientas de diagnóstico y prognósis con el conocimiento derivado de la identificación de marcadores genéticos podría suponer una aceleración en la identificación de enfermedades además de un abaratamiento de los costes asociados a la medicina de prevención.
- **Medicina personalizada:** la identificación de marcadores genéticos serviría para mejorar la orientación de la medicina personalizada sirviendo tanto para mejorar el estudio de individuos concretos como para tomar decisiones estratégicas según la genética de la población según zonas geográficas.
- **Farmacogenómica:** la farmacogenómica busca la precisión en sus fármacos para afectar a genes concretos o para alterar los procesos que estos controlan. La identificación de marcadores genéticos ofrecería información de gran utilidad para el diseño de estos fármacos.
- **Ampliación del conocimiento de redes de autoregulación génica:** la autoregulación génica es un hecho. Esta herramienta, en conjunto con algoritmos de cálculo de redes de autoregulación génica, podrían ofrecer mejores ajustes de las mismas, las cuales son aplicadas en la medicina, la farmacogenómica y el control de procesos biológicos en general.

Para todas estas aplicaciones habría que tener en cuenta sus implicaciones bioéticas y se recomiendan que todas las herramientas derivadas de ésta sean estudiadas por un Consejo de Bioética.

7.5 Implicaciones bioéticas

El conocimiento en profundidad, aunque sea de forma probabilística, de nuestro propio ser, de nuestro futuro, del de nuestra descendencia, puede producir un conflicto personal de tipo religioso, moral, existencial, o a nivel económico o social. Por lo tanto, es de destacar que las conclusiones que se deriven de este trabajo y sus aplicaciones prácticas pueden conllevar problemas bioéticos que deberían resolverse en los respectivos comités dedicados al tema, una vez que fueran surgiendo.

7.6 Líneas futuras de trabajo

Después de la realización del proyecto y del análisis del cumplimiento de los requisitos y la aplicabilidad de la herramienta se considera por satisfechos los objetivos marcados para este proyecto. Como objetivos futuros se pueden marcar varios, entre los que se incluyen las nombradas en el apartado de oportunidades de mejora, la siguientes se consideran como interesantes y útiles para la mejora y desarrollo de esta herramienta:

- Estudio de más y más variados sets de datos.
- Implementación de evaluadores de sensibilidad y especificidad.
- Implementar módulo de exportación de datos en diferentes formatos.
- Crear paquete en R con la herramienta.

- Implementación de pipeline que busque información en bases de datos utilizando las relaciones calculadas por la herramienta.
- Estudio, en laboratorio húmedo, de los resultados obtenidos por esta herramienta.

8 - Concepts

8.1 Omics sciences

In biomedicine. omics are all disciplines, technologies and research areas that study a set or the whole biological system.
Term omics is a new-flanged sufix that is added to another concept to define a biological system, understandin as biological system a whole or a functional part of an organism. The most popular example and used is genomics that includes all knowledge areas and technologies that research about an individue or specie genome. Based on this, genomics includes all researches related with sequentation and genomes annotation, related with functional genome regulation, with the mutatuions and genetic modifications studies, etc. As you can appreciate, omics are important disciplines by their own, but also as source of new information tools that permit increas the knowledgement about more specific areas. Other omics are transcriptomic (genes transcripts study in an organism), proteomic (proteins presence study in a biological system) and metabolomic (molecular study of metabolics systems). To have a more complete set we must include conectomic, epigenomic, filogenomic and metagenomic. To know more you can see this web[1].

8.2 Deep Learning and Deep Neural Networks

For a long time, several algorithms have been developed in the machine learning field. Wit this algorithms we want teach (adjust parameters and models of algorithm) to a computer to take advantage of his computational capacity to detect patterns and make predictions using the identified patterns.
In recent years a new family of machine learning algortihms have been developed. These are characterized by their capacity to learn about the data representation. This family is known as Deep Learning (DL) algorithms. These algorithms are a result of increasing the complexity of themselves using non-linear multiple composed transformations architectures. Furthemore, the current boom of these technologies it's given by the relevant enhacement of results regarding its homologues of non-deep (traditional) machine learning algorithms.
An specific case of these DL algorithms is Deep Neural Networks (DNN) based on tra-

[1] https://en.wikipedia.org/wiki/Omics

ditional artificial neural networks (ANN) but using several hidden perceptrons layers between the input and output. Their main quality is their capacity to handle non-linear complex relationships.
Some examples of results obtained using DNNs are the ones obtained in Automatic Speech Recognition (ASR)[2] or in image identification[3]. One of the new fields where DNns it's being applied are in genomics studies with objective of identifying genetic patterns and signatures.

8.3 Cross-validation method and KFold strategy

Cross-validation in a technique used to evaluate results obtained from an statistical analysis and to guarantee the independence between the training and testing partition. It consists on iterate and calculate the arithmetic mean of de evaluations over diferent partitions. It's used in studies where the main objective is generate predictors and it's wanted to estimate the accuracy with a more reallistic data set. It's a popular technique in fields like Artificial Intelligence and Machine Learning.
In cross-validation over K iterations or k-cross-validation, data is divided in K subsets. One of the subsets will be used as testing set and the rest (K-1) as training set. We repeat this cross-validation process K iterations, using all possible divisions, at least, once as testing set. Finally, arithmetic mean of obtained values is calculated to obtain an unique accuracy result. This method is soo precise because it evaluates using K combinations of test and train datasets, but there'is a disadvantage, in contradistinction to retention method, it's too slow from the point of view of computational. In practice, the number of divisions dependes on the data set size.
For more information check this link[4].

8.4 Sensibility and specificity

Specificity and sensibility are conceps used on classifiers (or predictors) that return a binary discret statistitcal variable (commonly parsed to Correct classification or not).
Specially in Medicina the specificity and sensibility concepts are basics and must be taken in account in diagnosis tests. A description of these concepts, oriented on medicine, is:

- **Sensibility(S):** it's defined as the probability of obtain a positive result if the dissease is present. Sensible variables are ones which alter their values when there are any disease, but withoout being specific, it means, they alter they are affected by any problem or alteration. The formula of sensibility is:

$$S = \frac{TP}{TP + FN}$$

[2]Geoffrey Hinton, et.al. "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups". I EEE Signal Processing Magazine (Volume:29 , Issue: 6),Pages 8297,doi: 10.1109/MSP.2012.2205597

[3]Alex Krizhevsky, et.al."ImageNet Classification with Deep Convolutional Neural Network".

[4]https://en.wikipedia.org/wiki/Cross-validation_(statistics)

- **Specificity(E):** it's defined as the probability of obtain a negative result if the individue is healthy. Specific variables are ones which alter their values only when an specific patology is present, but not always they are affected, generating errors of False Negatives. The formula of specificity is:

$$E = \frac{TN}{TN + FP}$$

It's important to prioritize in a classifier if it's wanted as a specific or sensible classifier. Commonly, sensible tests are used to select a first group of candidates which will receive a specific test to have a reliable result. It's done in this way because decrease the error ratio and, also, commonly is cheaper than other strategies.

8.5 Variables correlation

In probabilistic and statistics, correlation is the value of the strength and direction of a linear and proportional relationship between two statistical variables. It's consider that two cuantitative variables have a relationship when the alteration of the values of one, systematically alter the homonyms values of the other one: if we have two variables (A and B), we can say that exists a relationship when by increasing A values, B values increase tu (direct relationship) or decrease (inverse relationship).
A relationship between two variables it's represented by the best fit line draw from a points set. The main components of a fit line and, in consecuence, of a relationship, are:

- **Strength:** It measures the degree to which the line represents the points set: if the poitns set is distributed in a narrow line the fit line will be a straight line and it means that there is a good strength. In other cases, if the representation of the points set is an elicoide or circular fit line, strength will be weak.
- **Direction:** study the variation of B values wregarding to A values: if increasing A values makes increase B values, the direction is *direct*; if decreasing A values makes increase B values, the direction is *inverse*.
- **Form:** stablished the best fit line: straight line, monotonic curve or not monotonic curve.

8.6 ROC curve and area under ROC curve (AUC)

In signal detections theory, a ROC curve (Receiver Operating Charasteristic) is a graphic representation of sensibilitiy versus specificity for a binary classfication system. Other way to understand it is: if there is a binary classificator possible values are correct classification or not from a more complex system, weel we will represent the true positives versus the true negatives ratio to generate the ROC curve.
The ROC analysis gives information about the accuracy of the model making possible evlauate and select an optimal option and discard suboptimal and independent models. The ROC analysis is related directly with the cost/benefits analysis to make diagnostics decissions.

ROC curve can be used to generate statistics that resume the effectivity of a classifier. Some of them are:

- Insert point of the ROC curve in the convex line to the line of discrimination.
- The area between ROC curve and the convex-parallel discrimination.
- Area under ROC curve (AUC).
- Sensibility index, distance between the mean distribution of systema activity under only signal conditions nd divided by typical desviation.

8.7 Microarrays

Microarray is a technology used to identify and study genetical sequences presence levels. Consists on a solid surface, commonly made in glass or plastic, where a grid of wells have been prebuilt.
In each well there are an spcific sequence probe and what it's measured is the hybridation levels between our probe and the sample. Usually, this hybridizing is measured using fluorescence. This microarray results are taken by image procressing using a computer because this well grids are microscopic.
There are several uses of this microarrays, one is the genetical presenece in a sample (genomics). Other one is the identification of RNA or proteins presence obtaining transcript levels (transcriptomics) or specific proteins levels (proteomics).
Some of the most famous companies about microarrays design, fabrication and analysis are: Affymetrix with their Affymetrix 417 Arrayer mycroarray and their own scanners. Also is known Cartesian Technologies with their PyxSys, ProSys and their own scanners. You can find more information (in spanish) in the document developed by Genoma España[5].

[5]Marta López, et.al. *Aplicaciones de los Microarrays y Biochips en Salud Humana*. Genoma España, Noviembre 2011. ISBN: 84-609-9226-8.

9 - Bibliography

Bibliography used to develop this projec have been the following:

- Geoffrey Hinton, et.al. *Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups*. IEEE Signal Processing Magazine (Volume:29 , Issue: 6),Pages 8297,doi: 10.1109/MSP.2012.2205597.
- Alex Krizhevsky, et.al. *ImageNet Classification with Deep Convolutional Neural Networks*.
- José M. Jerez Aragonés, et.al. *A combined neural network and decision trees model for prognosis of breast cancer relapse*. Artificial Intelligence in Medicine 27 (2003) 45–63.
- Dan Ciregan, et.al. *Multi-column deep neural networks for image classification*. Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on 16-21 June. doi 10.1109/CVPR.2012.6248110.
- Daniel Graupe. *Principles of Artificial Neural Networks*. World Scientific, Advanced Series in Circuits and Systems vol.7. ISBN 978-981-4522-73-1.
- Mateo Seregni, et.al. *Real-time tumor tracking with an artificial neural networks-based method: A feasibility study*. Acta Medica Iranica, 2013.
- Farin Soleimani, et.al. *Predicting Developmental Disorder in Infants Using an Artificial Neural Network*. Acta Medica Iranica51.6 (2013): 347-52.
- Hon-Yi Shi, et.al. *Comparison of Artificial Neural Network and Logistic Regression Models for Predicting In-Hospital Mortality after Primary Liver Cancer Surgery*. Found at: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0035781#references.
- Filippo Amato, et.al. *Artificial neural networks in medical diagnosis*. Journal of Applied Biomedicine, vol.11, issue 2, pages 47-58, 2013.
- Web: https://cienciasomicas.wordpress.com/. Checked: julio 2016.
- Web: https://en.wikipedia.org/wiki/Omics. Checked: julio 2016.
- Web: https://en.wikipedia.org/wiki/Cross-validation_(statistics). Checked: julio 2016.
- Web: https://es.wikipedia.org/wiki/Red_neural_artificial. Checked: junio 2016.
- Web: https://en.wikipedia.org/wiki/Artificial_neural_network. Checked: junio 2016.
- Web: http://neuralnetworksanddeeplearning.com/chap1.html. Checked: junio 2016.
- Web: http://deeplearning.net/. Checked: junio 2016.
- Web: http://scikit-learn.org/stable/modules/cross_validation.html. Checked: agosto 2016.
- Web: https://es.wikipedia.org/wiki/Validacion_cruzada. Checked: agosto 2016.

- Web: https://es.wikipedia.org/wiki/Correlacion. Checked: agosto 2016.
- Web: https://explorable.com/es/la-correlacion-estadistica. Checked: agosto 2016.
- Web: https://es.wikipedia.org/wiki/Sensibilidad_y_especificidad_(estadistica). Checked: septiembre 2016.
- Web: https://www.fisterra.com/mbe/investiga/pruebas_diagnosticas/pruebas_diagnosticas.asp. Checked: septiembre 2016.
- Web: https://es.wikipedia.org/wiki/Chip_de_ADN#Chips_de_oligonucle.C3.B3tidos_de_ADN Checked: septiembre 2016.
- Marta López, et.al. *Aplicaciones de los Microarrays y Biochips en Salud Humana*. Genoma España, Noviembre 2011. ISBN: 84-609-9226-8.

Acknowledgments

Quiero dedicar este trabajo a todas las personas que me han acompañado, ayudado y, sobretodo, soportado durante estos cuatro años de carrera. De todas y cada una de ellas he sacado buenos momentos y he aprendido algo, ya sea en la amistad o en la enemistad.
Es especial quiero dar las gracias a mis padres por todo el apoyo dado, a mis amigos (en especial al extinto trio maravilla) por hacer ameno cada momento amargo, a aquellos profesores que han conseguido transmitirme su pasión por sus áreas del conocimiento y a la comunidad de Stackoverflow que es la que realmente me ha enseñado a programar.
Por último agradecer a mi ejército de mosquitos montados en cucarachas voladoras el haber estado ahí en esa última etapa de agobio que parece que va a poner fin a estos estudios de grado.