

Geovanny Alexander Risco Carrera

**MACHINE LEARNING TO PREDICT PROTEIN FUNCTION THROUGH CO-
EVOLUTIONARY INFORMATION**

FINAL DEGREE PROJECT

Directed by Aïda Valls and Toni Gabaldón

Degree in Computer Engineering



UNIVERSITAT ROVIRA I VIRGILI

**Tarragona
2021**

Acknowledgements

This work was possible thanks to many people who have contributed somehow on its accomplishments.

I would like to start acknowledging the Institute for Research in Biomedicine (IRB) of Barcelona and the Barcelona Supercomputing Center (BSC) for letting me the opportunity to realize an internship stay in their centers. They provided me with the resources for making this project possible. I am deeply grateful for that.

My special thanks to Toni Gabaldon, my supervisor for the project, and his team. Toni proposed me the project last year, when the Covid-19 situation was still uncertain, but I accepted the challenge. I would like to express my gratitude to him, for relying on me for such big project and for supporting me despite his limited time, and to his team, for letting me acquire invaluable knowledge throughout the group meetings.

I would also like to thank Aïda Valls, my academic tutor at URV, for her valuable guidance throughout the writing of this thesis. Your suggestions and involvement in the project helped to bring this work to a higher level.

Finally, I wish to thank my family for always supporting me, in every decision and every step of my life. Despite the difficulties of being far from home, they encouraged me to keep going and as always, effort pays off. Thanks, of course, to Anna, for being my closest support, my way of inspiration and fun in this journey.

Abstract.

With the advances in the Next Generation Sequencing technologies, the number of identified gene sequences is growing exponentially. The function of most of these newly identified sequences remains unknown. Different bioinformatics approaches have been proposed to help experimental techniques to annotate protein function. Among them, Phylogenetic Profiling (PP) is a method that exploits the evolutionary co-occurrence pattern to identify functional related proteins. The identification of these phylogenetic “profiles” allows us to infer the function of uncharacterized proteins from others already annotated. However, the correlation of these patterns was normally performed by clustering methods. In this project, we propose combining a Phylogenetic Profiling analysis with the learning ability of Machine Learning methods in order to predict protein function in the form of GO terms. A pipeline was developed to test different models and filters in a High-Performance Computing environment. Among the six supervised classification algorithms evaluated, Random Forest method provided the most accurate predictions, with a mean F1-score of 0.61. The same data was used to predict potential human disease genes and the obtained results were very similar. Although this performance is still far for the needs of scientific community, this study validates Phylogenetic Profiling as a method that can be combined with other approaches to further improve the performance of protein function assignment.

Resum.

Amb els avanços de les tecnologies de *Next Generation Sequencing*, el nombre de seqüències gèniques identificades creix exponencialment. Es desconeix la funció de la majoria d'aquestes seqüències identificades recentment. S'han proposat diferents enfocaments bioinformàtics per ajudar les tècniques experimentals a anotar la funció proteica. Entre ells, el *Phylogenetic Profiling* (PP) és un mètode que analitza el patró evolutiu de co-ocurrència per identificar proteïnes relacionades amb la funcionalitat. La identificació d'aquests "perfils" filogenètics ens permet inferir la funció de les proteïnes no caracteritzades a partir d'altres ja anotades. Tanmateix, la correlació d'aquests patrons es realitza normalment mitjançant mètodes de *Clustering*. En aquest projecte, hem proposat combinar un anàlisi de perfils filogenètics amb la capacitat d'aprenentatge dels mètodes de *Machine Learning* per predir la funció de les proteïnes en forma de GO terms. S'ha desenvolupat un programa en *pipeline* per provar diferents models i filtres en un entorn *High-Performance Computing*. Entre els sis algoritmes de classificació supervisats avaluats, el mètode Random Forest ha proporcionat les prediccions més precises, amb un F1-score mitjà de 0,61. Les mateixes dades s'han utilitzat per predir possibles gens de malalties humanes i els resultats obtinguts han estat molt similars. Malgrat que aquest rendiment encara és lluny de les necessitats de la comunitat científica, aquest estudi valida el perfil filogenètic com un mètode que es pot combinar amb altres tècniques per millorar encara més el rendiment en l'assignació de funcions a proteïnes.

Resumen.

Dado el gran avance de las tecnologías de *Next Generation Sequencing*, el número de secuencias de genes identificadas está creciendo exponencialmente. La función de la mayoría de estas secuencias sigue siendo desconocida. Se han propuesto diferentes métodos bioinformáticos para ayudar a las técnicas experimentales a anotar la función de estas proteínas. Entre ellos, el *Phylogenetic Profiling* (PP) es un método que aprovecha el patrón de co-ocurrencia evolutivo para identificar proteínas relacionadas funcionalmente. La identificación de estos “perfiles” filogenéticos nos permite inferir la función de proteínas no caracterizadas a partir de otras ya anotadas. Sin embargo, la correlación de estos patrones se realiza normalmente mediante métodos de *Clustering*. En este proyecto, se propone combinar un análisis de *Phylogenetic Profiling* con la capacidad de aprendizaje de los métodos de *Machine Learning* para predecir la función de las proteínas en forma de GO terms. Se desarrolló una *pipeline* para probar diferentes modelos y filtros en un entorno de *High-Performance Computing* (HPC). De entre los seis algoritmos de clasificación evaluados, el método Random Forest obtuvo las predicciones más precisas, con un F1-score promedio de 0.61. Los mismos datos se utilizaron para predecir posibles genes de enfermedades humanas y los resultados obtenidos fueron muy similares. Aunque este rendimiento aún está lejos de las necesidades de la comunidad científica, este estudio valida el perfil filogenético como un método que se puede combinar con otros para mejorar la precisión de la asignación de funciones de proteínas.

Contents

CHAPTER 1. INTRODUCTION	1
1.1 BIOINFORMATICS	2
1.1.1 <i>Central dogma of molecular biology</i>	2
1.1.2 <i>Homology</i>	3
1.1.3 <i>Gene function</i>	4
1.1.4 <i>Gene Ontology</i>	4
1.2 MACHINE LEARNING.....	6
1.2.1 <i>Types of Machine Learning</i>	6
1.2.2 <i>Life cycle of a ML model</i>	7
1.2.3 <i>Evaluation metrics</i>	8
1.3 ABOUT THE CENTRE	10
1.4 OBJECTIVES.....	10
1.5 STRUCTURE OF THE DOCUMENT	11
CHAPTER 2. LEARNING METHOD SELECTION	12
2.1 INTRODUCTION	12
2.2 MATERIALS	12
2.2.1 <i>Tools</i>	12
2.2.2 <i>HPC environment: MareNostrum</i>	13
2.3 DESIGN	14
2.3.1 <i>Specifications</i>	14
2.3.2 <i>Activity Diagram</i>	14
2.4 METHODS	15
2.4.1 <i>Databases and data retrieval</i>	15
2.4.2 <i>Machine Learning methods</i>	16
2.4.3 <i>Classification based on Decision Trees</i>	18
2.5 IMPLEMENTATION	22
2.5.1 <i>Data preprocessing</i>	22
2.5.2 <i>Binary classification on Phylogenetic Profiling matrix</i>	24
2.5.3 <i>One-Vs-All classification on Phylogenetic Profiling matrix</i>	24
2.5.4 <i>Parallelized code</i>	25
2.6 RESULTS.....	26
2.6.1 <i>Classification of “mitochondrial localization” function</i>	26
2.6.2 <i>Classification of GO terms</i>	26
2.6.3 <i>Parameter optimization of RandomForest</i>	29
CHAPTER 3. CLINICAL APPLICATION	31
3.1 INTRODUCTION	31
3.2 METHODOLOGY	31
3.3 RESULTS.....	32
CHAPTER 4. DISCUSSION.....	34
4.1 CONCLUSIONS	34
4.1.1 <i>Automatic Function Prediction</i>	34
4.1.2 <i>Predictions for disease-related genes</i>	35
4.2 FUTURE IMPROVEMENTS.....	36
4.3 SELF-EVALUATION.....	36
REFERENCES.....	38
ANNEX A: DISPONIBILITY OF THE CODE	40
ANNEX B: RESULTS TABLES FOR GO TERMS PREDICTIONS	41
ANNEX C: JOB SCRIPT RUN IN MARENOSTRUM4	42

List of tables

TABLE 1. PREVIEW OF THE PHYLOGENETIC PROFILING MATRIX (WITH GO TERMS ASSIGNED) FOR THE HUMAN
PROTEOME.....**23**

TABLE 2. COMPARISON OF THE PERFORMANCES BY F1 SCORE OF THE DIFFERENT PARAMETERS SELECTED.....**29**

TABLE 3. DATASETS DESCRIPTIONS FOR THE 3 DISEASES RETRIEVED FROM DISGENET.....**32**

TABLE 4. MEAN PREDICTION PERFORMANCES OF RANDOM FOREST CLASSIFIER AT PREDICTING 3 DIFFERENT DISEASES
BASED ON PHYLOGENETIC PROFILES. SCORES OBTAINED BY PERFORMING 5-FOLD CROSS VALIDATION 10 TIMES,
RESAMPLING THE NEGATIVE SAMPLES ON EACH.....**33**

List of figures

FIGURE 1. THE CENTRAL DOGMA OF MOLECULAR BIOLOGY. FROM HTTPS://IB.BIONINJA.COM.AU/STANDARD-LEVEL/ TOPIC-2-MOLECULAR-BIOLOGY/27-DNA-REPLICATION-TRANSCRI/CENTRAL-DOGMA.HTML.	3
FIGURE 2. ONTOLOGY GRAPH FOR <i>GO:0005739</i> . FROM HTTPS://WWW.EBI.AC.UK/QUICKGO/TERM/GO:0005739	5
FIGURE 3. DIFFERENCES BETWEEN ARTIFICIAL INTELLIGENCE, MACHINE LEARNING AND DEEP LEARNING.	6
FIGURE 4. EXAMPLE OF A SUPERVISED MACHINE LEARNING WORKFLOW.	7
FIGURE 5. CONFUSION MATRIX, WHERE <i>P</i> REPRESENTS POSITIVE AND <i>N</i> NEGATIVE.	8
FIGURE 6. ACTIVITY DIAGRAM OF THE MAIN WORKFLOW.	15
FIGURE 7. EXAMPLE OF A DECISION TREE. FROM M. KUBAT, <i>AN INTRODUCTION TO MACHINE LEARNING</i> . SPRINGER INTERNATIONAL PUBLISHING, 2017.	18
FIGURE 8. RANDOM FOREST FOR CLASSIFICATION. FROM HTTPS://WWW.RESEARCHGATE.NET/FIGURE/VOTING-IN-RANDOM-FOREST_fig2_272815108	21
FIGURE 9. STEPS FOR OBTAINING THE DATA MATRIX FOR THE BINARY CLASSIFICATION.	24
FIGURE 10. ROC CURVE OF SEVERAL MODELS AT PREDICTING "MITOCHONDRIAL LOCALIZATION".	26
FIGURE 11. PERFORMANCE OF SEVEN ML CLASSIFIERS AT PREDICTING 997 DIFFERENT BIOLOGICAL PROCESS GO TERMS BASED ON PREVIOUS PHYLOGENETIC PROFILE MATRIX. (A) BOXPLOT OF ACCURACY SCORES. (B) BOXPLOT OF F1 SCORES. (C) BOXPLOT OF ROC-AUC SCORES.	28
FIGURE 12. ROC CURVE COMPARING THE BEST, WORST AND DEFAULT PARAMETERS FOR RANDOM FOREST CLASSIFIER AT PREDICTION "MITOCHONDRIAL LOCALIZATION".	30

CHAPTER 1.

INTRODUCTION

With the advances in the Next Generation Sequencing technologies and the decrease in their prices, the number of identified gene sequences is growing exponentially. However, the function of the majority of these newly identified genes remain unknown. The UniprotKB database currently stores more than 189 million sequences but only around 600 thousand are reviewed and functional annotated [1]. Experimental methods are not scalable enough to fill in this gap, as they are very laborious, take much time and money, and are typically focused on a single gene or a single function. Therefore, different bioinformatics approaches have been proposed to address this problem [2].

Machine learning represents an attractive avenue to help fill in this gap. It has recently become very popular, since it is a tool that has been proved to be effective at handling and exploiting the potential of the current overwhelming amount of data of modern society. Although the maths behind have been known since the early 80s, it has not been until now that we are able to take advantage of it, as now we have powerful enough machines that can deal with the computational demands. In this way, supercomputers have played an important role, enabling scientists to run intense and complex Machine Learning algorithms.

One recent successful example of Machine Learning in Biology is the AlphaFold project, developed by Google's DeepMind, which consists of an artificial intelligence program able to accurately predict protein structures from their amino-acid sequence, a 50-year-old grand challenge in biology [3]. This algorithm placed first, with a great difference from the second, in the 2020's CASP¹ challenge.

Similarly, it also exists a challenge, called CAFA², which provides a large-scale assessment of computational methods dedicated to predicting protein function. Automatic Function Prediction (AFP) has been an active and growing research field for decades and has made considerable progress in that time [4], but it is still far from being solved. The performance of the techniques used so far has not reached the expected level of accuracy. Here, we would like to make our little contribution to such community-driven effort and assess a method that could help in the computational annotation of protein function.

Next subsections introduce the main necessary concepts to formulate the objectives of the work and to understand the next chapters.

¹ **CASP**: Critical Assessment of Structure Prediction.

² **CAFA**: Critical Assessment of Functional Annotation.

1.1 Bioinformatics

Bioinformatics, also called computational biology, is an emerging interdisciplinary field between Biology, Computer Science and Statistics. It has gained a lot of attention since the completion of the Human Genome Project³ almost two decades ago, when life scientists started to generate data at high rates. This made impractical for human to conduct research with traditional methods. Information technologies were required to handle this huge amount of data and to analyse and extract insights from it. In addition, the data generated by scientists is grown exponentially every year, making bioinformatics a challenging field in which having a good computer science foundation is more and more relevant.

Common applications of bioinformatics include medical image analysis, ontology and data integration, new drugs discovery, **omics data analysis**, among others. We will focus on the field of omics data analysis. First, to correctly understand where the word *omics* comes from, let me introduce some of the basic concepts in Biology:

- **Genome**: refers to all genetic material of an organism, such as DNA⁴ and/or RNA⁵. A sequence of DNA or RNA within a genome is called **gene** when it encodes the synthesis of a molecule, usually a protein.
- **Proteome**: comprises all the proteins of an organism.

As you may probably notice, all these concepts end with a *-ome* suffix. This usually indicates that the coverage of the group is an entire organism/organ/tissue, and the starting part of word denotes the type of molecule in question. In the same way, the suffix *-omics* refers to the study of such fields. So, the study of the genome is called genomics and of the proteome, proteomics. Biologists, as well as other scientists, make use of plenty of Latin words, both as suffix and prefix, to help characterize and add small additional meanings to some already established words in the field. Another relevant prefix in bioinformatics is *meta-*, which means that the field covers not only one but several organisms. Therefore, metagenomics is defined as the study of the genomes across various different organisms. These are only some examples of them, but every year more subfields come out.

We are not going into detail on what each of these molecules (gene, protein, etc.) implies or are made up of, but it is important for the understanding of this work to have at least an overall knowledge of their roles and life cycles.

1.1.1 Central dogma of molecular biology

In *Figure 1*, it can be seen what it is known as central dogma of molecular biology. It represents the foundation on which molecular biology is based. It describes the process by which the DNA instructions are converted into proteins, also known as **gene expression**. This includes two main stages: transcription, or conversion of DNA into RNA messages, and translation, or conversion from RNA to protein. There are also other intermediate steps like replication or reverse transcription, but they are not relevant for the scope of this work.

It is important, though, that we keep in mind the differences between a gene and a protein. These terms are usually used interchangeably, as in general one gene produces one protein, but there can be cases in which one gene can produce different proteins (by different kind of molecular mechanisms) or even no protein (then, they are called non-coding genes).

³ **Human Genome Project (HGP)**: international scientific research project with the goal of determining the base pairs that make up human DNA. It was successfully completed on April 14, 2003.

⁴ **DNA**: Deoxyribonucleic acid.

⁵ **RNA**: Ribonucleic acid.

That is why it exists organizations, such as NCBI⁶ or EMBL⁷, whose intention is to organize and standardize both DNA and protein sequences as well as provide a unique reference database to retrieve reliable genetic information.

Among the different types of genes, we are interested in those that have or can have a function, that is basically, coding genes. These type of genes would pass through the 3 steps stated in *Figure 1*, given proteins as a result. Proteins are the molecules that are in charge of carrying out some of the main and most important tasks within a living organism. We can

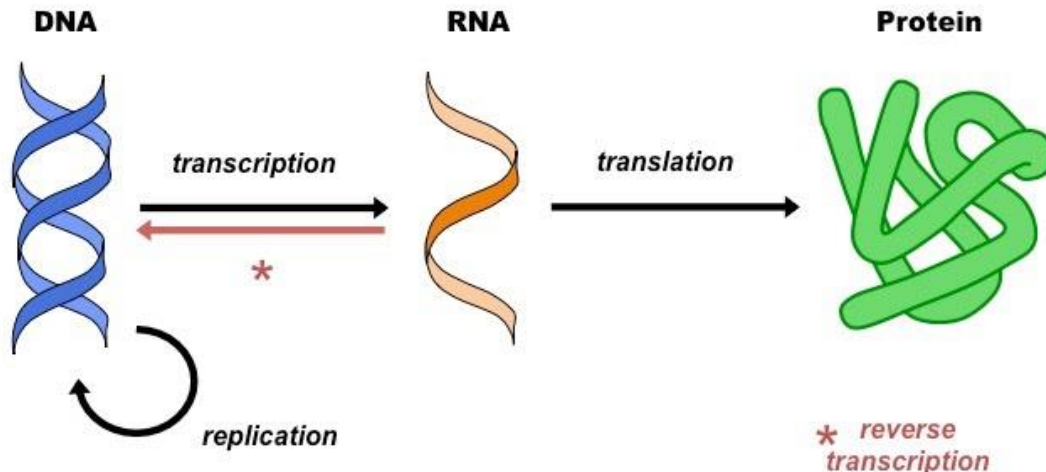


Figure 1. The central dogma of molecular biology. From <https://ib.bioninja.com.au/standard-level/topic-2-molecular-biology/27-dna-replication-transcri/central-dogma.html>.

see them as the executors or “workers” within our body. That is why knowing their function is of high relevance for the scientific community. It does not matter if we have already identified a new gene, if we do not know its function, we cannot infer if this new gene is involved in any pathway, disease or biological process of interest.

1.1.2 Homology

Homology is a widely used term in different fields. It is often used to express “similarity” or “homogeneity” between entities. From an evolutionary point of view, homology make reference to the relationship of two characters that descended from another common ancestral character. A character could be a gene, morphology, or behavioural feature of an organism. In this case, as we will be working in a genomics context, we will be referring to homology as sequences or genes that share a common ancestor. Homologous genes are usually identified by means of sequence similarity. Significant similarity is strong evidence that two sequences are related by evolutionary changes from a common ancestral sequence. There are two types of homology of genes:

- **Paralogous genes:** pair of homologous genes resulted from a duplication event.
- **Orthologous genes:** pair of homologous genes that have emerged through a speciation event.

In short, we will be referring as orthologs to homologous genes between species, and paralogs when it is within the same specie. Resolving the orthology across genes of interest can be used to derive possible functions of uncharacterized genes, since, as it is claimed by the “**orthology–function conjecture**”, orthologs are on average functionally more similar

⁶ NCBI: National Centre for Biotechnology Information.

⁷ EMBL: European Molecular Biology Laboratory.

than paralogs [5] and thus, can be used for function transfer. Under this premise, gene function for experimentally inaccessible organisms is hypothesized using known functions from orthologs of well-investigated organisms. This is the same foundation a technique called **Phylogenetic Profiling** is based on. We will be using this technique in order to infer functional relationships between genes.

1.1.3 Gene function

The function of a gene is typically defined as the role that its protein product plays in situ, i.e. the live cell. However, live cells are not perfectly designed systems where every part has its own well-defined role (the “function”), in reality, cell components participate in a complex network of interactions and often have more than one role. Historically, gene function is based on the first description of the properties of the respective mutant or the biochemical activity of the purified protein. For non-coding genes, the definition of “function” can be a bit fuzzy, as they usually do not contribute directly to a biological process or pathway. Therefore, we will be referring to gene function to only those genes that produce a gene product or protein, that is, protein function.

1.1.4 Gene Ontology

A proper definition of the gene product function should probably combine characterization of its biochemical activity, if any (i.e., the nature of the catalysed reaction and the range of utilized substrates and products), with the description of the biological process (e.g., a metabolic or signalling pathway) that involves this protein. The Gene Ontology consortium tries to do so by providing three different ontologies⁸:

- **Cellular Component:** locations relative to cellular structures in which a gene product performs a function component of a cell, either cellular compartments (e.g., *mitochondrion*), or stable macromolecular complexes of which they are parts (e.g., the *ribosome*).
- **Molecular Function:** activities occurring at a molecular level (e.g., nucleotide binding, virus receptor activity, etc.).
- **Biological Process:** the larger processes, or ‘biological programs’ accomplished by multiple molecular activities.

Each of these ontologies represents a structured and controlled vocabulary, which includes terms which show a relationship that can be visualized in a graph. Each term belonging to one of three ontologies has a definition, a unique name and an identifier.

For instance, a very interesting GO term for evolutionary studies is the *GO:0005739*. It is defined as: “A semiautonomous, self-replicating organelle that occurs in varying numbers, shapes, and sizes in the cytoplasm of virtually all eukaryotic cells. It is notably the site of tissue respiration.”. It belongs to the Cellular Component ontology and its whole lineage can be seen on *Figure 2* in a graph representation. All this information is freely accessible on websites such as QuickGO⁹ or AmiGO¹⁰.

⁸ **Ontology:** formal representation of a set of classes (or terms or concepts) with relations that operate between them. Usually used to describe the overall knowledge of a given domain.

⁹ <https://www.ebi.ac.uk/QuickGO/>

¹⁰ <http://amigo.geneontology.org/amigo>

It is important to note that, although each ontology (cellular component, biological process, and molecular function) can be used in a completely independent way, there can be some relationships, such as *part of* or *regulates*, that can operate between the GO ontologies. Further, genes tend to have multiple GO terms associated and the ideal would be that they had at least one GO term from each ontology. Unfortunately, there is still a long path to achieve this.

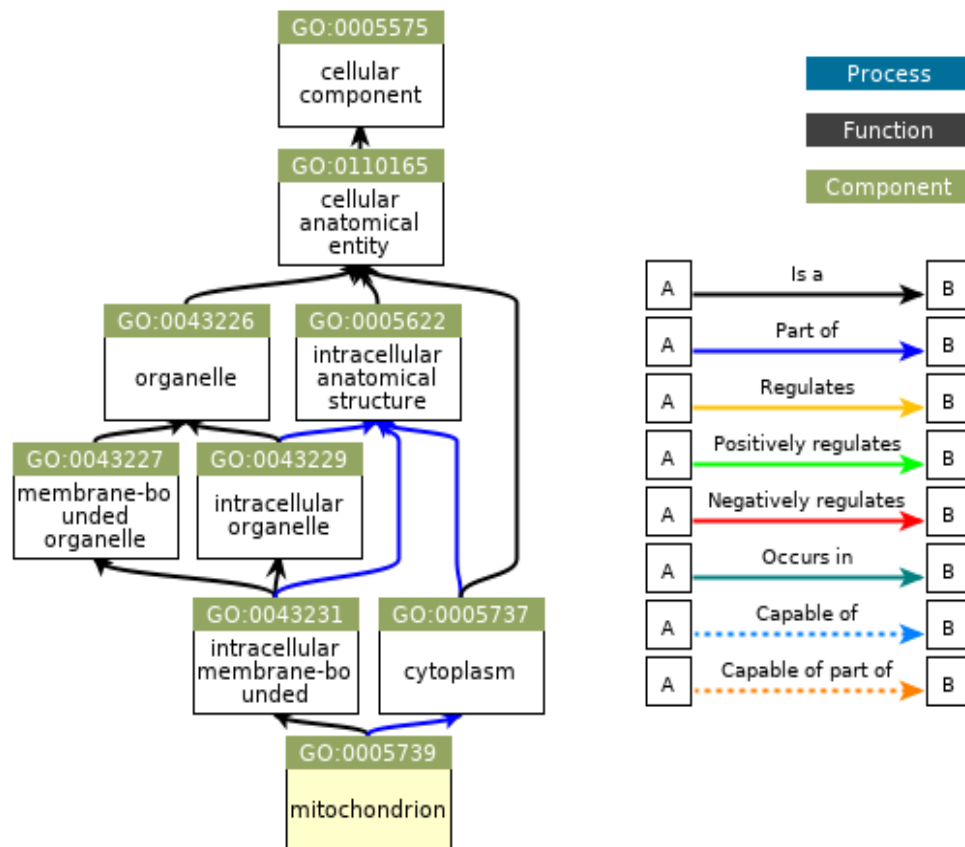


Figure 2. Ontology graph for *GO:0005739*. From <https://www.ebi.ac.uk/QuickGO/term/GO:0005739>.

Although we will usually be using the terms GO annotation and GO term indistinguishably, we should make clear that, in fact, they do not have a one-to-one relationship. This is, one GO annotation does not uniquely correspond to one GO term, and vice versa. A GO term is not an identifier of a GO annotation either, as one could think. A GO term itself only represents an entity within the ontology graph, but an annotation intends to make a statement about the function of a particular gene by combining 4 different pieces of information. Even though it may have additional components, these four are the pieces that uniquely identify a GO annotation:

- Gene product: may be a protein, RNA, etc.
- GO term.
- Reference: database the annotation comes from, which can be organizations like the official Gene Ontology Database or others such as Reactome or Pubmed.
- Evidence: supporting evidence, which can be of different types such as experimental evidence, phylogenetic-inferred, computational evidence or author and curators' statements, among others.

1.2 Machine Learning

In recent years, Machine Learning has become a very popular topic in many areas because it has demonstrated to be a powerful and effective tool to gain knowledge from complex and large data. However, many people overused this term, and often it is used as a synonym of Artificial Intelligence, which is wrong.

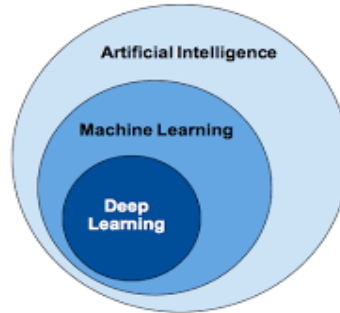


Figure 3. Differences between Artificial Intelligence, Machine Learning and Deep Learning.

As can be seen in *Figure 3*, Deep Learning is a subset of Machine Learning, and this itself is a subset of Artificial Intelligence, which is a term used to refer to any technique that enables computers to mimic human behaviour. Machine Learning though, refers to an algorithm that is able to extract knowledge from a given data to then learn to further differentiate the patterns underlying this information. This also applies to Deep Learning but with the addition that deep learning algorithms rely on an special kind of artificial neural networks¹¹ to perform the analysis.

1.2.1 Types of Machine Learning

There are two primary types of machine learning methods: supervised learning and unsupervised learning.

Supervised Machine Learning models learn the relationship between input features of data and some labels or outputs associated with a given known data. There are two types of supervised machine learning:

- **Regression:** process of finding a mapping function for distinguishing the data into **continuous** real values.
- **Classification:** process of finding a mapping function which helps in separating the data into multiple categorical classes i.e. **discrete/categorical** values.

In this work, we are going to focus on classification, since it is the type of problem that will be facing on this project. In classification algorithms, we provide the algorithm with a set of objects described by different features and also provide the given label of each object. We should understand **features** as any kind of characteristic or “descriptor” of the data, while **labels** are the final discrete outputs or classes of the given data. In binary classification, we have only one class/label of interest (e.g., cancerous cell), and therefore, the model will try to classify the objects into cancerous or non-cancerous cells. If there is more than one target class (for instance, in a fruits classification problem we might provide the algorithm with examples of apples, bananas, and oranges), we have a multiclass classification.

¹¹ **Artificial neural networks:** computing systems consisted of connected units or nodes, called *artificial neurons*, designed to simulate the way the human analyzes and processes information.

Once the model has been fit to a set of examples, which can also be referred to as *training data*, it can be used to predict/classify the labels of newly, unseen data. That is why these models are often called *classifiers*. Many different supervised machine learning methods are available, e.g., *Random Forest*, *Support Vector Machines* or *Neural Networks* (base model for Deep Learning). We are not going to get into further detail on how each model works, but it is important to keep in mind that this is the type of machine learning that we will be using for this work.

Unsupervised Machine Learning models infer patterns from unlabelled data. *Clustering* and *Dimensionality reduction* are some very popular unsupervised algorithms. Clustering algorithms discover and build distinct groups of data, while Dimensionality reduction algorithms search for more succinct representations of the data. In biology, these algorithms have been using for decades. For instance, dimension reduction tools like PCA¹², tSNE¹³ or UMAP¹⁴ are often used in life sciences studies to visualize and find patterns in high dimensionality data such as *omics* data. Also, clustering tools such as Hierarchical Clustering are being used for the construction of phylogenetic trees.

1.2.2 Life cycle of a ML model

The diagram in *Figure 4* gives a high-level overview of the stages in a typical ML workflow [6]–[8]. Data pre-processing is a very broad term that can include different steps such as data collection, data cleaning, feature engineering, etc. It is used to be the most time-consuming step in the analysis since the data must be reliable and have enough information for the ML model to learn from it.

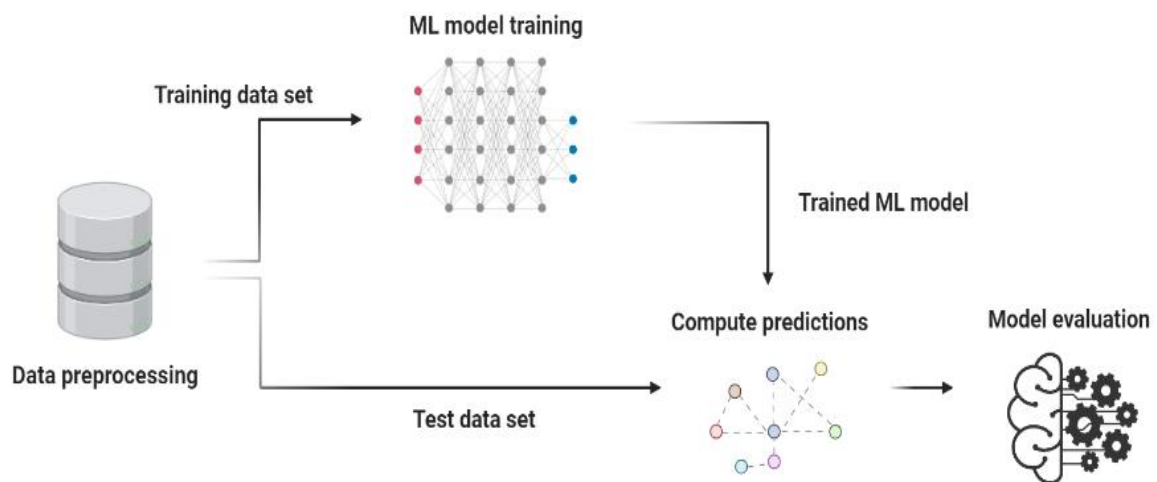


Figure 4. Example of a Supervised Machine Learning workflow.

Also, we should differentiate between two kinds of data during the analysis: training and test data. The training data is the part of the data that we will be using to train the ML model, whereas the test data is the part of the data that will be used for evaluating the performance of the model at the end of the analysis. This split is done before the model training step, as the test data will be held apart until the evaluation stage. Usually, the test data represents a 20-25% of the total available data. A good practice for avoiding biases in the test data, is to perform what is known as: **Cross Validation**. This technique consists of

¹² **PCA**: Principal Component Analysis.

¹³ **tSNE**: t-Distributed Stochastic Neighbor Embedding.

¹⁴ **UMAP**: Uniform Manifold Approximation and Projection.

repeating the whole analysis a determined number, k , of times, taking as test data a different set of the data. Thus, the average performance of these k classifiers stands in as the performance of the full model. The k will depend on different factors, like the size of the original data or the hardware restrictions, but it is usually around 5 or 10.

Another important point to take into account is regarding overfitting and underfitting. Overfitting occurs when the method learns to perfectly classify the training examples. This supposes that the model can not generalize well on unseen data and thus the performance in the test data drops dramatically. In contrast, underfitting refers to a model that can neither classify correctly the training nor the test set. This indicates that the model is not suitable for the analysis.

1.2.3 Evaluation metrics

There are several metrics that can help us measure how good a model is performing [8]. In Supervised Machine Learning, evaluating a model means measuring how accurately our predicted labels match the labels of the test dataset. For the common case of two distinct labels or class (binary classification), we typically call the smaller and more interesting of the two classes as positive and the larger/other class as negative. For instance, in a spam classification problem, the spam would typically be positive, and the non-spam would be negative.

		Predicted class	
		P	N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

Figure 5. Confusion matrix, where P represents positive and N negative.

Based on the correctness of the predicted labels, we can differentiate 4 possible results, which can be represented in a table called *confusion matrix*, as shown in Figure 5. Biologists are used to some of these terms, such as *false positive* or *false negative*, but in the data science, these terms are only the base line to other more descriptive evaluation metrics:

- **Accuracy:** ratio of the number of correct predictions over total predictions.

$$accuracy = \frac{TP + TN}{TP + TN + FN + FP} \quad (1)$$

- **Precision:** measures how often the classifier is correct when it gives a positive result. It penalizes the false positive results.

$$precision = \frac{TP}{TP + FP} \quad (2)$$

- **Recall (or True Positive Rate or Sensitivity):** measures the sensitivity to make positive predictions. It penalizes the false negative results.

$$recall = \frac{TP}{TP + FN} \quad (3)$$

- **False Positive Rate (FPR):** ratio of between the number of negative results wrongly categorized as positive (*false positives*) and the total number of actual negative results.

$$FPR = \frac{FP}{FP + TN} \quad (4)$$

The metric we should use depends totally on the analysis we are carrying out. The accuracy gives us a general idea of the performance of our model but sometimes can be misleading. If the test data we are using to measure the performance is not balanced, that is, there is more samples of one label and only a few of the other, the accuracy would be high just by guessing that all samples were of the label overrepresented. For these cases, it is much more convenience to use another metric such as precision or recall.

Depending on what we are interested in more, we will choose either precision or recall. For instance, in cancer diagnosis, we might be more ready to tolerate false positives (errors where we scare a healthy person with a wrong diagnosis) than false negatives (errors where we kill a sick patient by misdiagnosing their illness). In this case, we should use recall. For email spam detection though, we do not want to lose our non-spam emails in the spam folder, we rather prefer to eventually send some of these spam emails to its folder, so precision is the metric to use.

Also, there are others metrics also very used in research, which are based on the previous ones. These two following metrics are very useful when we deal with imbalanced data, which are most of the real problems:

- **F1 score:** harmonic mean between recall and precision. Achieving a high F-score requires both high recall and high precision.

$$F1\ score = \frac{1}{\frac{1}{recall} + \frac{1}{precision}} \quad (5)$$

- **ROC-AUC or AUROC (Area Under the Receiver Operating Characteristics):** indicates the total area under a ROC (Receiver Operating Characteristics) curve. Each point on this curve represents a particular classifier threshold, defined by its *True Positive rate (TPR)* and *False Positive Rate (FPR)*. If we computed a ROC curve of a random classifier, which predicts an observation as positive or negative class at random, then the examples example should be positive or negative with equal probability should be positive or negative with equal probability (that is, a ROC-AUC of 0.5). ROC curves can be very useful for task such as analysing the predictive power of a classifier or determining optimal threshold for a binary classifier [9]. The higher AUC, the better the model is at distinguishing between positive and negative class. It ranges from 0 to 1, being 1 the ideal measure of separability.

1.3 About the centre

I was fortunate to carry out this project as a research student at the Comparative Genomics group¹⁵. The group leader, Toni Gabaldon, which acted as my supervisor for this project, has published over 600 papers and accumulates more than 20.000 citations. The group is jointly affiliated to the Biomedical Research Institute (IRB) and the Barcelona Supercomputing Centre (BSC), at Barcelona. Two of the most recognized research centres in Spain. This allows the group to work in both experimental lab (wet lab) and *in-silico* (dry lab). IRB provides the group with the best lab equipment, while BSC let them run high intense and complex algorithms thanks to the power of MareNostrum4 supercomputer.

The main research interest of the group is to understand the complex relationships between genome sequences and phenotypes and how these two features evolve within and across species. They generally use large-scale phylogenetics and molecular evolution approaches that allow looking at the evolution of genomes from the perspective of all of their genes, and they apply these analyses to a variety of biological questions related the evolution and function of biological systems, ranging from individual molecules to entire biological communities.

During my stay in the group, I was able to learn a lot about phylogenomics and bioinformatics. Sharing and discussing the achievements of the project with the group helped me to improve my critical-thinking and communication skills. This project was developed as a continuation of a previous project that I was in charge of during the 2020's summer. For that stay I was awarded with a Maths4Life¹⁶ fellowship from the IRB, which let me have the first contact with the group. The objective of the project was to develop a pipeline for benchmarking MetaPhOrs' orthology database using the Quest for Orthologs service. These same databases and results were the starting point for the present project.

1.4 Objectives

The overall objective of this work consists of studying the use of Phylogenetic Profiling combined with a Supervised Machine Learning approach to predict protein function. As particular case of study, we want to test the algorithm to find proteins with potential disease-related functions.

To achieve this goal, we have the following sub-objectives:

1. Develop an algorithm to mine the MetaPhORs database to detect co-evolving proteins based on presence/absence across the reference species set by the Quest for Orthologs consortium.
2. Use this information as input of a Machine Learning method to construct a model that predicts protein function.
3. Train the model with reference proteomes of around 78 organisms, set by the Quest for Orthologs, and more than 12 million orthologs available in the MetaPhOrs database. The management of this large dataset requires the use of a large computational power. We will implement our workflow using parallelization algorithms to be executed in the MareNostrum supercomputer.
4. Assess the performance of the predictions of this model in terms of accuracy and other relevant metrics.
5. Select different diseases for the testing of the model obtained, in particular, neurodegenerative diseases or other diseases with a high phylogenetic signal.
6. Evaluate the performance of the model in that cases.

¹⁵ <http://cgenomics.org/>

¹⁶ <https://www.irbbarcelona.org/en/talent/young-scientists/maths4life-2020>

1.5 Structure of the document

The rest of the document is organised in three chapters as follows:

- Chapter 2 explains the process of representing the data using a Phylogenetic Profile matrix, the design and implementation of the software needed to construct and test different models with several Machine Learning algorithms. The chapter ends with a proposal of the best ML method for the purpose of this work.
- Chapter 3 focus on the clinical application of the project. In this chapter we apply the ML method selected before to a potential use case of this kind of algorithms.
- Chapter 4 makes a discussion of the work done, gives the conclusions drawn from this work and presents some lines of future work.

CHAPTER 2.

LEARNING METHOD SELECTION

In this chapter we will go through the first step in order to achieve the objectives of the project: The Machine Learning method selection. We will discuss the different methods we tried as well as the problems we encountered on each phase. At the end, given the results we obtained, we will show which learning method has been selected and why.

2.1 Introduction

Phylogenetic Profiling has already proven several times to be an effective method to assign protein function, and it has also demonstrated that can be successfully applied to the human proteome [10]. However, these analyses were normally carried out by clustering methods. Instead, we want to study if we can construct a model that can effectively learn from the phylogenetic profiles to then use this model to assign protein function (in the form of GO terms) to uncharacterized proteins. We are going to use the power of a supercomputer like the MareNostrum4 to leverage this analysis into large-scale. We are going to use the reference proteomes of around 78 organisms, set by the Quest for Orthologs, and more than 12 million orthologs available in the MetaPhOrs database. Note that we will be using only orthologs, since they tend to conserve function more than paralogs [5].

The learning method selection is the process of selecting a single Machine Learning (ML) method from among a collection of candidate machine learning techniques for a training dataset. In the following sections, we will discuss what methods we used and why we chose Random Forest as the best suited for our purposes, as well as the whole process to achieve this decision. Details about the databases and algorithms will be explained.

2.2 Materials

Before designing an implementation proposal, it is important to have a good knowledge of what tools and materials we have. This will conditionate all the design of the project.

2.2.1 Tools

We have been able to use **MareNostrum 4 (MN4)**, the most powerful supercomputer in Spain, for conducting this project. With this computer, we were able to run demanding algorithms without any further complication. The MN4 uses openSUSE distribution of

Linux as Operative System and SLURM¹⁷ as workload manager. A supercomputer is no more than a cluster (or group) of small computing units (or nodes) that can work together to run a given task. The main point to take into account, however, is that we need design the program or algorithm to be run in parallel, otherwise we will not take the most of this kind of machines. More details about its specs in the section below.

The programming language chosen for this project was **Python** (v3.6), since it is the most used and popular language for data science and artificial intelligence. There are plenty of ready-to-use packages and frameworks for Machine Learning in Python. The one that we will be using is called **Scikit Learn** (v0.23.2), the best suited for classical Machine Learning. Other packages that were used and I would like to highlight are the following:

- Pandas (v1.1.5), for general data analysis.
- Seaborn (v0.11.1) and matplotlib (v3.2.2), for data visualization.
- ETE3 (v3.1.2), for phylogenomics data analysis [11].
- GOAtools (1.0.15), for GO terms and annotations analysis [12].

In addition, something that I would like to add as one of the most helpful tools used for this project is **Google Colab**, a very powerful online programming IDE¹⁸. It is like a Jupiter Notebook but vitaminised and with a “free” (with a maximum of 12 hours) runtime. It allows you to code in Python, write down notes, visualized plots and more. All in real time and in the same place. I highly recommend it.

2.2.2 HPC environment: *MareNostrum*

MareNostrum is the name of the main supercomputer of the Barcelona Supercomputing Center (Barcelona, Spain). Its last version, the MareNostrum4, is the most powerful supercomputer in Spain and it ranks 42th in the Top500 to date [13]. The supercomputer is used in human genome research, protein research, astrophysical simulations, weather forecasting, geological or geophysical modelling, and the design of new drugs. BSC-CNS also has other clusters, mainly of emerging technologies, such as IBM POWER9 (with NVIDIA Volta GPUs), AMD Rome (with AMD Radeon Instinct MI50) or 64 bits ARMv8. However, the main and most powerful block of MareNostrum4 has the following characteristics:

- Block of 48 racks with 3456 nodes. Each node has two **Intel Xeon Platinum** chips, each with 24 processors, amounting to a total of 165,888 processors and a main memory of **390 Terabytes**.
- Peak power of **11.15 Petaflops**. While its performance is 10 times greater than its predecessor, MareNostrum 3, its power will only increase by 30% to 1.3 MW.
- Disk storage capacity of 14 Petabytes.
- **High-speed Omnipath network** connecting all the components.

Among the 3456 nodes available, there are 216 specially reserved for high memory executions, that is, each *highmem* node has 384Gb of RAM instead of 96Gb, as the others. Everything is managed by SLURM, and thus if we want to run a job, it has to be sent to a queue and we will have to wait more or less depending on the resources available at that time or the priority we had assigned. Researchers in my group normally books from 1 to 3 nodes, and if we would like to use more than that, we must contact the IT team from BSC. Nevertheless, we should take into account that if we will use more than one node, the parallelization must be designed for multi-computing resources, and not only for multi-

¹⁷ **SLURM**: Simple Linux Utility for Resource Management.

¹⁸ **IDE**: integrated development environment.

processors (within only one node). This means that tools like openMP or built-in parallelization libraries such as *joblib* in Python, will not work for multiple nodes. We should use MPI or related technologies. In fact, BSC provides a proprietary tool called Greasy, that facilitates this task. It uses a MPI engine to provide the user with multi-computing parallelization support but as a ready-to-use and configurable wrapper.

Personally, one node was enough for my needs of computation, and thus I did not need to use Greasy. One normal node (not *highmem*) has 48 cores (24 cores x 2 processors) and 96 Gb of RAM. More than enough for many users.

To run jobs with SLURM, we had to login in one the 3 login nodes available. Only one of them has internet (recently implemented). Before submitting a job with *srun* or *sbatch*, we should make sure that all the data and packages that we need for the correct execution of the job are found in one of the storage nodes, as none of the computing nodes has internet connection. An example of a submitted job can be found on *Annex C*.

2.3 Design

In this section, we will present the specifications and details about the design of the code for learning a classification model. Specifications were agreed with the supervisor of the Comparative Genomics research group.

2.3.1 Specifications

Some specifications are given by the requirements of the project and some others by the tools and materials we need to use. The main specifications of the software to develop in this project are the following:

01. The algorithm should be designed to be run in an HPC¹⁹ environment. This implies the following: avoid having too many dependencies and parallelize the code as much as possible.
02. For the specific case of MN4, there are some points to be aware of:
 - i. It has no internet access in the computing nodes, so the code should not need online communication during the execution.
 - ii. There is a maximum runtime of 3 days for each submitted task, so the program should not take more than this to finish its execution.
03. The phylogenetic profiling should support as input a tabulated list of orthologs, no matter if the orthologs are from MetaPhOrs' or other orthology database.
04. It should be able to be run with any number of proteins and reference species.
05. It should be aware of what organisms the protein samples belong to.
06. It should be modular and prone to future changes and improvements.
07. It should allow different filters, especially for the GO terms manipulation. We would like to adjust parameters such as the GO aspect (Cellular Component, Biological Process or Molecular Function), GO evidence codes or maximum and minimum of GO terms.
08. The Machine Learning process must have support for different ML methods.
09. The workflow must be modular, that is, the different steps must be able to be run separately.

2.3.2 Activity Diagram

Given the specifications above, we designed the activity diagram shown in *Figure 6*. The databases to be used for this algorithm are in different format and they will have to be parsed accordingly. The resulting model would be able to predict the function for new

¹⁹ HPC: High Performance Computing.

protein instances. As we are interested in test the performance of this model, the final output are the performance values, which will be saved in a tabular-separated file. In addition, the intermediate Phylogenetic Profiling matrix, both with and without GO terms associated, can also be saved in a tabular-separated file, if the user explicitly specify so.

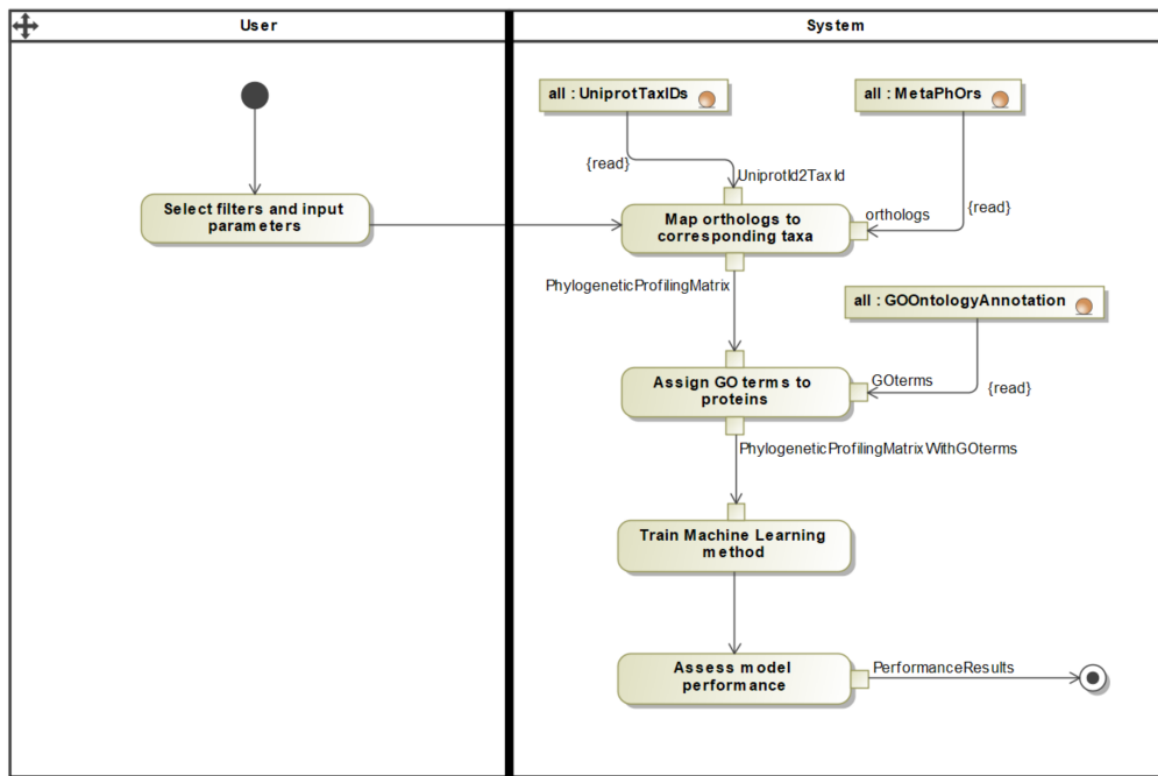


Figure 6. Activity Diagram of the main workflow.

2.4 Methods

The methods and tools described in this section are the ones that have been studied in this work.

2.4.1 Databases and data retrieval

We retrieved data from different databases. For more information about endpoints, downloading commands, etc., you can check out the notebook called *1_Data_Retrieval.ipynb* on my GitHub repository (see Annex A).

First of all, we have used previous data developed in the research group. In particular, we used **MetaPhOrs (MtP)**²⁰, a public repository that integrates phylogenetic information from different sources to provide orthology and paralogy relationships based on a common phylogeny-based predictive algorithm [14]. In my previous stay in the group, I was in charge of developing a pipeline that could periodically perform an orthology benchmarking of the MetaPhOrs database using an online service provided by the **Quest for Orthologs**²¹ consortium. This consortium aims at simplifying and standardizing orthology benchmarking by providing and maintaining a reference set of proteomes as well as a web server for launching a benchmarking workflow [15]. The service requires a tabulated list of orthologs

²⁰ <http://orthology.phylomedb.org/>

²¹ <https://orthology.benchmarkservice.org/>

predictions as input. Apart from the experience acquired from this previous project, we took some results files from it. In summary, what we took from this previous project is:

- The Quest for Orthologs' reference set of proteomes from the 2018 release. In total there were 78 organisms.
- The MetaPhOrs' tabulated list of orthologs from the 2016 release. The orthologs were represented by its Uniprot identifier (UniprotID). In total, there were 12.564.907 orthologs.

Apart from these datasets, we also needed additional data to perform the data analysis. We retrieved most of the data from official and widely used bioinformatics databases such as **Uniprot**²², for the taxonomic ID mapping step, or **Gene Ontology Annotation (GOA)**²³, for the retrieval of the latest functional annotations. It is interesting to highlight that when working with this kind of databases that contains huge amount of information (around 20Gb for one file), it can be a bit tricky to filter or search for an item in them, as we will easily run into out of memory errors. In order to deal with this situation, we often have to use some Linux commands, which are usually faster and more efficient than Python functions in terms of resources management.

For the clinical application of the work, we used a databased called **DisGeNET**²⁴, which is a versatile platform, developed by Integrative Biomedical Informatics Group from the PRBB²⁵, that integrates information of genes and variants associated to human diseases from a great variety of reliable sources such as ClinGen, the Human Phenotype Ontology or Genomics England. It provides large datasets of disease-related genes associated with a level of evidence score [16].

2.4.2 Machine Learning methods

After a research in related literature [17]–[19], some well-known Machine learning methods were selected. No restrictions were imposed a priori, as we were in disposition of such powerful machine like MareNostrum, we could afford to train any model we were interested in. Thus, we selected at least one representative model of each family, for learning and comparative purposes.

For selecting the models that might perform better, it is important to know what kind of data we will be dealing with. The features, in this case, are all continuous values, specically, integer values. They correspond to the number of orthologs found for each specie (the features). The labels correspond to categorical values, in concrete, the GO terms assigned for each protein, which will be different depending on the protein. If we gather all the GO terms assigned, we will notice that they might be a lot and non-uniformly distributed. Accordingly, we should make sure that the methods selected does support multilabel classification.

2.4.2.1 Dummy Classifier

This not a real ML model. It is a fake model that tries to classify the given data using simple defined rules. It does not generate any insight about the data. In contrast, its behaviour is completely independent of the training data, as the trends in the training data are completely ignored and instead uses certain strategies to predict the class label. There exists different strategies. The more straightforward is the “most frequent” strategy, which consists of always predicting the most frequent class label in the training data. Another strategy,

²² <https://www.uniprot.org/>

²³ <https://www.ebi.ac.uk/GOA>

²⁴ <https://www.disgenet.org/>

²⁵ **PRBB**: Barcelona Biomedical Research Park.

called “prior”, always predicts the class that maximized the class prior (like “most frequent”). This model is used as a simple baseline to compare with the other classifiers. The final classifier must perform better than this dummy method. Moreover, it is especially useful for identifying imbalance or very biased data.

2.4.2.2 K-Nearest Neighbour (KNN)

This is the simplest method of classification. It classifies data points based on their proximity and association to other available data. This algorithm assumes that similar data points can be found near each other. As a result, it seeks to calculate the distance between data points, usually through Euclidean distance, and then it assigns a category based on the most frequent category or average. By finding the class of the majority of neighbours, you can conclude that the new instance belongs to that class. There is actually no learning involved, other than keeping track of all labeled data. The number of neighbours is given by the parameter K . The value of K may affect significantly to the quality of the model constructed. This method supports multilabel classification.

2.4.2.3 Support Vector Machine (SVM)

Support vector machines was introduced in the early 1990s and has been successful in applying to real-world classification and regression problems. One of the advantages of SVM is that it can operate with sparse data and the models are generated with relatively small sample set. It constructs a hyperplane where the distance between two classes of data points is at its maximum. This hyperplane is known as the decision boundary, because it is used for separating/classifying the classes of data points on either side of the plane. SVM becomes really powerful when it is combined with the so-called *kernels*. These consists of projecting the data into higher-dimensional space defined by polynomials and Gaussian basis functions, and thereby finding the best separator over a variety of non-linear functions without much additional cost. It does not inherently support multilabel classification.

2.4.2.4 Naïve Bayes

Naïve Bayes’ algorithm is another very simplistic algorithm for classification. Despite its simplicity, it has demonstrated some very good results in classification. It adopts the basis of a Bayes Theorem, which is stated by:

$$P(c|x) = P(x|c) \cdot \frac{P(c)}{P(x)} \quad (6)$$

Where:

- $P(c|x)$ means probability of c , when x has already been observed.
- $P(x|c)$ means probability of x , when c has happened.
- $P(c)$ and $P(x)$ denote the probability of c and x , respectively.

The conditions assumed in the Bayes Theorem are the following: the presence of one feature does not impact the presence of another in the probability of a given outcome, and each predictor has an equal effect on that result. There are three types of Naïve Bayes classifiers: Multinomial Naïve Bayes, Bernoulli Naïve Bayes, and Gaussian Naïve Bayes. It does not inherently support multilabel classification.

2.4.2.5 Neural Network (NN)

Primarily leveraged for deep learning algorithms, artificial neural networks, also known as Multilayer Perceptron (MLP), process training data by mimicking the interconnectivity of the human brain through layers of nodes. Each node is made up of inputs, weights, a bias (or threshold), and an output. If that output value exceeds a given threshold, it “fires” or

activates the node, passing data to the next layer in the network. Neural networks learn this mapping function through an algorithm called “backpropagation”, adjusting based on the loss function through the process of gradient descent. When the cost function is at or near zero, we can be confident in the model’s accuracy to yield the correct answer. It does support multilabel classification.

2.4.2.6 Random Forest (RF)

They are based on what are known as *decision trees*. These are high intuitive ways to classify or label objects, as it builds a binary branching structure in the form of a tree, which we can then traverse in order to find a solution. This usually implies a high chance of overfitting and therefore, Random Forest claims as a great option to reduce the overfitting by constructing more than one decision tree, each trained on different random parts of the same training set. These trained trees are then passed through a voting process in order to make a consensus choice. Random Forest can perform very well with categorical or semi-continuous data, or even with noisy or sparse data with a lot of missing values. In contrast, the computational cost is usually high, in both memory and time. It does support multilabel classification. We will explain more about this method within the following sections.

2.4.2.7 Gradient Boosting

As Random Forest, this model it is an ensemble method that is based on multiple decision trees construction. The main difference is that the trees are constructed one at a time, and each tree helps to correct errors made by previously trained trees [20]. One great disadvantage of this model is that it normally takes longer than others, as trees must be built sequentially. It does not inherently support multilabel classification.

2.4.3 Classification based on Decision Trees

As we will see later, models based on decision trees, i.e., Random Forest and Gradient Boosting, were the best performing learning methods for our problem. Therefore, in this section we will try to explain in more detail the foundation of these family of algorithms.

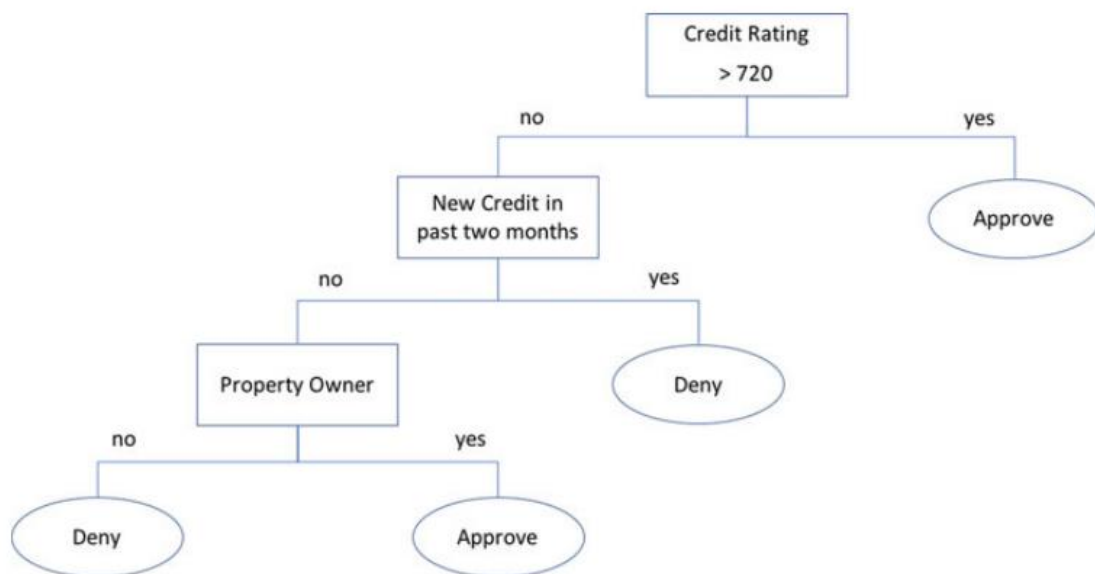


Figure 7. Example of a Decision Tree. From M. Kubat, *An Introduction to Machine Learning*. Springer International Publishing, 2017.

As explained before, these models are based on decision trees. A decision tree is composed of nodes representing deterministic decisions based on variables and edges representing the possible values of a variable, which are used to link with the next node or a leaf, with a class decision [7]. Each node represents a question and the edge from the node leading to next node represents the path taken based on the response. Leaf node or terminal node of the tree represents a class label as output of prediction. In summary, they intend to represent simple if-then rules in a tree structure (better for human readability). This can be seen more clearly in the decision tree example of *Figure 7*.

The decisions made at each node do not need to be binary decisions. However, due to practical reasons, they are usually constructed using binary decisions. There are different algorithms for learning decision trees. The most basic and used so far is the ID3 algorithm.

2.4.3.1 ID3 algorithm

ID3 learns decision trees by constructing them topdown, beginning with the question “which attribute should be tested at the root of the tree?”. To answer this question, each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples [6]. The best attribute is selected and used as the test at the root node of the tree. A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node (i.e., down the branch corresponding to the example's value for this attribute). The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree. This forms a greedy search for an acceptable decision tree, in which the algorithm never backtracks to reconsider earlier choices.

Algorithm 1: ID3

```

Input : examples ( $X$ ), targetAttribute ( $C(x)$ ), attributes ( $A$ )
Output: tree node
1 if  $C(x_i) = C(x_j) \quad \forall x_i, x_j \in X$  or  $A = \emptyset$  then
2   | return Node  $C(x_0)$ 
3 else
4   |  $a \leftarrow$  the best attribute among attributes having a range of values  $v_1, v_2 \dots v_k$ 
5   | Partition examples according to their value for  $a$  into sets  $S_1, S_2 \dots S_k$ 
6   | Create a decision node  $N$  with attribute  $a$ 
7   | for  $i \leftarrow 0$  to  $K$  do
8     | Attach a branch  $B$  to node  $N$  with attribute values  $v_i$ 
9     | if  $S_i \neq \emptyset$  then
10    | | Attach to  $B$  node from ID3( $S_i$ , targetAttribute, attributes -  $a$ )
11    | | else
12    | | Attach to  $B$  a leaf node with the most common targetAttribute
13    | | end
14  | end
15  | return decisionNode  $N$ 
16 end

```

The central choice in the ID3 algorithm is selecting which attribute to test at each node in the tree. The best attribute selection is generally performed using two metrics: *information gain* (based on *entropy*) or *Gini impurity* criterion. *Information gain* is generally used when the variables are categorical, i.e., when the variable values fall into classes or categories and do not have a logical order, for example, types of fruits. *Gini impurity* is used when the

variable values are continuous, i.e., the values are numerical, for example, age of a person [7].

2.4.3.1.1 Information Gain

In order to define *information gain* precisely, we begin by defining a measure commonly used in Shannon's information theory, called *entropy*, that characterizes the (im)purity of an arbitrary collection of examples. Entropy allow us quantify the variation in a set of examples with respect to the target attribute values. For instance, if we assume a binary value outcomes of 0 or 1:

- If all the outcomes are 1, then the entropy is zero as there is no uncertainty in predicting the outcome
- If outcome of 0 or 1 is equal, then the uncertainty or entropy is high.

The goal is picking the attribute that favors the best reduction in entropy, that is, the one best classifies the samples. For this, the Information Gain is expected to be reduced in entropy due to sorting S (training samples) on attribute A , as stated by:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \quad (7)$$

2.4.3.1.2 Gini impurity

Gini impurity is a measure of how often a randomly chosen element from the set is incorrectly labelled if it were labelled according to the distribution of labels in the subset. Higher *Gini impurity* refers to higher chance of misclassification conversely, and lower impurity refers to lower chance of misclassification. In constructing a Decision Tree split, the goal is to split with the lowest weighted Gini impurity value for the child trees. This translates to split with the largest reduction in weighted Gini impurity for child nodes from Gini impurity of parent node. In other words, the split will result in largest possible homogeneity in the child node.

The Gini impurity can be computed by summing the probability of an item p_i with label i being chosen times the probability (8) of a mistake in categorizing that item. It reaches its minimum (zero) when all cases in the node fall into a single target.

$$\sum_{k \neq i} p_k = 1 - p_i \quad (8)$$

2.4.3.2 Random Forest

Although decision trees has many advantages, such as comprehensibility and scalability, they suffer from important drawbacks:

- Instability, as a small change in the input may lead to great changes in the structure of the tree.
- Prone to overfitting, as the tree can grow as deeply to perfectly classify all the training samples, and thus, loose generalization capabilities.

An interesting way of overcoming these drawbacks is using ensemble methods, such as Random Forests. This method builds several decision trees and the final decision is taken by a *voting function* process (see Figure 8). This process can be implemented in different ways. *Scikit-learn* performs this process by averaging the class probabilities of the trees in

the forest. The predictions are then simply the class with the highest average class probability.

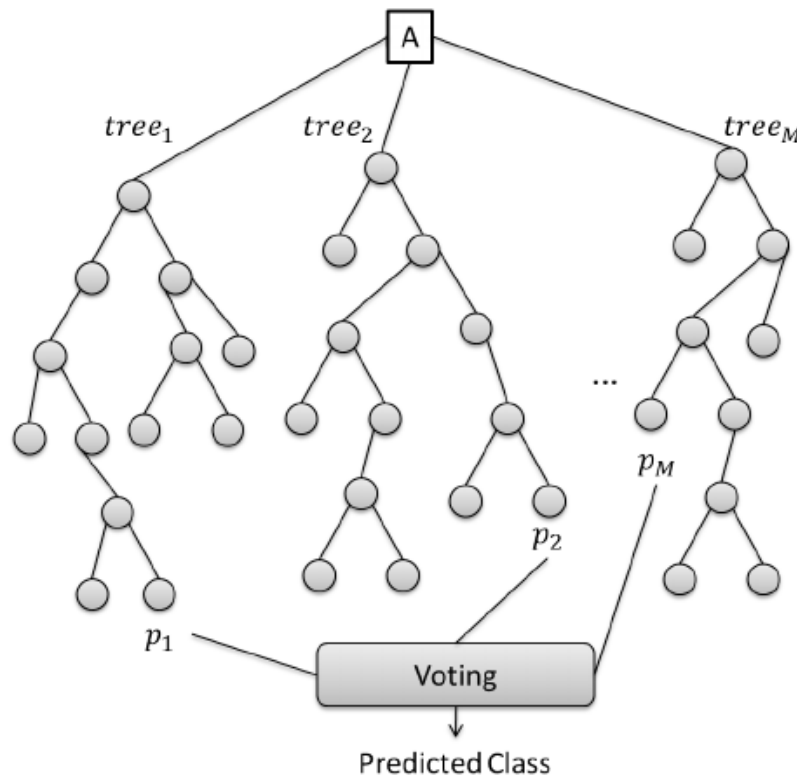


Figure 8. Random Forest for classification. From https://www.researchgate.net/figure/Voting-in-Random-Forest_fig2_272815108

In RandomForest, each tree is built with subset of randomly chosen features (*feature bagging*), not all of them. This minimizes the effect of potential outliers and biased features. Typically, for a classification problem with n features originally, $\sqrt{n}$ features are used in each split [8]. For regression problems, number of features recommended is $n/3$.

The main disadvantage of Random Forest is that it has a high computational cost, both in terms of memory and performance. Nevertheless, Random Forest works very well with real-world data, which is normally noisy and contains many missing values. It is also able to take the most of categorical or semi-continuous values. These might be some of the reasons why Random Forest performed that well for our problem.

The standard RF model has two basic parameters: the number of trees of the forest and the number of attributes to considered in each node. In our case, *Scikit-learn* allow us to tune the followings* [21]:

- **n_estimators** (default to 100): number of trees in the forest.
- **criterion** (default to *gini*): function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.
- **max_features** (default to square root): number of features to consider when looking for the best split. Supported values: *n_features*, *square_root(n_features)*, *log2(n_features)*.
- **max_depth**: maximum depth of the tree. If no value given, then nodes are expanded until all leaves are pure or until all leaves contain less than *min_samples_split* samples.
- **min_samples_leaf** (default to 1): minimum number of samples required to be at a leaf node. This may have the effect of smoothing the model, especially in regression.

- **max_weigth_fraction_leaf** (default to 0.0): minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when `sample_weight` is not provided.
- **max_leaf_nodes**: Grow trees with *max_leaf_nodes* in best-first fashion. Best nodes are defined as relative reduction in impurity. If no value is given, then unlimited number of leaf nodes.
- **min_impurity_decrease** (default to 0.0): node will be split if this split induces a decrease of the impurity greater than or equal to this value.

*Note that there are more parameters available, but these are the most relevant for the algorithm.

2.5 Implementation

After the design step, we proceded with the implementation of the workflow. We followed the steps as stated in *Figure 6*. These secuencial steps were implemented as a pipeline with the purpose of facilitating the testing of different filters and algorithms. The main steps of the pipeline would be the following:

1. Map orthologs to corresponding taxa.
2. Assign GO terms to proteins.
3. Train Machine Learning method.
4. Evaluate mode's performance.

Within the following sections we detail how we implemented each of these steps and how we decided the best model.

2.5.1 Data preprocessing

The first point is to define the structure to represent a Phylogenetic Profile. We decided to use the counts of the orthologs found for each protein in each reference species, instead of a just the presence/absence (binary vector). Due to gene duplications occurred in one lineage, one protein can have multiple co-orthologs within a given species and the number of co-orthologs may provide some extra information to the ML model.

For the creation of the Phylogenetic Profiling matrix, we used as protein samples the proteins set by Quest for Orthologs for the reference human proteome. Not all of them mapped with the available human proteome in MetaPhOrs database (in the 2016 release). In total, 18821 human proteins were found, which was around 90% of coverage with QfO reference human proteome. High enough to perform the analysis. After creating the Phylogenetic Profiling matrix, we assigned GO terms to each protein. Of course, not of all of them have a GO term associated, this would be the final goal of this kind of algorithms, to predict the function for these uncharacterized proteins. However, our main concern now is not to predict the function of these proteins but to evaluate if the Phylogenetic Profiles provide enough information to help in this objective. For this, we should use only those proteins that already have at least one GO term assigned, so we then can compare this GO term to one that the model had predicted. Further, we decided that not all GO terms are valid to be assigned. They must meet the following conditions:

- They must belong to the **Biological Process** ontology.
- The GO term must be supported by one of the following evidence codes: Experimental (EXP, IDA, IPI, IMP, IGI and IEP), Author statements (TAS y NAS) and others that are known to be high reliable (ISS, RCA and IC).

In addition to the explicit GO terms associated with each protein, we experimentally proved that assigning the whole lineage of implicit GO terms helped the ML algorithm to better find patterns within the data. With implicit GO terms we refer to the relation that exists between GO terms and their parents in the ontology. When a GO term is assigned to a protein, each parent of that GO term is also implicitly assigned to the same gene (this is due to the directed acyclic nature of ontologies). Thus, we could follow the path from any GO term to its root (i.e., the lineage), which in this case will be always the GO term *GO:0008150* (“biological process”).

A snippet of the resulting matrix of this step can be seen in *Table 1*. This matrix is then used as input for the ML algorithm. Before stepping into the ML model training, the proteins without any GO term associated are dropped out from the matrix. Further, those GO terms that are not present in at least 100 proteins or present in more than 5000 proteins, are removed from the matrix, as suggested in related works [17]. Too common GO terms tend to have low information content, which may raise the problem of giving good results without really learning anything, but just guessing this too repetitive GO term (like a dummy classifier does) would result in a high accuracy. Too rare GO terms, on the other hand, does not provide enough information to the prediction, as they are not present in enough samples for the ML model to extract a pattern from it. In contrary, they may add noise to the learning process.

Table 1. Preview of the Phylogenetic Profiling matrix (with GO terms assigned) for the Human proteome.

Proteins	Reference species												GO terms	
	189518	85962	208964	122586	83333	272561	...	374847	515635	330879	224911	7918	GO IDs	
Q96E14	0	0	0	0	0	0	...	0	0	0	0	0	[GO:0005575, GO:0005654, GO:0005829, GO:001660...	
Q13021	0	0	0	0	0	0	...	0	0	0	0	0	[GO:0000139, GO:0005575, GO:0005886, GO:001602...	
Q96H35	0	0	0	0	0	0	...	0	0	0	0	0	[GO:0005575, GO:0005654, GO:0005829, GO:004517...	
A0A0A6YYL6	0	0	0	0	0	0	...	1	0	0	0	0	[]	
P51531	0	0	1	0	0	2	...	0	0	1	0	0	[GO:0000785, GO:0005575, GO:0005622, GO:000563...	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
Q495M3	0	0	0	0	0	0	...	0	0	0	0	0	[GO:0005575, GO:0005886, GO:0016020, GO:003198...	
Q92990	0	0	0	0	0	0	...	0	0	0	0	0	[GO:0000151, GO:0005575, GO:0031461, GO:003146...	
Q9NYV9	0	0	0	0	0	0	...	0	0	0	0	0	[GO:0005575, GO:0005886, GO:0016020, GO:001602...	
Q16625	0	0	0	0	0	0	...	0	0	0	0	0	[GO:0005575, GO:0005765, GO:0005774, GO:000588...	
A0FGR9	0	0	0	0	0	0	...	0	0	0	0	0	[GO:0005575, GO:0005789, GO:0005887, GO:001602...	

18821 rows × 78 columns

Regarding the ML step, we first tried an aggressive approach in which we addressed the problem with multilabel classifiers. Multilabel refers to the multiple class/labels, in this case GO terms, nature of the samples (the proteins in our case). Most of them have not only one but several GO terms associated. Thus, the ML model should be able to predict multiple GO terms for each protein. Unfortunately, this approach did not work. The ML models were not able to find enough patterns within the given data to predict that high number of labels. We understood that there were just too many GO terms to predict: almost 1000 different GO terms.

Then, we tried different approach, which at the end resulted in better performance: instead of training a single model to predict multiple labels, we decided to train multiple models to predict each of the different labels.

2.5.2 Binary classification on Phylogenetic Profiling matrix

A binary classification consists of making predictions on only one class, dividing the data into two groups, those that belong to this class (positive samples) and those that do not belong to this class (negative samples). In our particular case, a class will be a GO term, but it can also be a disease (as in *Chapter 4*) or a determined curated function (see section 2.6.1). The steps we followed in all these cases can be seen in *Figure 9*.

We take the same number of negative samples than of positive samples in order to avoid class imbalance. The negative samples correspond to those proteins that are not among the positive samples list, that is, the rest of Phylogenetic Profiling matrix. After assembling the data matrix, it is important to shuffle the samples (the proteins) so that the negative and positive samples are not ordered, otherwise, when we split the test set, this may get only negative or positive samples, depending on what part of the matrix we had split.

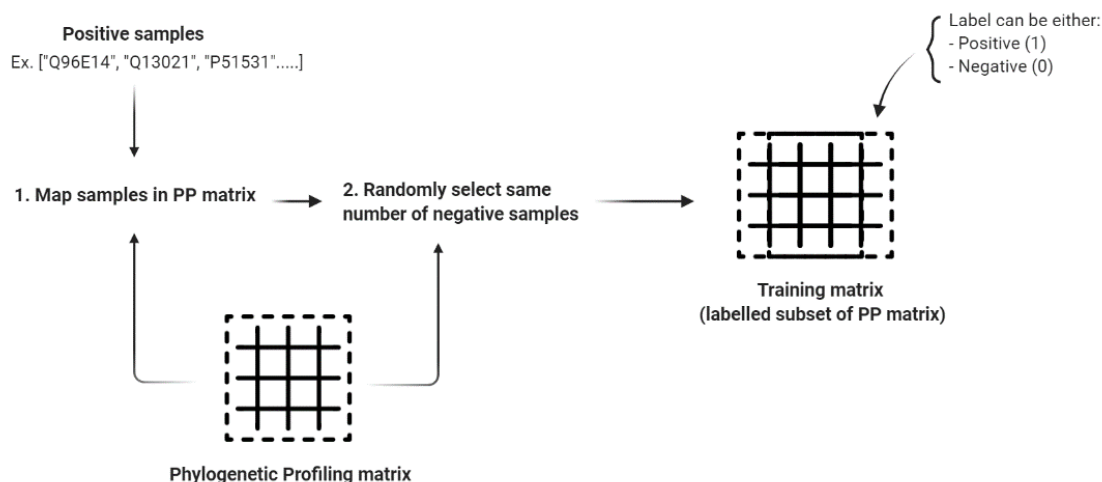


Figure 9. Steps for obtaining the data matrix for the binary classification.

As a result, we will get a data matrix, which basically consists of a labelled subset of the original Phylogenetic Profiling matrix. The 5-fold cross validation will split this matrix into 5 parts and one of them will be use as the test set and the rest as the training set. It is recommendable though, that step 2 is repeated a determined number of times to avoid possible biases within the negative samples (they are chosen randomly but anyway, it is a good practice to repeat and check).

2.5.3 One-Vs-All classification on Phylogenetic Profiling matrix

In One-vs-All (also called One-vs-Rest) classification, for the N possible classes, we have to generate N binary classifier models. This strategy is often used in multiclass classification to adapt classifiers which does not inherently support multilabel classification. *Scikit-learn* provides a built-in method to perform this: *OneVsRestClassifier()*. The disadvantage of it is that it takes as negative samples all the rest of samples within the original matrix, and this ends up in a high imbalance training matrix. Thus, we developed our own OneVsAll method, in which we wrapped our binary classification method and additionally, we allow for a parameter to select the number of resamplings, which we decided to ne set to 5.

2.5.4 Parallelized code

As we worked on HPC environment, everything must be parallelized as much as possible. There are some parts where the difficulties of writing the code in parallel are higher than the time we could save. So, an analysis of which parts to parallelize must be done. Fortunately, *python* facilitates this task by providing some built-in backend libraries to manage and create parallel jobs. These are *multiprocessing* and *joblib*. We used this last, as it was the one that *sklearn* (the Machine Learning library) uses by default. This same library integrates parallelization in a very practical way. In all the models' initialization, it provides an extra parameter called *n_jobs*, with which we can automatically select the level of parallelization we want to train the model. It has 4 different operational modes:

- *n_jobs* = 1 indicates that only one thread will be used (no parallelization).
- *n_jobs* = -1 uses all available threads.
- *n_jobs* = *n* uses “*n*” threads.
- *n_jobs* = -*n* uses all available threads except “*n*” threads.

Thanks to this, it was very easy to parallelize the models training. We assigned -1 to all of them, except for Random Forest. We rapidly realized that Random Forest could not be run in parallel, as it showed a very high rate of memory consumption. Even though, we had 48 cores and 96Gb of RAM available, we usually run into *out of memory errors*. Therefore, we decided to set *n_jobs* to 1 for Random Forest. This parameter was also available for the Cross Validation step, and we did not have any problems at setting it at -1 as well.

On the other hand, apart from the models training and evaluation, the other demanding step of the code was the orthologs mapping. It was not a complex step itself, but it must deal with a lot of data. As before, we used the *joblib* library to parallelize it, but this time we had to manually manage the tasks. In *Algorithm 2*, we can see the pseudocode of the method SearchOrthologs that will be running in parallel.

Algorithm 2: SearchOrthologs. Search and count the number of orthologs found for a certain specie.

```

Input : specie[1], proteinsToSearch[N], listOfOrthologs[M]
Output: list of number of orthologs found for each protein [N]
1 filteredOrthologsList  $\leftarrow$  filterOrthologsListBySpecie(specie, listOfOrthologs)
2 OrthologsCount  $\leftarrow$  list of size N
3 if size(filteredOrthologsList)  $\neq$  0 then
4   for i  $\leftarrow$  0 to size(proteinsToSearch) do
5     foundOrthologs  $\leftarrow$  searchInFilteredOrthologsList(proteinsToSearch[i])
6     orthologsCounts[i].add(foundOrthologs)
7   end
8 end
9 return orthologsCounts

```

We decided to parallelize by species, that is, we created as many tasks as species we had, which in our case were 78. Each task will take as input a certain specie, the list of proteins to search, and the list orthologs. The only difference between tasks is the specie. Therefore, there will be queued 78 tasks and each of this will be in charge of searching orthologs for a given specie. We assigned all the cores available to run this queue of jobs. This considerably reduced the runtime, from hours to just a few minutes.

2.6 Results

2.6.1 Classification of “mitochondrial localization” function

We used the function “mitochondrial localization” (from MitoCarta inventory [22]) as a way to validate if a ML classifier is able to efficiently learn from Phylogenetic Profiling matrix information, as well as for comparing and improving the performances of different classification models. We chose this function, as we knew in advance that it has a high phylogenetic signal.

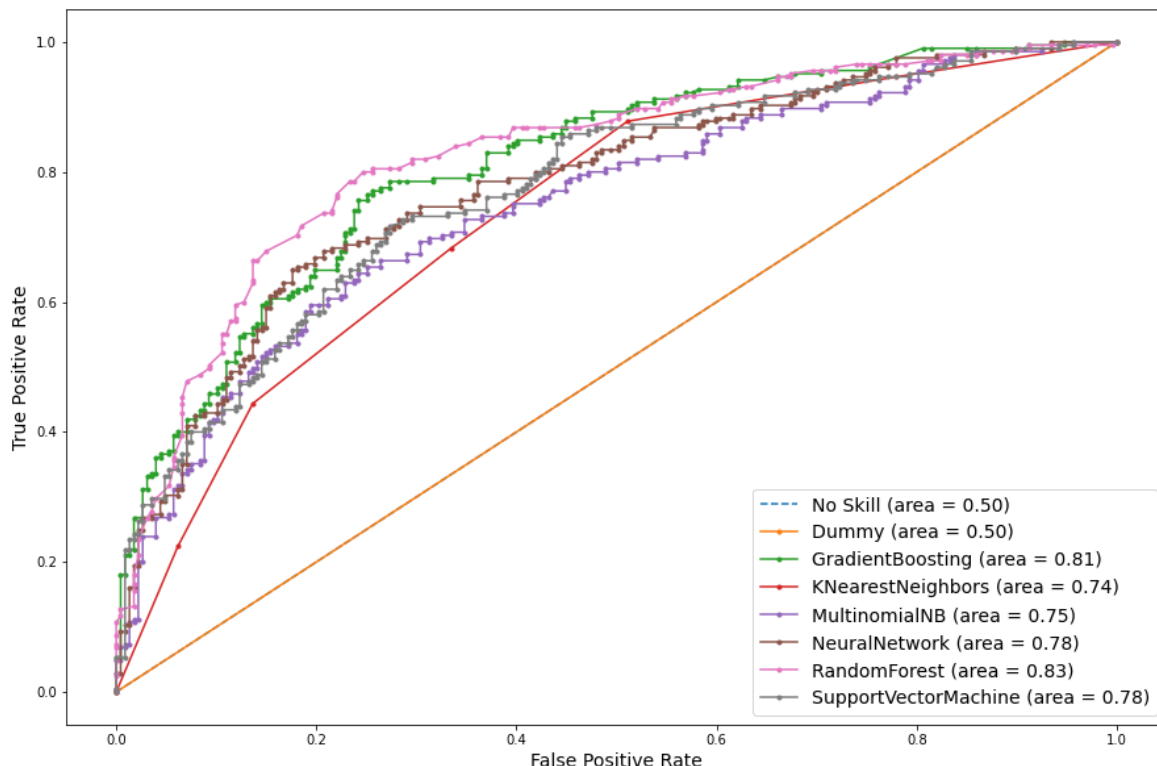


Figure 10. ROC curve of several models at predicting "mitochondrial localization".

For obtaining the ROC curves shown in *Figure 10*, the default parameters of the classifiers were used. If we do not take into account the Dummy Classifier, which showed a non-skill curve (as expected), they all resulted in similar curves. We can notice how Random Forest (in pink) and Gradient Boosting (in green) performed slightly better than the others. K-Nearest Neighbours, on the other hand, showed the worst performing curve, together with MultinomialNB.

With these results we concluded that the models could effectively learn from the phylogenetic profiles. The quality of the classifiers is not equal, as expected, but they all were able to outperform the baseline model (Dummy Classifier). Although we could already discard some of these models, like KNN or Multinomial, we decided not to do in order to validate if the behavior is the same with the case of predicting not only one function, but many (in the form of GO terms).

2.6.2 Classification of GO terms

In this section, we present the results of the classification of a set of 997 different GO terms. Not all the matrix was used for the training set on each GO term, only a subset of it. This depends on how many occurrences of the GO term were found within the Phylogenetic Profiling matrix. First, the GO terms were filtered by number of occurrences, with a minimum of 100 and a maximum of 5000 occurrences. By this, the number of total GO terms

was reduced from 13448 to 997. This includes both explicitly and implicitly (parents' lineage) assigned GO terms. As explained in the methodology section, to prevent class imbalance, the training data was made up of all the “positive” samples and a set of the same number of randomly chosen negative samples. This process was repeated 10 times.

The values of different evaluation metrics shown in *Figure 11* correspond to the mean values of the 5-fold Cross Validation performed by each binary classifier at predicting 997 different GO terms of Biological Process Ontology.

Although the distribution of the values is not shown in the figure, the overall mean of each classifier is considerably close to its median (see the results tables in *Annex B*), meaning that the distribution follows an approximated normal tendency. We can observe many outliers (small black points on the sides of the whiskers) for almost all the classifiers. Most of them correspond to high peaks of performance. This might be related with the phylogenetic signal of some concrete GO terms. The GO term predicted with the best performance in terms of accuracy (a score of 0.91) was *GO:0031424*, which correspond to “keratinization”. This same GO term was the best correctly predicted in terms of F1 score, with a value of 0.90. Both values were obtained by Random Forest Classifier. On the other side we found *GO:0043271* (“negative regulation of ion transport”), which exhibited the worst performance with an accuracy score of 0.47, obtained by a Multinomial Naïve Bayes classifier.

In general, the performance across the different evaluation metrics seems very consistent. Accuracy, the more general metric, moves around 0.6 except for Dummy Classifier, which obtained a considerably lower accuracy of around 0.5. This turned out to be not as low as someone would expect of a classifier that is just “guessing”. However, the F1 score reveals that the Dummy Classifier, in reality, has not learnt anything, as it did not manage to obtain a good recall, neither did precision. This is a clear example of why accuracy is not the best score to measure the performance of classifiers. Dummy Classifier is also useful to demonstrate that the maths behind each of the other “real” classifiers actually works, and its performance can be seen as a base line performance to compare actual learning rate of the others.

Among the other classifiers, although the difference is not really high, Random Forest outperforms all of them in all the metrics analysed. It obtained a mean and median F1 score of 0.613 and 0.607, respectively. The second-best performing classifier was Gradient Boosting, with a mean f1 score of 0.602 and a median f1 score of 0.595. On other side we have the K-Nearest Neighbours classifier, which ranked as the worst performing classifier (without considering Dummy Classifier, of course). Though, the difference is not too large. It obtained the following F1 scores: mean of 0.57 and median of 0.566. Note that we are using the F1 score to compare between the classifiers as, in my opinion, it is the clearer and more descriptive metric for this purpose. ROC-AUC scores, in *Figure 11C*, also confirmed this trend, with slightly higher values, but keeping the same ranking as F1 score and accuracy.

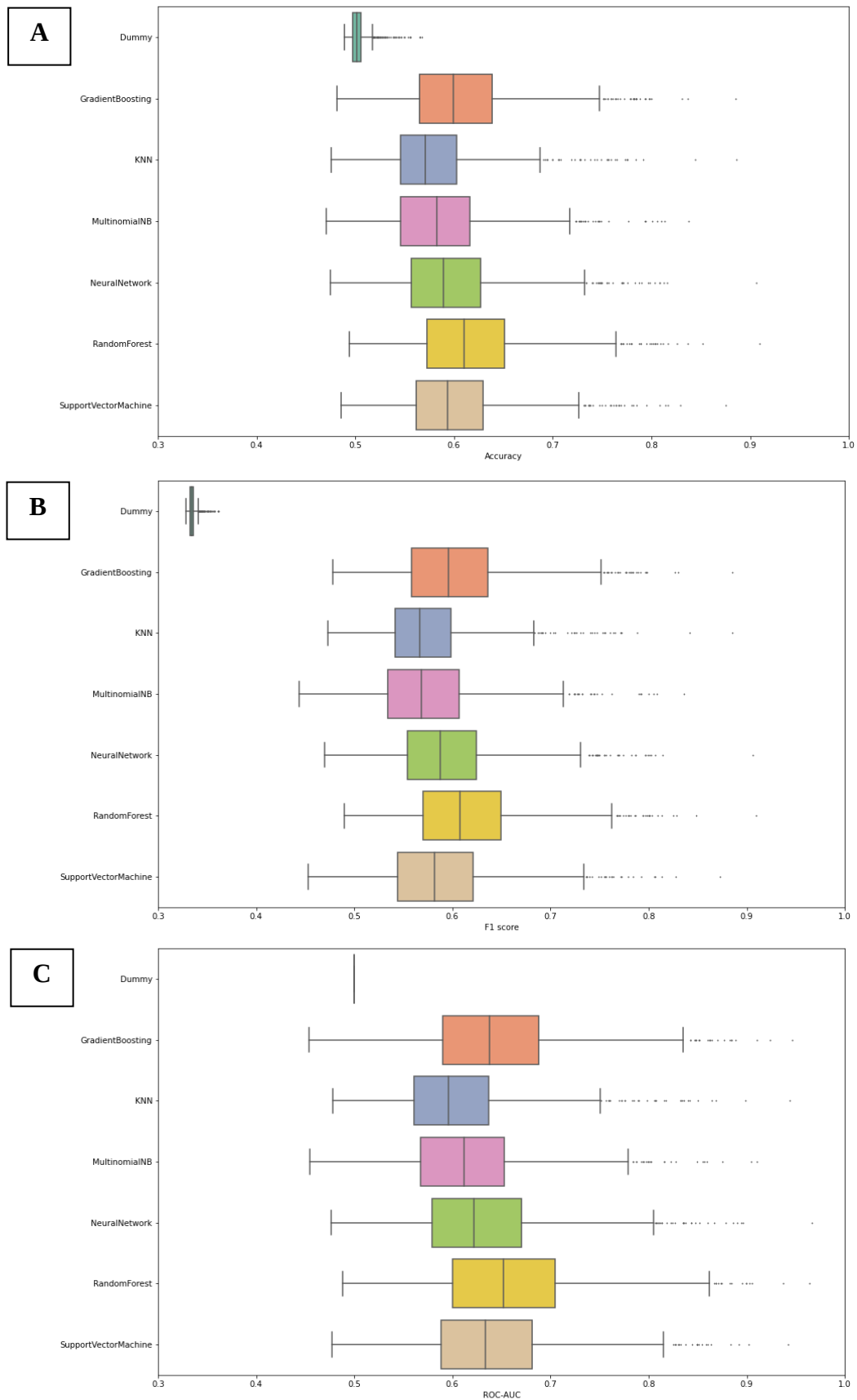


Figure 11. Performance of seven ML classifiers at predicting 997 different Biological Process GO terms based on previous Phylogenetic Profile matrix. **(A)** Boxplot of accuracy scores. **(B)** Boxplot of F1 scores. **(C)** Boxplot of ROC-AUC scores.

2.6.3 Parameter optimization of RandomForest

Once we validated that Random Forest was the best performing classifier with our data, we went on with the following step in a model selection process: parameter optimization. This consists of selecting the best parameters to have a good enough performance (which can be any evaluation metric such as ROC-AUC, accuracy, f1, etc.). The parameters to modify will depend exclusively on the learning method we have selected, in this case, Random Forest.

Scikit-learn provides a great variety of parameters to customize in Random Forest, such as the number of trees, the features-selection function, the split criterion, or the maximum depth of the trees. By default (in the 0.22 version), it takes a series of carefully selected parameters. For instance, the number of trees is set to 100, the criterion to “gini” and the features are selected by a square root function. We decided to tune these 3 parameters, as they are known to be the more relevant for its performance (see section 2.4.3.2).

We used a *Scikit-learn* method called *GridSearchCV* for this task, and it basically consists of a tool that automatically combines the given parameters to each other, to then train a model and finally output which was the best combination (given a metric score). The predictions are obtained over a 5-fold Cross Validation analysis. It also provides a table with the ranking of all the combinations and their results.

Table 2. Comparison of the performances by F1 score of the different parameters selected.

Ranking	Split criterion	Features-selection function	Max trees	F1 score
1°	entropy	log2	300	0.7687 ± 0.0181
2°	entropy	sqrt	500	0.7686 ± 0.0323
3°	gini	log2	300	0.7676 ± 0.0290
4°	entropy	log2	500	0.7674 ± 0.0222
5°	gini	log2	500	0.7674 ± 0.0314
6°	gini	sqrt	100	0.7674 ± 0.0250
7°	gini	sqrt	300	0.7668 ± 0.0264
8°	gini	sqrt	500	0.7644 ± 0.0288
9°	entropy	sqrt	300	0.7641 ± 0.0281
10°	entropy	sqrt	100	0.7618 ± 0.0272
11°	entropy	log2	100	0.7600 ± 0.0251
12°	gini	square	100	0.7587 ± 0.0158

For this analysis we used again the “mitochondrial localization” dataset. This time, we only trained a Random Forest model but with a set of the combinations of the following parameters:

- Split criterion: gini and entropy.
- Max trees: 100, 300 and 500.
- Features-selection function: square root and log2.

In total, there were 12 different combinations. The performance in terms of F1 score is given in *Table 2*. We also computed a ROC curve comparing the best performing parameters, the worst and the default parameters (see *Figure 12*). The best performing configuration was: entropy criterion, 300 max trees and log2 function. The worst performing combination coincided with the default parameters: gini criterion, 100 max trees and square root function. However, the differences were insignificant (as also the ROC curve shows). The improvement corresponds to just 1 hundredth with respect to the default parameters. Anyway, we will use the best parameters in future trainings.

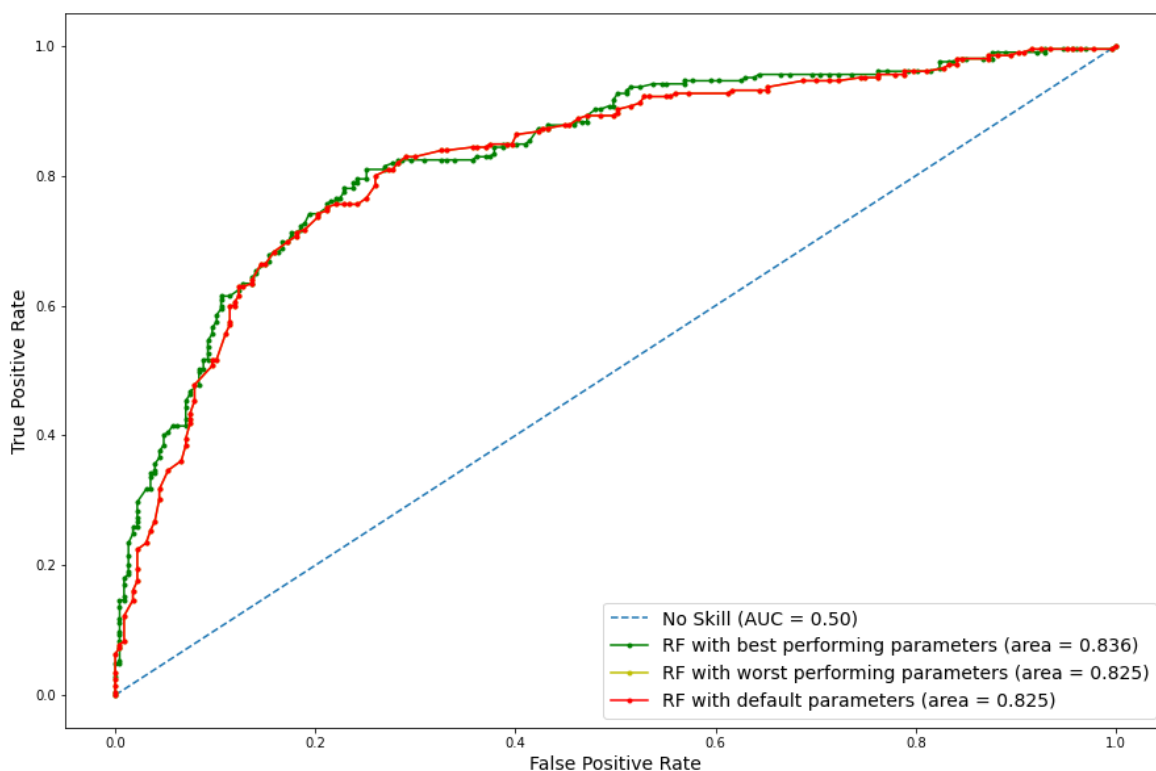


Figure 12. ROC curve comparing the best, worst and default parameters for Random Forest classifier at prediction “mitochondrial localization”.

CHAPTER 3.

CLINICAL APPLICATION

In this chapter we will evaluate the possibility of applying the best ML algorithm found in the previous chapter in a clinical application. Therefore, we will focus on the Random Forest method for building a classifier using phylogenetic profiling data.

3.1 Introduction

The concept that some disease could be inherited by parents was always present, but only after the discovery of DNA as the genetic material, the research about molecular causes of diseases started. Since then, our knowledge of the role of genetics in human diseases has increased exponentially. As in other fields, computational methods have helped to speed up innovation and leverage experimental analysis into large-scale.

Here we introduce an approach that makes use of phylogenetic profiling foundation and Machine Learning potential in order to search for possible relationships between proteins that allow us to further exploit therapeutic targets.

As demonstrated in the previous chapter, Random Forest was proved to be the best performing algorithm to predict in function in the form of GO terms. We want to further take this analysis to predict disease-related genes instead to function. Disease-related genes used to be functionally linked, and thus we could exploit this pattern by the use phylogenetic profiling and then to further predict potential disease-related genes with Machine Learning.

3.2 Methodology

For this use case, we remove the GO terms from the data matrix. That means, the GO terms column is eliminated from the Phylogenetic Profiling matrix, as now we will not be using the assigned GO terms as labels for the classification. Instead, we will use a given set of genes, specifically, disease genes.

The data of the genes has been obtained from the DisGeNET database (see section 2.4.1). Once downloaded the dataset of disease genes from DisGeNET, these were filtered by a minimum threshold of level of evidence called GDA (Gene-Disease Associations) score [16]. Thus, the number of genes is shrunk to only those that are known, with a minimum level of confidence, to be related with the determined disease. The profiles of these genes are searched within the Phylogenetic Profiling matrix. Each element of the matrix receives as label the corresponding gene identifier. If the profile of a gene is not found, it is just ignored and not used (most of them are found).

The methodology we followed was very similar to the previous chapter (2.5.2). A diagram of the steps followed to obtain the training matrix can be seen in *Figure 9*. The

number of negative or non-disease genes were the same as of positive or disease genes and they were chosen randomly among the rest of human phylogenetic profiling matrix (see size of the training dataset in *Table 3*). This process, called resampling, was repeated 10 times, so that we avoid possible biases within the negative sample's dataset. In addition, each time the evaluation metrics were obtained by a 5-fold Cross Validation analysis, which also eliminates the possibility of biases caused by test datasets. The code for this analysis can be seen on the project's repository (see *Annex A*), in the notebook called *4_Machine_Learning.ipynb*.

Table 3. Datasets descriptions for the 3 diseases retrieved from DisGeNET.

	Alzheimer	Parkinson	Autism Spectrum Disorder (ASD)
Disease Id	C0002395	C0030567	C1510586
Total disease genes	3397	2078	1071
Diseases genes for GDA score > 0.2	136	108	115
Training dataset size (negative samples+positive samples)	130+130	106+106	106+106

Three different diseases were chosen for the analysis: Alzheimer, Parkinson, and Autism Spectrum Disorder (ASD). They were not randomly chosen. We wanted to study diseases with a high phylogenetic signal, and neurodegenerative diseases are found among these. Also, we were looking for diseases with a high number of already identified disease genes as well as a solid evidence level. DisGeNET [16] provides a level of evidence score, called GDA score²⁶, which ranges from 0 to 1, being 1 the maximum level of evidence. We chose a GDA threshold of 0.2 because this gave us a sufficient number of genes to analyse as well as considerable level of evidence. We would like to have set a higher threshold but then the number of genes decreased drastically.

3.3 Results

A numerical description of the data is given in *Table 3*. The disease with more genes is Alzheimer, with a final training set size of 260 elements. The other two datasets have 212 elements but with different number of total and disease genes.

²⁶ **GDA score:** Genetic-Disease Association score. More information at <https://www.disgenet.org/dbinfo>

Table 4. Mean prediction performances of Random Forest classifier at predicting 3 different diseases based on phylogenetic profiles. Scores obtained by performing 5-fold Cross Validation 10 times, resampling the negative samples on each.

	Alzheimer	Parkinson	Autism Spectrum Disorder (ASD)
Accuracy	0.571 $\pm$ 0.038	0.565 $\pm$ 0.039	0.595 $\pm$ 0.039
F1 Score	0.556 $\pm$ 0.041	0.548 $\pm$ 0.039	0.603 $\pm$ 0.037
ROC-AUC	0.595 $\pm$ 0.040	0.598 $\pm$ 0.049	0.643 $\pm$ 0.046

Table 4 shows the values of three quality indicators. The performance values seem very consistent across the different metrics evaluated, as with the with GO terms predictions. The overall results are in a similar range than the ones obtained in the GO terms prediction *Figure 11*). The difference among the three diseases is not very significant, although we can notice that ASD shows a slightly higher performance in comparison with the others.

Something to highlight is that, as can be seen in *Table 3*, after filtering by a minimum GDA score, the number of disease genes decreases considerably. With barely 100 hundred positive samples and another 100 hundred of negative samples (a total training dataset of 200 samples) might be not enough to perform an analysis that highly depends upon the presence/absence of samples in a certain specie (feature). Other thing to take into account is that species belonging to groups such as *Fungi*, *Bacteria* or *Archaea* might not be contributing at all to the learning process of the ML model, as none of the disease genes are present in those organisms. This could be improved in the future.

CHAPTER 4.

DISCUSSION

In this chapter, the conclusions we draw from the results obtained are presented. We will distinguish the two applications made, one about prediction of GO terms and the other for predicting disease-related functions. We will also comment some the future improvements that could be implemented.

4.1 Conclusions

4.1.1 Automatic Function Prediction

The main objective of the project was to assess whether a ML model could effectively learn from the Phylogenetic Profiles or not. I think that we successfully demonstrated that it could. With a mean F1 score of 0.61 for a Random Forest classifier, we can claim that the phylogenetic profiles can be used to differentiate between functions in the form of GO terms using a classifier. Although this performance is still far from what it would be needed for the scientific community, it can certainly provide scientists with another tool to complement existing methods. This can open the door for future studies aiming to combine phylogenetic profiling with other approaches in order to improve the performance of protein function assignment. In addition, it was demonstrated in the last CAFA challenge that ensemble methods that combine predictions from different sources tend to perform very well [23].

However, this approach based on machine learning classifiers may suffer from some caveats. The first, and as in other methods to predict protein function [4], simply predicting GO terms does not provide information about whether the protein performs this function in specific tissues or under specific conditions, which is especially important for the Biological Process Ontology. In other words, although a protein can be involved in multiple biological processes, it does not have to always execute all these functions. In the same way, biological roles are described by sets of GO terms from the hierarchically organised GO domains, but they only correspond to the leaf nodes of the acyclic direct graph that is the ontology. We are not taking into consideration the whole topology, apart from using the lineage of GO terms. Finally, given the structured and multi-label nature of the problem, it supposed a major challenge for traditional supervised machine learning methods.

Regarding the phylogenetic profiling, a consideration that we must keep in mind is that it is highly dependable of what reference organisms are being used. Both the size and the diversity of organisms to use must be carefully selected, since this may considerably affect the quality and relationships between the profiles, as demonstrated in related works [24]. Moreover, the orthologs dataset becomes of highly importance for this use case. A highly reliable orthologs dataset is crucial for the analysis. Meanwhile, the competition between tree-based and graph-bases orthologs detection still remains unsolved. In the same

way, certain species or functions of interest for either industry or research, are very often overrepresented in terms of studies and characterization. This induces some biases over some model species such as *Drosophila melanogaster* or *Caenorhabditis elegans*.

In our particular case, there were some organisms for which many proteins were missing. The 2016 version of MetaPhOrs database was found to have some errors that made some species to incorrectly map many proteins. Fortunately, this problem was not affecting the human proteins (over 90% mapped correctly). The group is already working on a new version of the database in order to solve all these limitations.

In addition, phylogenetic profiling may suffer from what are known as “orphan genes”. It is estimated that 10% to 20% of identified genes in the genomes of many species are “orphans” without detectable homology to any protein with a characterized function [25]. Further, because functional annotations are often drawn from datasets and databases which were originally annotated based on homology, it is possible for incorrect functional annotations to propagate through biological databases indefinitely. Estimates of the mis-annotation using experimentally well-characterized sets of enzyme can range from about 25% to over 60% [26].

Nevertheless, the present work has validated that Machine Learning methods can be successfully combined with phylogenetic profiling in order to infer functional relationships between proteins. It is, therefore, another example on how Machine Learning can play an important role in resolving great Biology problems. All of these is mainly possible thanks to the advances on computational performance, specially in HPC environments, which allow us to deal with high dimensional and large amount of data and algorithms. The next generation of supercomputers are meant to include powerful GPUs, which will facilitate the use of deep learning in this machines and thus, improve the current Machine Learning methods.

4.1.2 Predictions for disease-related genes

The average performance of the disease-related gene predictions was similar to the obtained with the GO terms prediction. Apart from the particular disease related reasons, this may suggest that the phylogenetic profiles can provide information about the functional relationships between disease genes, but it is not enough to reach high rates of accuracy. It may lack some extra information that could be achieved by improving, for instance, the reference organism’s selection or the number of reference proteins. The first can be easy to achieve. We might have to adapt the method to disease in concrete that we want to analyse. That is, if we are addressing human diseases, groups of organisms such as *Bacteria* or *Arquea* will not discard of the analysis as they will not be providing us any useful information. The second involves a more complex discussion, as this will totally depend on the disease we are studying. Also, we would need to be careful at including reference proteins that are homologs to each other. This would end up in making predictions based on homology, which is not the purpose of phylogenetic profiling.

Also, the number of positive disease genes used for the analysis may not be enough for the Machine Learning methods to find patterns underlying the data, as they usually need large amount of information and well curated. Although we filtered by a low level of confidence score (GDA), the number of genes drastically decreased from thousands to barely a hundred genes. This occurred even though we selected well studied diseases. With other less popular diseases or rare diseases, this study may be impossible to carry out. The low size of the training dataset can also lead to misleading results, as the test set is smaller and thus the probabilities that the predictions were correct by luck are higher.

Finally, it is important to highlight that Phylogenetic profiling highly depends on the phylogenetic signature of the disease to study. We selected neurodegenerative diseases, as

we knew in advance that they meet this characteristic. If one would want to perform this same analysis with other diseases, it should be aware of this requirement. The approach could be used to improve other existing methods and help dissect the evolutionary relationships underlying the disease.

4.2 Future improvements

There are some improvements that we could make for this analysis. Some are easy to implement or are already implemented but not used, some others are future improvements that might be a good idea to further investigate.

The first improvements are related with the datasets and databases. We used MetaPhOrs to retrieve the orthologs for *Homo Sapiens*, but we could also try others orthology databases such as PANTHER, OrthoFinder or EnsemblCompara. We used PANTHER orthologs dataset, but we did not continue with the next steps to obtain concluding results. In the same way, the reference organisms used for the analysis were set by the Quest for Orthologs, and thus they are already carefully selected and curated, but we could add or drop organisms by demand depending on the needs of our analysis. For instance, we could try adding more human close organisms in order to assess if this would benefit the learning process of the Machine Learning models.

Although Random Forest showed the best performance at classifying the data we provided, given the multi-label nature of the problem it might be useful to use other Machine Learning methods, namely, deep learning models. Multi-task deep neural networks (MTDNN) has demonstrated to be an effective tool to tackle this problem [19]. We also used neural networks and they did not show a great performance, but that could be simply because of *Scikit-learn* (the python's Machine Learning library we used) implemented a very limited version of neural networks.

Improvements for the distant future are related with the graph-based structure of the GO ontology. If we find a tool that take advantage of the underlying information of entire graph in order to make predictions based on this, it will considerably increase the probabilities for the Machine Learning models to extract patterns from it. In addition, if this hypothetical structure can also add information about the expression and/or the specific tissue the function takes place, the method should definitely improve.

4.3 Self-evaluation

This final bachelor thesis consists of one of the two thesis that I am submitting to earn the double degree in Biotechnology and Computer Engineering. The Biotechnology thesis explains the biology background, while the present thesis focuses more in depth on the methodology and algorithms of the project.

I worked very hard to develop this project. It was challenging, which required to have clear a plenty of concepts of different fields that I was not very familiar with. I did not see this as a limitation but as an opportunity to learn and prove myself. I was already interested in Artificial Intelligence before, and this opportunity was perfect to start off. In addition, the topic combined a high content of both computer science and biology specifications, which let me put into practice all the skills learnt during these last 5 years within the double degree in Biotechnology and Computer Engineering.

From the beginning, I encountered several informatics problems, such as the databases management or out of memory errors. This can be clearly seen in the Jupyter Notebooks of the project, where I coded different ways to achieve the same objective and finally chose the most efficient/suitable for the given specifications. For this task it was very useful to have covered courses such as Programming, Data Structures, Parallel and Massive Computing

and of course, Artificial Intelligence. At the same time, I had to deal with a set of biology concepts such as “GO terms” or “orthology”. Some of these concepts were already covered by Biotechnology courses but others required me to make a research in online resources and available literature. After reading several papers and making my own trials, I finally fully understood all the concepts related to the project. This, indeed, took a lot of time and effort.

Regarding my stay in the group, I am glad to say that it was very worthy. My online stay within the Comparative Genomics Group lasted from January to May of this year. Although the stay was full online (due to COVID-19 prevention) and thus the communication sometimes could not be as fluid as in presence, I was able to learn a lot from the group and they highly contributed to improve my bioinformatics skills. An online stay implies to be more active and self-learning, but after almost one year with remote classes and work, I think this is a skill that we all are used to.

This internship combined with the degree’s work required me to be very focused and organized. Fortunately, remote work also has advantages and one of them was that I did not need to move from one place to another. I could deal with everything from the commodity of home.

Finally, being able to independently use such a powerful machine like MareNostrum4 gave me a lot of confidence. I realized, by experience, how computer science and biology can jointly work to achieve great goals. In fact, today Biology needs computer science more than ever and I am committed to keep contributing to this end.

References

- [1] D480-D489, “UniProt: the universal protein knowledgebase in 2021 The UniProt Consortium,” *Nucleic Acids Res.*, vol. 49, 2021.
- [2] T. Gabaldón, “Comparative genomics-based prediction of protein function,” *Methods Mol. Biol.*, vol. 439, pp. 387–401, 2008.
- [3] E. Callaway, “‘It will change everything’: DeepMind’s AI makes gigantic leap in solving protein structures,” *Nature*, vol. 588, no. 7837. NLM (Medline), pp. 203–204, 01-Dec-2020.
- [4] S. Makrodimitris, R. C. H. J. Van Ham, and M. J. T. Reinders, “Automatic gene function prediction in the 2020’s,” *Genes*, vol. 11, no. 11. MDPI AG, pp. 1–18, 01-Nov-2020.
- [5] T. Gabaldón and E. V Koonin, “Functional and evolutionary implications of gene orthology HHS Public Access,” *Nat Rev Genet*, vol. 14, no. 5, pp. 360–366, 2013.
- [6] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.
- [7] M. Kubat, *An Introduction to Machine Learning*. Springer International Publishing, 2017.
- [8] S. S. Skiena, *The Data Science Design Manual*. Cham: Springer International Publishing, 2017.
- [9] “Demystifying ROC Curves. How to interpret and when to use... | by Ruchi Toshniwal | Towards Data Science.” [Online]. Available: <https://towardsdatascience.com/demystifying-roc-curves-df809474529a>. [Accessed: 09-Jun-2021].
- [10] Y. Tabach *et al.*, “Human disease locus discovery and mapping to molecular pathways through phylogenetic profiling,” *Mol. Syst. Biol.*, vol. 9, no. 1, p. 692, Jan. 2013.
- [11] J. Huerta-Cepas, F. Serra, and P. Bork, “ETE 3: Reconstruction, Analysis, and Visualization of Phylogenomic Data,” *Mol. Biol. Evol.*, vol. 33, no. 6, pp. 1635–1638, Jun. 2016.
- [12] D. V. Klopfenstein *et al.*, “GOATOOLS: A Python library for Gene Ontology analyses,” *Sci. Rep.*, vol. 8, no. 1, pp. 1–17, Dec. 2018.
- [13] “TOP500 List - November 2020 | TOP500.” [Online]. Available: <https://www.top500.org/lists/top500/list/2020/11/?page=1>. [Accessed: 22-May-2021].
- [14] U. Chorostecki, M. Molina, L. P. Prysycz, and T. Gabaldón, “MetaPhOrs 2.0: Integrative, phylogeny-based inference of orthology and paralogy across the tree of life,” *Nucleic Acids Res.*, vol. 48, no. W1, pp. W553–W557, Jul. 2021.
- [15] A. M. Altenhoff *et al.*, “The Quest for Orthologs benchmark service and consensus calls in 2020,” *Nucleic Acids Res.*, vol. 48, no. W1, pp. W538–W545, Jul. 2021.
- [16] J. Piñero *et al.*, “The DisGeNET knowledge platform for disease genomics: 2019 update,” *Nucleic Acids Res.*, vol. 48, no. D1, pp. D845–D855, Jan. 2020.
- [17] X. Dai *et al.*, “Non-homology-based prediction of gene functions,” *bioRxiv*. bioRxiv, p. 730473, 09-Aug-2019.
- [18] T. W. Lin, J. W. Wu, and D. T. H. Chang, “Combining Phylogenetic Profiling-Based and Machine Learning-Based Techniques to Predict Functional Related Proteins,” *PLoS One*, vol. 8, no. 9, Sep. 2013.
- [19] R. Fa, D. Cozzetto, C. Wan, and D. T. Jones, “Predicting human protein function with multitask deep neural networks,” *PLoS One*, vol. 13, no. 6, p. e0198216, Jun. 2018.
- [20] J. Frery, A. Habrard, M. Sebban, O. Caelen, and L. He-Guelton, “Efficient Top Rank Optimization with Gradient Boosting for Supervised Anomaly Detection,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2017, vol. 10534 LNAI, pp. 20–35.
- [21] “sklearn.ensemble.RandomForestClassifier — scikit-learn 0.24.2 documentation.” [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>. [Accessed: 08-Jun-2021].
- [22] S. Rath *et al.*, “MitoCarta3.0: An updated mitochondrial proteome now with sub-organelle localization and pathway annotations,” *Nucleic Acids Res.*, vol. 49, no. D1, pp. D1541–

- D1547, Jan. 2021.
- [23] N. Zhou *et al.*, “The CAFA challenge reports improved protein function prediction and new functional annotations for hundreds of genes through experimental screens,” *Genome Biol.*, vol. 20, no. 1, Nov. 2019.
 - [24] N. Škunca and C. Dessimoz, “Phylogenetic profiling: How much input data is enough?,” *PLoS One*, vol. 10, no. 2, p. e0114701, Feb. 2015.
 - [25] K. Khalturin, G. Hemmrich, S. Fraune, R. Augustin, and T. C. G. Bosch, “More than just orphans: are taxonomically-restricted genes important in evolution?,” *Trends in Genetics*, vol. 25, no. 9. Trends Genet, pp. 404–413, Sep-2009.
 - [26] A. M. Schnoes, S. D. Brown, I. Dodevski, and P. C. Babbitt, “Annotation error in public databases: Misannotation of molecular function in enzyme superfamilies,” *PLoS Comput. Biol.*, vol. 5, no. 12, p. 1000605, Dec. 2009.

ANNEX A: DISPONIBILITY OF THE CODE

All the code that made this work possible can be found and is freely accessible under a **GNU General Public License** on my personal GitHub website. Every line of code, unless explicitly indicated, is my own. The link to the specific repository of the project is the following:

<https://github.com/geovalexis/PP2GO>

The repository is organised as follows*:

- **data/**: various data needed for the pipeline to run. They are not enough to run the pipeline. GitHub strictly limits the size of individual files to 100MB, so some files such as the GOA annotations or `taxid_mapping` could not be uploaded. These must be downloaded manually. See the instructions on *1_Data_Retrieval.ipynb*.
- **orthologs/**: folder with the orthologs datasets from different sources. As before, the size limitation does not allow us to upload the entire MetaPhOrs' orthologs dataset but only a subset of it, which can be used for testing purposes.
- **notebooks/**: jupyter notebooks (powered by Google Colab) used as a testing/debugging environment for the algorithms. It also includes the implementation of all the plots shown in the work as well as most of the data analysis performed to achieve the results.
- **src/**: source code of the pipeline used to test different filters and approaches. It was the code optimized to be run in the MareNostrum4.
- **pp2go_env.yml**: pipeline's package requirements to install with Anaconda.

*The folders organization and content are subject to future changes.

ANNEX B: RESULTS TABLES FOR GO TERMS PREDICTIONS

The following tables correspond the mean and median results for the prediction of 997 different GO terms of the Biological Process Ontology. Obtained on date: April 6th, 2021.

- Mean results:

	Dummy	Random Forest	KNN	Neural Network	Support Vector Machine	GradientBoosting	MultinomialNB
Mean accuracy	0.503207	0.616134	0.579263	0.596403	0.599325	0.606504	0.586492
Mean accuracy's standard deviation	0.002221	0.020348	0.019796	0.021561	0.019167	0.021819	0.018545
Mean F1 score	0.334722	0.613624	0.574678	0.593952	0.588134	0.602449	0.574980
Mean F1 score's standard deviation	0.000983	0.020559	0.020256	0.021812	0.020911	0.022196	0.019746
Mean ROC-AUC	0.500000	0.657759	0.605299	0.629603	0.639038	0.644408	0.616162
Mean ROC-AUC's standard deviation	0.000000	0.022717	0.022316	0.024799	0.022097	0.025306	0.021847

- Median results:

	Dummy	Random Forest	KNN	Neural Network	Support Vector Machine	GradientBoosting	MultinomialNB
Mean accuracy	0.501359	0.610248	0.570840	0.589411	0.593322	0.599316	0.582551
Mean accuracy's standard deviation	0.001651	0.018600	0.018145	0.019713	0.017490	0.019176	0.016184
Mean F1 score	0.333932	0.607356	0.566502	0.587332	0.581564	0.595716	0.568619
Mean F1 score's standard deviation	0.000727	0.018728	0.018445	0.020000	0.018728	0.019657	0.017716
Mean ROC-AUC	0.500000	0.651844	0.596071	0.621644	0.633764	0.638018	0.611719
Mean ROC-AUC's standard deviation	0.000000	0.020375	0.020826	0.022693	0.019678	0.023286	0.019441

ANNEX C: JOB SCRIPT RUN IN MARENOSTRUM4

The following script corresponds to a launched job in MareNostrum4. We reused this same script with different SLURM options as well as with different filters, databases and parameters for the pipeline. Most of the time we used 1 node (*ntasks=1*) and 48 cores (*cpu-per-task*) but we also tried different options.

All packages must be loaded (with the dedicated *module load* command) before the execution and we had to make sure that all the data needed for correct execution of the pipeline is located within one of the available storages of the nodes. In my personal case, I had everything within my user's (*bsc40356*) folder in the cluster.

In order to submit this job to the queue, we needed to first connect to one of the login nodes and then executed the script below with a *sbatch* command.

```
#!/bin/bash

#SBATCH --job-name=a4a_f_p3
#SBATCH --workdir=.
#SBATCH --output=logs/all4all.out
#SBATCH --error=logs/all4all_%j.err
#SBATCH --time=20:00:00
#SBATCH --cpus-per-task=48
#SBATCH --ntasks=1

module load gcc/8.1.0 impi/2018.1 mkl/2018.1 opencv/4.1.2 python/3.6.4_ML
export PYTHONPATH=/home/bsc40/bsc40356/site-packages:$PYTHONPATH

python src/pp2go.py
    --orthologs orthologs/MtP_201601_blasted.tab
    --pp-matrix results/MtP_201601_blasted-include_go_parents-
min_gos_100-max_gos_5000- pp_matrix_counts_with_taxa -one_vs_rest.tab
    --ml-results MtP_201601_blasted-include_go_parents-min_gos_100-
max_gos_5000-ML_assessment-one_vs_rest.tab
    --go-aspects P
    --include-go-parents
    --min-gos 100
    --max-gos 5000
-v
```