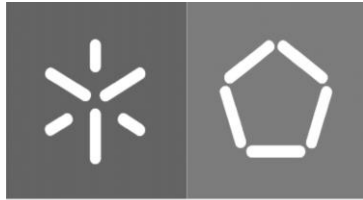




Universidade do Minho
Escola de Engenharia

Daniel Torres Varzim Faria

**Development of a computational approach
for the identification and annotation of
transport proteins**



Universidade do Minho
Escola de Engenharia

Daniel Torres Varzim Faria

**Development of a computational approach
for the identification and annotation of
transport proteins**

Dissertação de Mestrado

Mestrado em Bioinformática

Trabalho efetuado sob a orientação de

Professor Doutor Oscar Dias

Professor Doutor Miguel Rocha

DECLARAÇÃO

Nome: Daniel Torres Varzim Faria

Endereço eletrónico: pg27662@alunos.uminho.pt Telefone:926454154

Cartão do Cidadão: 14325697

Título da dissertação: Development of a computational approach for the identification and annotation of transport proteins

Orientadores:

Professor Doutor Oscar Dias

Professor Doutor Miguel Rocha

Ano de conclusão: 2016

Mestrado em Bioinformática

É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA DISSERTAÇÃO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE.

Universidade do Minho, 31 / 10 / 2016

Assinatura: Daniel Torres Varzim Faria

ACKNOWLEDGEMENTS/AGRADECIMENTOS

Apesar deste trabalho ter sido realizado por mim tal não seria possível sem um grande grupo de pessoas que me apoiou durante todo o ano. Por esse motivo guardo a primeira seção do meu trabalho para agradecer o apoio de todas elas e quero realçar que sem elas não teria conseguido chegar tão longe.

Gostaria de agradecer aos meus orientadores, professor Oscar Dias e professor Miguel Rocha por todo o tempo dispensado para me ajudar e rever o meu trabalho, pelos seus conselhos e pela orientação nas horas em que me sentia mais perdido.

Aos colegas do grupo de investigação de Biosistemas por toda a simpatia e boa disposição demonstrados no local de trabalho, principalmente ao Vítor Vieira, Fábio Ramalho e Filipe Liu por me terem ajudado nos momentos de maior dificuldade.

Um grande obrigado a todos os meus colegas de mestrado, pelo companheirismo, amizade e por todos os momentos de diversão que me proporcionaram ao longo do mestrado que vou guardar para sempre na memória. Queria agradecer também a todos os meus restantes amigos que, apesar de não me terem acompanhado neste percurso, estiveram sempre lá para me apoiar com palavras de apoio e força.

Um agradecimento para toda a minha família especialmente para os meus pais e irmão, por todo o carinho, força e conselhos que me deram ao longo da vida. Queria agradecer principalmente à minha mãe por ser um dos pilares da minha vida e por se ter sacrificado e trabalhado tanto para que eu pudesse estudar e fazer o que realmente gosto.

Quero agradecer também à minha namorada pelo apoio incondicional, pelo carinho, por nunca me deixar sentir em baixo e por sempre me incentivar a lutar pelos meus objetivos. Obrigado por seres um exemplo de perseverança, força e coragem para mim.

A todos, o meu sincero obrigado!

RESUMO

Na última década, dada a evolução nas técnicas de sequenciação de nova geração, o número de genomas sequenciados tem vindo a crescer exponencialmente [1]. A ferramenta *merlin*, desenvolvida pelo grupo de investigação Biosystems (Universidade do Minho) é uma ferramenta capaz de gerar modelos metabólicos à escala genómica. A identificação de genes que codificam proteínas transportadoras e os metabolitos transportados por estas são tarefas essenciais para o desenvolvimento de modelos metabólicos à escala genómica mais robustos e precisos.

Para este trabalho foram treinados e testados sete modelos de aprendizagem máquina diferentes, usando um processo validação cruzada repetido 5 vezes, em conjuntos de dados diferentes, para identificar e classificar proteínas transportadoras. Para provar o valor dos modelos desenvolvidos foram criados quatro conjuntos de dados diferentes compostos por proteínas curadas provenientes das bases de dados TCDB e SwissProt.

Os conjuntos de modelos criados usando vários conjuntos de dados apresentaram um bom desempenho global, com o melhor a atingir 91% de acerto e desvio padrão baixo; o valor de F1-score atinge os 0.90 (+/- 0.00), fazendo destes modelos uma boa solução para a identificação e caracterização de proteínas transportadoras dado um genoma não anotado.

Os modelos usados para identificar proteínas transportadoras apresentaram um maior número de falsos negativos comparado com o número de falsos positivos (quase três vezes maior) o que significa que os níveis de confiança para uma classificação em proteína transportadora são elevados, e que os modelos falham um número ainda significativo de proteínas transportadoras que são incorretamente ignoradas.

Palavras chave: Linguagem máquina; Proteínas transportadoras; Modelos; Caracterização

ABSTRACT

In the last decade, given the evolution of next-generation sequencing techniques, the number of sequenced genomes has grown exponentially [2]. The framework *merlin* [1], developed by the Biosystems research group (University of Minho) is a tool capable of generating genome-scale metabolic models. The identification of genes encoding transport proteins and the metabolites transported by them are essential tasks for the development of more robust and accurate genome-scale metabolic models.

For this work, seven different machine learning models were trained and tested, using a five-fold cross validation process, on different datasets to identify and classify transport proteins. To prove the value of the developed models, four different datasets composed by well annotated proteins from TCDB and SwissProt were used.

Ensembles of the models created using different datasets showed good overall performance with accuracy reaching 91% and low standard error; F1 scores reach 0.90 (+/- 0.00), making them a good solution for the identification and characterization of transport proteins given a new unannotated genome.

The models used to identify transport proteins had a bigger number of false negatives compared to false positives (almost three times bigger) meaning that the confidence level of the classification of a protein as a transporter is high, and that these models miss a relevant number of transporter proteins that misclassified.

Keywords: Machine Learning; Transport Proteins; Models; Characterization

INDEX

Acknowledgements/Agradecimientos	iii
Resumo	v
Abstract	vii
List of Figures	xi
List of Tables.....	xiii
List of Abbreviations and Acronyms	xv
1. Introduction.....	1
1.1 Motivation	1
1.2 Objectives	2
1.3 Structure of the document.....	2
2. State of the art	5
2.1 Machine Learning concepts and definitions.....	5
2.2 Sequence analysis algorithms and tools	17
2.3 Relevant bioinformatics tools and databases.....	20
2.4 Relevant development environments.....	25
3. Methods.....	27
3.1 Data.....	27
3.2 Input attributes and output attributes	27
3.3 Data sets.....	30
3.4 Dataset pre-processing.....	31
3.5 Feature selection	32
3.6 Models	32
3.7 Ensemble Methods	33
3.8 Cross-validation.....	34
3.9 Performance evaluation	34
4. Development	35
4.1 Code Developed	35
4.2 Workflow.....	45
5. Results and Discussion	47
5.1 Case studies	47

5.1.1	Case study: Transport and Transport related protein models performance	47
5.1.2	Case study: Influence of negative cases in model performance.....	54
5.1.3	Case study: Transport protein models performance.....	55
5.1.4	Case study: Transport proteins characterization models performance.....	56
6.	Conclusions and further work.....	65
	Bibliography.....	68
	Attachment I.....	72
	Attachment II.....	76

LIST OF FIGURES

Figure 1- Seven step process for developing a machine learning algorithm	7
Figure 2- Calculation of the probability of classifying as “Yes” or “No” in the example “X1, Y2”	8
Figure 3- Voting classifier prediction method (Hard Vote classifier’s final prediction is the class predicted by most of the models, while weighted voting classifiers considers that each model has a weight given by the user. When making the final prediction, models with bigger weight have more influence in the final result.	33
Figure 4- "TCDB_ProteinType_Division" generated files	35
Figure 5- "TCDB_ProteinType_Division2" example of the generated files	35
Figure 6- Pseudo code for the creation of the amino acid occurrence and composition features	37
Figure 7- Pseudo code for the creation of the amino acid physicochemical occurrence and composition features	38
Figure 8- Pseudo code for the creation of the dipeptide composition feature.....	38
Figure 9- Pseudo code for the creation of the number of transporter related pfam domains feature	39
Figure 10- Pseudo code for the creation of the single transporter related Pfam domains feature	40
Figure 11- Pseudo code for the creation of the number of alpha helices and signal peptide features	40
Figure 12- Pseudo code for the creation of the beta barrel feature	41
Figure 13- Pseudo code for the location prediction feature	41
Figure 14- Pseudo code for the creation of the out attributes “Is Transporter” and “TCDB_ID”	42
Figure 15- Pseudo code used for mixing the final dataset and splitting it’s in and out attributes	43
Figure 16- Pseudo code for the data transformation, feature selection and model training and testing	44
Figure 17- Pseudo code for the ensemble methods “VoteClassifier” with hard voting and weighted hard voting	44

Figure 18-Pseudo code for the prediction method. Predicts if a protein is a transporter protein and if it is, predicts the first level of the TC system from 1 to 5..... 45

Figure 19- Workflow of the developed algorithm 45

LIST OF TABLES

Table 1-Table containing the number of times a class occurs given a certain feature.....	8
Table 2-Confusion matrix example.....	13
Table 3-Different Models used by some Localization prediction tools and their respective websites.	21
Table 4- Overview of the objective and models used in some Membrane protein topology tools and their respective websites.....	23
Table 5- Dataset1 content.....	31
Table 6- Dataset2 content.....	31
Table 7- Dataset3 content.....	31
Table 8- Dataset4 content.....	31
Table 9- Mean of PECC, F1 and ROC-AUC scores after a 5-fold cross validation process using Dataset1 without filters.	48
Table 10- Mean of PECC, F1 and ROC-AUC scores after a 5-fold cross validation process using a variant of Dataset1 with variance threshold filter.....	49
Table 11- Mean of PECC, F1 and ROC-AUC scores after a 5-fold cross validation process using a variant of Dataset1 with StandardScaler().	49
Table 12- Mean of PECC, F1 and ROC-AUC scores after a 5-fold cross validation process using a variant of Dataset1 with MaxAbsScaler().	50
Table 13- Mean of PECC, F1 and ROC-AUC scores after a 5-fold cross validation process using a variant of Dataset1 with a Variance threshold followed by a RFE filter.....	51
Table 14- Mean of PECC, F1 and ROC-AUC scores after a 5-fold cross validation process using a variant of Dataset1 with the best pre-processing parameters.	51
Table 15- Mean performance of the ensemble methods: “VotingClassifier” with and without weights on the models using a variant of Dataset1.	52
Table 16-Confusion Matrix for all models after a cross-validation process with training dataset representing 80% of the dataset1 and the test dataset the other 20%.....	53
Table 17- Mean of PECC, F1 and ROC-AUC scores after a 5-fold cross validation process using a variant of Dataset2.	55

Table 18- Mean of PECC, F1 and ROC-AUC scores after a 5-fold cross validation process using a variant of dataset12345.....	55
Table 19- Mean performance of the ensemble methods: “VotingClassifier” with and without weights on the models.....	56
Table 20-- Mean of PECC scores after a 5-fold cross validation process using a variation of the transporter dataset.	57
Table 21-- Mean performance of the ensemble methods: “VotingClassifier” with and without weights on the models.....	57
Table 22- Confusion Matrix for the NB model after a cross-validation process with training dataset representing 80% of the transporter dataset and the test dataset the other 20%.	58
Table 23- Confusion Matrix for the ET model after a cross-validation process with training dataset representing 80% of the transporter dataset and the test dataset the other 20%.	58
Table 24- Confusion Matrix for the KNN model after a cross-validation process with training dataset representing 80% of the transporter dataset and the test dataset the other 20%.	59
Table 25- Confusion Matrix for the LR model after a cross-validation process with training dataset representing 80% of the transporter dataset and the test dataset the other 20%.	59
Table 26- Confusion Matrix for the GB model after a cross-validation process with training dataset representing 80% of the transporter dataset and the test dataset the other 20%.	60
Table 27-Confusion Matrix for the RF model after a cross-validation process with training dataset representing 80% of the transporter dataset and the test dataset the other 20%.	60
Table 28- Confusion Matrix for the SVM model after a cross-validation process with training dataset representing 80% of the transporter dataset and the test dataset the other 20%.	61
Table 29-Table of sensibility and Specificity for each class predicted by every model.....	62

LIST OF ABBREVIATIONS AND ACRONYMS

AA- Amino acid

ET- ExtraTreeClassifier

GB- GradientBoostingClassifier

KNN- K -nearest neighbours

LR- Logistic Regression

ML- Machine Learning

NB- Naive Bayes

RF- RandomForest

SVM- Support Vector Machine

1. INTRODUCTION

1.1 Motivation

Genome-scale metabolic models are mathematical representations of living organisms. These models are sets of reactions and constraints, which mimic the behaviour of the organisms when exposed to different environmental and genetic conditions [1]. The identification of genes encoding transport proteins and the metabolites they carry are important tasks for the development of more robust and accurate genome-scale metabolic models [3].

Transporter proteins are polypeptide chains, which promote the transport of several compounds across cell membranes [4]. For instance, carrier proteins control the uptake of nutrients to the cell, thus being important for growth, and provide resistance to drugs, which allow organisms to thrive in severe conditions. These transporters may use several mechanisms to move the compounds, including facilitated diffusion, symport and antiport.

Manually annotated transporter proteins are described and stored in databases, such as the Transporter Classification Database (TCDB) [5]. TCDB is a curated repository for factual information compiled from literature references. It is available in a web-accessible relational database encompassing sequence, classification, structural, functional and evolutionary information for transport proteins from a variety of living organisms.

These proteins have several specific characteristics. For instance, they are usually found in membranes, they contain specific motifs on their tail residues [4], and usually have one or more transmembrane domains (alpha helices or beta barrels) on their sequences.

Transport proteins are usually annotated with reactions for metabolites known to be taken in from the medium, excreted from the cell or transported across intracellular compartments [6]. However, transport proteins are often poorly annotated [7] as this information is usually only retrieved from literature evidences.

In genome-scale metabolic models, transport reactions are usually added manually based on experimental data or literature. Thus, new methods that can automate this task are required [8]. Recently, the Biosystems research group (University of Minho) has published a framework, merlin [1], fully written in Java and that uses a MySQL database, which includes a tool that provides a first approach for tackling this problem. The Transport Reactions Annotation and Generation (TRIAGE) [9] tool identifies the metabolites transported by each transmembrane

protein and its transporter family. The localization of the carriers is also predicted and, consequently, their action is confined to a given membrane. This thesis proposes a different approach to this problem through the development and implementation of several machine learning models that can be used to identify and characterize transport proteins.

1.2 Objectives

The main goal of this work is the development of a machine learning framework which is able to identify and annotate transporter proteins from their amino acid sequence.

In detail, the technological objectives are:

- 1) To review relevant bibliography about the state of the art techniques and existing tools for the annotation of proteins, in general, and, more specifically, transporters; assessment of the main features that characterize transporter proteins;
- 2) Studying and testing available relevant software tools;
- 3) To develop a machine learning tool to annotate transporter proteins, involving:
 - a) Selection of available tools that can be helpful for transporter annotation;
 - b) Development of machine learning methods for identifying transporter proteins, by developing classifiers and applying them to new sequences;
 - c) Development of classifiers to identify proteins able to transport specific metabolites;
- 4) Validation of the tool with gold standard corpora from previously manually annotated sources and if possible in real world scenarios.

1.3 Structure of the document

This document is organized in the following way:

Chapter2

State of the art

Introduction to some machine learning concepts and models as well as some ensemble and feature selection methods and model evaluation processes. Brief presentation of useful bioinformatics tools and databases for the characterization of proteins.

Chapter3

Methods

Overview of the data collected to create the datasets. Description of the used dataset pre-processing, feature selection, models, ensemble methods and statistics to evaluate the models.

Chapter 4

Development

Brief description of the code developed in the thesis with pseudo code of the main scripts.

Chapter 5

Results and Discussion

Presentation of the main results generated in the thesis followed by a discussion of the results.

Chapter 6

Conclusion and Future Work

Main conclusion of the thesis taken from the results obtained, followed by a description of the possible improvements and future work.

2. STATE OF THE ART

2.1 Machine Learning concepts and definitions

One of the differences between a computer and a human being is that, when facing a problem, humans tend to try to improve the way they solve it, while computers only execute procedures supplied to them. Although computers may be very effective in solving problems they lack the ability of self-improvement with experience [10]. Machine Learning (ML) is a branch of artificial intelligence/ computer science/ statistics that provides computers with the ability of predicting the outcome of an event or situation, and eventually improving its results with time, simulating the gain of experience [11].

Tom Mitchell defines ML as “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E” [12].

In order to help the reader to understand the concept of ML, the next paragraphs contain a summary of some ML concepts and definitions [11][13][14].

Attribute. An attribute represents a feature that describes instances. Attributes can be categorical or continuous. Categorical attributes take values from a set with a finite number of discrete values and can either be nominal, indicating that there is no order between the values e.g. names and colours, or ordinal e.g. little, medium, big, where an order can be identified. Continuous attributes take values from a domain which is a subset of real numbers and can take any value within a range e.g. height, time [14].

- Attribute values. For example, if “names” is an attribute, “John” is a value of the attribute “names”.
- Input and output attributes. Input attributes are the attributes that are going to be used to make a prediction of the output attribute.

Instance. Instance (or example) is an object composed by a set of input attributes (and possibly the respective output attribute). Instances represent individual cases of the concept to be learned.

Dataset. A matrix of data, where each column represents an attribute, and each row represents a different instance. Generally, the last column represents the output attribute, and the remaining represent the input attributes.

- Training Dataset. Group of instances that are used to learn the best model for that specific data.
- Test Dataset. Group of instances that are used to calculate different statistics on the generated model.

Model. A model can be defined as a function that given an instance's input attributes predicts its output attribute.

Algorithm. An algorithm defines a process that, given a training data set and some pre-chosen criteria, chooses a specific model.

Supervised learning is a method that uses a given input data (training data set) to generate a model function that infers the underlying relationship between that data. Using that model, the result of the class label (out-attribute) can be predicted. This method is really useful when it comes to predict the class label of a data set with hidden phenomena attached in unfamiliar or unobserved data instances [11]. The errors associated with the prediction can be minimized based on the quality of the data set (training data set) used. Small data sets (20-30 examples) are generally a poor choice for these algorithms. Data sets that contain many similar examples can be a bad choice as well, since they can cause overfitting of the model. The best choice for a data set would be a data set that is a good representation of all possible instances (generalized) in order for the model to have an example of each possible outcome [11].

Development of ML algorithms. When developing a ML algorithm, 7 major steps should be considered (Figure 1). The first one is to collect the data, where a subset of all available data attributes that might help in resolving the problem are selected. The second step is to process the data, making it understandable. Then the data is transformed, by feature scaling, decomposition, or aggregation (combining multiple instances into a single feature). The next step is to train the algorithm, where the training and testing datasets from the data previously transformed is selected. The algorithm is then trained (fourth step) using the training dataset, extracting the knowledge or information on that dataset, yielding a model that is a representation of that information. This model can then be used to predict the output attribute of other similar data.

Step five is where the algorithm using the test dataset is evaluated. In this step, evaluation of the effectiveness and performance of the model is performed. Giving the input attributes of the test dataset to the model, and hiding the output attribute, it predicts the output attribute for each instance. Comparing the two results, the known and the predicted, calculating different statistics about the performance of the model can be achieved. In the next step, the model created can eventually improve by using a different dataset or improving the old one. The last step is to apply the validated model, making reliable predictions on new datasets with unknown out-attributes.

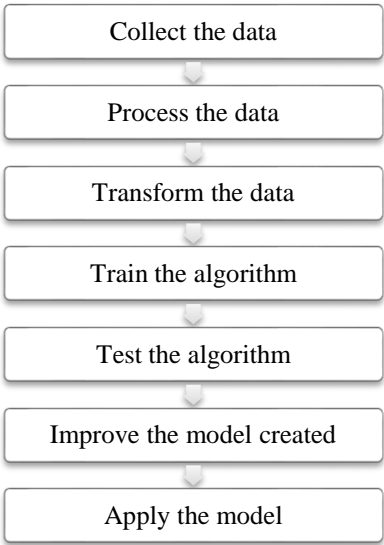


Figure 1- Seven step process for developing a machine learning algorithm

Machine learning models and algorithms. The next chapter will review some of the major models and algorithms used in ML, namely k-nearest neighbours (KNN), Naïve Bayes (NB), linear and logistic regression, decision trees, artificial neural networks (ANN), and SVMs (Support vector machines).

K-nearest neighbours (KNNs)

KNNs is an instance based learning method, meaning that it does not generate a model function. Instead, it stores the training set and uses it when a new prediction needs to be made, being called a lazy learning method for that reason, since it delays the learning process [13]. KNN has two simple ways of making predictions depending on the type of problem (classification or regression). In a classification problem, the algorithm will simply find the *k* training examples most similar to the new example to be predicted and the final prediction will be the most

common class on those k training examples. In a regression problem, the algorithm will predict the final result as the mean of the nearest neighbour values.

To calculate the nearest neighbours, a distance function must be used. For continuous (e.g. height) or linear discrete (e.g. number of children) features, Euclidean distance or Manhattan distance are usually used. For linear symbolic features (e.g. symptoms) the most common way of calculating their distances is to assume the value 1 for a different feature and the value 0 for an equal feature, and then calculate the examples distances using the Euclidean or Manhattan distance [13]. However, the previous method does not consider that some linear symbolic features may have an order, so a different distance function must be used. Value difference metric (VDM) considers two features to be closer if they have more similar classifications, thus ranking the features and providing a better calculation of the neighbours [15].

Naïve Bayes (NB)

NB is a simple algorithm that uses relative frequencies to estimate the probability of an example to present a certain result [13][16], assuming (although naively) the attributes are independent and cannot be differentiated in terms of importance. To better understand this algorithm, the data in Table 1 will be used.

Table 1-Table containing the number of times a class occurs given a certain feature

Attribute X			Attribute Y		
	Class-Yes	Class-No		Class-Yes	Class-No
X1	0	3	Y1	1	3
X2	5	1	Y2	4	1

Example “X1,Y2”

$$L(Yes) = \frac{0}{5} * \frac{4}{5} * \frac{5}{9} = 0$$

$$L(No) = \frac{3}{4} * \frac{1}{4} * \frac{4}{9}$$

$$= 0,08(3)$$

Figure 2-Calculation of the probability of classifying as “Yes” or “No” in the example “X1, Y2”

This table has the number of times each attribute's (X or Y) value (1 or 2) takes place in each possible class value (Yes or No). Given a new example "X1, Y2" one can classify the probability of resulting in a Yes or a No. This can be achieved calculating a function L (likelihood) by multiplying the relative frequencies of each of the examples attribute values with the relative frequency of the class. An example can be found in Figure 2. The final prediction will be the one that presents the highest L value. To get the probability of a given class to be predicted, its L value must be divided by the sum of all L values for all classes.

Linear Regression

Linear Regression is a method that tries to model the relationship between an independent variable and a set of dependent variables, generating a linear equation that fits the data [13][17]. Linear regression consists in finding a best-fitting set of coefficients minimizing the sum of squared errors (SSE) of prediction or gradient descent.

Logistic Regression (LR)

LR is used in classification problems, where it models the relationship between a set of independent variables and a dependent variable (binary class), predicting the probability of occurrence of the dependent variable [11][13][16]. In this process, a logistic function is calculated. With this function, estimating the probabilities of a given class can be performed. To minimize the error function, the minimization of a loss function is also conducted as in linear regression. Logistic regression can also be used in classification problems with more than 2 classes by creating a model for each class.

Decision Trees

Decision Trees, used for classification problems, generate classifiers by synthesizing a model based on a tree structure [13]. This tree is composed by n nodes and each node corresponds to an input attribute to be tested. On each node, n possible branches come out corresponding to the values or conditions the attribute can present, leading to n different nodes. If a leaf is reached, the information on that leaf corresponds to the output attribute's value (or class). To make a prediction on a new example, each attribute is tested on its specific node, starting from the root. After the root's specific attribute has been tested on the condition, it follows the branch that suits that condition, getting to another node. This process is repeated till a leaf is reached,

and the end prediction is obtained. Examples of decision trees training algorithms are the algorithms ID3 and C4.5 [13][11].

Regression Trees

Regression Trees are a variation of decision trees that can be used in regression problems. They are adaptations of decision trees where the leafs instead of class values are composed of a numeric value. M5 is an algorithm that tackles one of the regression trees fundamental problems: the fact that it can only assign a constant value to its leafs. On this algorithm each leaf is composed by a linear model allowing the calculation of the out-attribute as a linear function of the in-attribute's values [13].

Artificial Neural Networks (ANNs)

ANNs represent attempts of simulating the human brain's neurons [13][11]. These simulated neurons (nodes) like normal neurons, receive inputs and give outputs to other simulated neurons. There are two major types of ANN: Feedforward ANNs (no cycles) and Recurrent ANNs (with cycles) and both can be represented by a graph. In feedforward ANNs, this graph can be divided into an organized disposition with 3 layers: an input layer, a hidden layer, and an output layer. A node (y) has n nodes connected to it (x_n) with different output values (X_1, \dots, X_n) and each connecting has a weight associated (W_{y1}, \dots, W_{yn}). A node's output value is calculated by an activation function using the activation value. The activation value of a node "y" can be calculated through the sum of all the "x" nodes output values times the connection weight of the nodes ($Av = \sum X_n * W_{yn}$). Additionally, a bias connection with value of +1 can also be added, adding to the activation value the value of that nodes weight. There are a variety of training algorithms that minimize the cost function for ANN, leading to modifications in the weights of the connections of the nodes, namely backpropagation, Rprop, Quickprop, etc [13].

Support Vector Machines (SVMs)

SVMs are used for both classification and regression problems. These models are based on creating support vectors from the dataset, which are only a subset of the total dataset calculated by an optimization step that regularizes an objective function by an error term and a constraint [11][13][16]. For a classification problem, the support vectors are used to calculate a hyperplane that separates the data into two classes, always maximizing the margins of the hyperplane. For regression problems, the data will lie within a "tube" around the hyperplane. However, there are data that can't be divided into the two classes by a linear hyperplane or don't fit a linear

hyperplane tube, so a more complex polynomial function has to be applied. In that case kernel methods are used, like polynomial, Gaussian and spline kernels, and can be configured using different parameters.

Other relevant algorithms. Hidden Markov models (HMMs), are based on Markov chains that can be defined as a sequence of states over time (S_1, \dots, S_n) , where the change of a state to the next has a certain probability associated [11]. HMMs, like Markov chains, have a sequence of states S (S_1, \dots, S_n) but they are hidden (latent variables) and each S state is associated with an observed variable X (X_1, \dots, X_n) . Transitions between hidden states $(S_n \rightarrow S_{n+1})$ and observed variables in a hidden state $(S_n \rightarrow X_n)$ have a probability associated. The HMM can be defined by an initial distribution $(P(S_1))$, the transitions probabilities $(P(X/X-1))$, and the emissions probabilities $(P(X/S))$. In a HMM by analysing the observed states, one cannot say exactly which sequence of hidden states generated the observed ones. However, it is possible to calculate the probability of a given sequence, of hidden states, attaining the observed states. For more information about HMMs consult [11].

Ensemble methods. Ensemble methods are a way of developing learning algorithms that generate an ensemble of different models for a given problem [13]. The final result is obtained by a function that combines the individual models' results and returns a single value. To generate a better final prediction than the individual models these have to respect two conditions: the individual models have to be precise, meaning that they have to present better results than a random model, and be diverse, meaning that they have to make errors in different spaces of the test dataset [13].

The most popular way of creating ensemble models for unstable induction algorithms (those that show considerable changes in the model when faced with changes in the training dataset) is to change the training dataset presented to the algorithm, thus generating different models that will form the ensemble [13]. In this category bagging, cross-validation and boosting are the most frequently used. Bagging is based on bootstrap, where the bootstrap sample (training dataset) will be generated by a sampling process with substitution [13]. Cross-validation consists in splitting the dataset into portions of the same size, where each model will be created using different sets of training and test data. Boosting is also based on bootstrap, but in this case after each boosting iteration a weight is applied to each training example, increasing the weight on the incorrectly predicted examples and decreasing the weight in the correctly predicted examples [13][19].

Another approach to create ensemble methods is to introduce random choices in deterministic models, thus creating different models in each training. In the cases where the algorithm is already stochastic, ensemble models can be created by modifying some of the algorithm's initial parameters, like varying the number of intermediate nodes in a neural network algorithm [13]. Depending on whether it is a classification or regression problem, the functions that combine the results of the individual models can vary. When facing a classification problem, two approaches can be taken, a vote function or a winner-takes-all function. The former essentially chooses the result that was shown by most of the models, whilst the latter assumes the final result as the one shown by the model with the most confidence (assuming each model is capable of calculating the probability of the result to be correct). Regarding regression problems, a mean function that assumes the final result as the mean of the individual results or a weighted mean function that is similar to the previous but assigns a weight for each model can be used [13]. Besides creating ensemble methods that generate different models but use the same learning technique, creating hybrid systems that combine two or more learning techniques can obtain a more precise result [13].

Evaluating machine learning models. To evaluate the quality of a model for a given task different error metrics must be calculated. These error metrics will depend on the type of problem, being it a classification or a regression problem.

For classification problems, a confusion matrix is usually calculated. For a 2 classes problem, the confusion matrix (Table 2) is composed by 2 rows and 2 columns, where the rows represent the desired values (first row-negative values and second row-positive values) and the columns the predicted values (first column – negative values and second column-positive values). From the predicted values, if a value is predicted as negative and its real value is negative it is called a True negative (TN), but if its real value is positive it is called a False negative (FN). Similarly, if a value is predicted as positive and its real value is negative, it is called a False positive (FP), but if its real value is positive it is called a True positive (TP).

Table 2-Confusion matrix example

Confusion matrix		Predicted values	
		Negative	Positive
Desired values	Negative	True negative (TN)	False Positive (FP)
	Positive	False negative (FN)	True positive (TP)

With this, the calculation of accuracy (known as PECC - Percentage of Examples Correctly Classified) can be calculated by summing the TN with the TP and dividing by the sum of the TN, TP, FP and FN as seen in (14), recall (also known as sensitivity) can be calculated by the TP divided by the sum of the TP and FN as seen in (2), specificity (type error I) can be calculated by the TN divided by the sum of the TN with FP as seen in (3), precision (known as positive predicted value) can be calculated by the TP divided by the sum of the TP with FP as seen in (4), negative predictive value can be calculated by the TN divided by the sum of the TN with FN as seen in (5), and F score can be calculated by multiplying 2 to the multiplication of the precision and recall divided by the sum of the precision and recall as seen in (6) [13].

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

$$Recall = \frac{TP}{FN + TP} \quad (2)$$

$$Specificity = \frac{TN}{TN + FP} \quad (3)$$

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

$$Negative\ predictive\ value = \frac{TN}{TN + FN} \quad (5)$$

$$F\ score = 2 * \frac{precision * recall}{precision + recall} \quad (6)$$

For a classification problem with more than 2 classes, the values are calculated as if a 2x2 confusion matrix existed for each class, while the “negative” values are the elements of the other classes [13].

The best model will be the one that presents higher numbers in the PECC, Recall, and Specificity values. However, sometimes one must find a balance between the Recall and Specificity value because an increase in the Recall value can cause Specificity to lower.

ROC (Receiver Operating Characteristic) curves are also a good form to evaluate a model since they show the relationship between 1-Specificity and Recall. Calculating the area under the curve (AUC) of the ROC curve gives us information about the ability of the model to discriminate between the two classes. An AUC of 1 means that the model can distinguish the two classes perfectly and an AUC of 0.5 means that the model has a 50% chance of distinguishing the two classes correctly, no more efficiently than a coin toss. If the classification problem has more than 2 classes, ROC curves must be applied for each class, being the global AUC given by a weighed mean of the frequencies of each class [13].

In a regression problem, the error metrics are calculated based on the error presented by each example, this being the difference between the predicted value and the real value. Three different error metrics can be considered: SSE (sum of square errors) can be calculated by the summation of the squared subtraction of the desired value (y_i) by the predicted value (\hat{y}_i) as seen in (7), RMSE (square root of the mean of SSE) as seen in (8), and MAD (mean of absolute deviation) as seen in (9) [13].

$$SQE = \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (7)$$

$$RMQE = \sqrt{\frac{SQE}{N}} \quad (8)$$

$$MDA = \frac{\sum_{i=1}^N |y_i - \hat{y}_i|}{N} \quad (9)$$

For these error metrics, the model that shows lower values is more precise. A model that presents a value of 0 in this metrics is the ideal model.

For regression problems a REC (Regression Error Characteristic) curve can also be used, since sometimes the other metrics are not sufficient to understand the behaviour of the model [13].

To achieve a more accurate evaluation of a model's performance, a K-fold cross validation can be performed [20]. The data set will be divided into K subsets, where 1 subset will be used as a test dataset and the others as the training dataset [13]. Then the desired error metrics are computed by averaging them across all K trials. The advantage of this method is that every

subset is going to be used as a test dataset one time, and as a portion of the training dataset $k-1$ times.

Leave-one-out cross validation, can also be used. It is similar to K -fold cross validation, yet in this case K corresponds to the number of examples in the dataset, meaning that in each run has a training dataset of $K-1$ examples and a test dataset consisting in the 1 example that was left out [13]. The disadvantage of this process compared to the K -fold cross validation is that it requires a larger amount of time to compute.

A Bootstrap method can also be executed on the dataset, achieving that way a more accurate evaluation of the model's performance. The most used form of bootstrap considers a dataset of size n , where a bootstrap sample (training dataset) will be generated from that dataset by a sampling process with substitution [13][20]. The bootstrap sample created will have the same size as the original dataset, but will be composed by some repeated examples. The unused examples of the original dataset will compose the test dataset [4][6].

Overfitting and underfitting. Generating a model that is an excellent representation of the data is a difficult task. Since the dataset presented to the ML algorithm corresponds only to a fraction of the total data, the model generated will only be an approximation of the total data. ML is usually used to solve complex regression or classification problems, with many features and outcomes. Hence, choosing the dataset that is going to be used is a task of major importance. The best choice would be a dataset that is a good representation of all possible instances (generalized) for the model to have at least one example of each possible outcome.

Two of the major problems associated with some ML algorithms are over- and under-fitting of the generated model, and how to reach a balance between them. An over-fit model is a model that represents the training data too well, being unable to make correct predictions on the data that is not identical to the training data. This happens when the algorithm tries to generate an excessively complicated model, and ends up capturing the noise of the data (high variance). On the other hand, an under-fit model is a model that badly represents the training data. This happens when the algorithm can't capture the underlying trend of the data, and ends up generating an excessively simple model (high bias).

Overfitting can be caused by a bad choice of training datasets. Datasets containing a high amount of similar examples, or datasets that are not a good generalization of the total data can cause overfitting. When facing overfitting, several strategies can be used to lower the variance, such as, adding new data to the training dataset (when the training dataset is small), performing feature or model selection, regularization of the training process, reduce the model complexity,

or all of them combined. To reduce the high bias caused by under-fitting, the model complexity must be increased.

Model selection. Model selection is the process of choosing a model from a range of different models with different levels of complexity that were trained with the same dataset [13]. This process is conducted by evaluating the models on one or more error metrics and eventually choosing the one with the better scores. This can be difficult since all the scores must be considered (i.e. comparing two models, a model with a bit higher PECC score does not mean a better model if the F1 score is much lower). The main advantage of this process is to adjust the model to the complexity of the data, thus reducing the overfitting.

Since during the training of the model some randomness is induced, either it being in the division of the dataset into training and test datasets or in the model itself (stochastic models), a single comparison of the error metrics is not enough to conclude which model is performing the best. To conduct a reliable comparison of the models, a high number of simulations must be executed, each using different training and test datasets (variations of the initial dataset).

The higher the number of simulations the better, since it will generate a more precise mean of the error metrics, yet requiring a high demand of computational time [13]. Taking this into account, the most common number of simulations goes between 10 and 200.

Besides the mean of each model, calculating the mean's standard deviation and the respective confidence levels, allows making a more reliable choice between the models [13].

If the comparison is between 2 models, a t-test can be performed. This t-test will assess if the 2 means are significantly different from each other. Given the p-value of the t-test, if it is lower than 0,05 (considering a confidence level of 95%) the means are significantly different, otherwise the means are not significantly different. In the case of comparing more than two models, an ANOVA test can be conducted.

Feature selection. The curse of dimensionality is a problem that many models with high-dimensional features space have to face, meaning that the more in-attributes the dataset has (x) the more learning examples it needs, in many cases growing exponentially [13]. This problem causes the need to have big datasets which are difficult to save, and often hard to get.

The most obvious solution for this problem is to reduce the number of input attributes. This can be achieved by two ways: simply extracting features from the dataset (feature extraction), or selecting the most valuable features of the dataset by a process called feature selection [13]. One crucial aspect for feature selection is the way how the search for the best set of feature is

performed. For a ML classification problem, feature selection techniques can be grouped into 3 categories: filter methods, wrapper methods or embedded methods [21]. Filter methods select the features based only on the intrinsic properties of the data. Examples of these are univariate filter methods like Chi-square and Euclidean distance or multivariate filter methods like Correlation-based feature selection (CFS) or Markov blanket filter (MBF). Wrapper methods can be divided into deterministic methods and randomized methods. Examples of the first are Forward Selection and Backward Selection and for randomized methods, simulated annealing and genetic algorithms [21].

Forward selection and backward selection are examples of hill-climbing methods, where in the first the model starts with only one feature and then starts adding more features in each iteration, and in the second the model starts with all the features and then removes a feature in each iteration [13]. Both are greedy methods, meaning that both will stop if they find a local minimum, which may not be equal to the global minimum (best solution).

Examples of Embedded methods are decision trees, weighted naive Bayes and feature selection using the weight vector of SVM [21].

2.2 Sequence analysis algorithms and tools

Homology searching algorithms and tools. Sequence similarity searching aims to identify homologous sequences in databases through a process that provides additional and very important information about new sequences. This process requires assessing if the match between a query sequence and other sequences from a database is statistically significant. Such results allow inferring homology between sequences and using the annotated data of the homologue sequences to know more about the query sequence (e.g. function, family) [22].

There are a lot of different homology searching tools (a variety of this tools can be accessed at: <http://www.ebi.ac.uk/Tools/sss/>), but by far the most used is the Basic Local Alignment Search Tool (BLAST) [23] that can be accessed at: “<http://blast.ncbi.nlm.nih.gov/Blast.cgi>”. A “How to BLAST” guide can be found in [24]. Different BLAST programs can then be chosen: nucleotide blast, protein blast, blastx, tblastn, and tblastx.

Nucleotides BLAST allows searching a nucleotide database using a nucleotide query using the BLASTn (somewhat similar sequences), mega BLAST (highly similar sequences) or discontinuous mega BLAST (more dissimilar sequences) algorithms [24]. TBLASTx uses a translated nucleotide query to search a translated nucleotide database [24].

Protein BLAST allows searching a protein database using a protein query using BLASTp (protein-protein BLAST), psi-BLAST (Position-Specific Iterated BLAST), phi-BLAST (Pattern Hit Initiated BLAST), and delta-BLAST (Domain Enhanced Lookup Time Accelerated BLAST) algorithms [24]. BLASTx is used to search a protein database using a translated nucleotide query, and finally tBLASTn uses a protein query to search a translated nucleotide database [24].

Many aspects of the search can be defined in BLAST, such as the database that is going to be used for the query homology search, being the Non-redundant protein sequences (nr) the most frequently used. Also, the BLAST search can be limited to a specific organism or taxonomic group, or using a key-word or query size using Entrez Query. Many parameters of the algorithms selected can also be changed by the user.

The HMMER tool (available at: <http://www.ebi.ac.uk/Tools/hmmer/>) is a homology searching tool that can build a HMM profile using a multiple sequence alignment given from the user using the HMMER3 tool (available at: http://myhits.isb-sib.ch/cgi-bin/hmmer3_search) [25]. The HMM profile can then be used to search databases of protein sequences.

Motif finding algorithms and tools. Motifs are widespread patterns within sequences of nucleotides or amino acids that usually have biological significance, making them useful for inferring a protein's function or even to identify sequence homology [26]. Finding motifs can be achieved by different methods including simple heuristic algorithms, expectation-maximization algorithms (E-M) like the one used in the MEME tool, Gibbs sampling and HMMs like the ones used in HMMER.

Position weight matrices play a huge role in some motif finding algorithms. These matrices can be calculated through a conversion of a relative frequency matrix (P profile) using a formula [27][28]. In a multiple sequence alignment, the relative frequency matrix columns represent the positions of the sequence and the lines the possible alphabet character. The matrix shows the probability of a character to be in a certain position.

The MEME tool (http://meme-suite.org/doc/meme.html?man_type=web) uses a MM (mixture model) algorithm that is an extension of the expectation-maximization (E-M) algorithm [29] for fitting finite mixture models to discover motifs in a dataset of sequences [30].

Gibbs sampling (available at: <http://ccmbweb.ccv.brown.edu/gibbs/gibbs.html>) is a Markov Chain Monte Carlo (MCMC) approach, since like in Markov chains the results from every step depend only on the precedent step and the next step is calculated based on random sampling

[31]. Assuming n sequences (S_1, \dots, S_n) are being used and the sought motif has size W , the algorithm can be characterized into 6 steps:

1. Randomly chose an initial starting point for every sequence.
2. Randomly chose a sequence (S).
3. Create a P profile of the other sequences based on S .
4. For every single position p in S , calculate the probability of the segment initiated in p with length W being generated by P .
5. Chose p stochastically according to step 4.
6. Repeat steps 2 to 5 till the score cannot be improved.

The HMMER tool (available at: <http://www.ebi.ac.uk/Tools/hmmer/>) is a homology searching tool that can also find domains on a sequence using the hmmscan program [25]. This program uses the query given from the user and searches it against a HMM profile [32] library database e.g. Pfam (available at: <http://pfam.xfam.org/>) [33].

Analysis of protein sequences. A protein sequence is a sequence of amino acids that resulted from the translation of a nucleotide sequence. Analysing a protein sequence can thus help in the characterization of the protein (e.g. structure, function, localization).

A protein structure can be analysed in different levels: the primary structure composed only by the protein's linear amino acid sequence, the secondary structure where the interactions between amino acids using hydrogen bonds form alpha-helices, b-sheets, turns or coils, the tertiary structure in which the protein folds by attraction forces between the secondary structures, and finally, the quaternary structure formed by two or more different proteins interacting [34]. Information about the protein structure can be obtained by analysing its sequence, although the determination of the structure from the sequence is a very challenging task.

A protein can be located in different areas or organelles of a cell. Information about the protein location may also be predicted by analysing its sequence (e.g. signal peptides).

One huge problem in the analysis of protein sequences is that proteins usually undergo a process of modifications after being translated. Post-translation modifications (PTMs) usually take place after the translation of the mRNA and can induce changes in the protein's physical or chemical properties, activity, localization, or stability. These PTMs cause an increasing complexity of the proteome in comparison to the genome. Some examples of PTMs are, according to [35] [36], described below.

- Glycosylation involves the connection of sugar molecules to the protein and is very important to the subcellular localization of the protein (fixation to the extracellular matrix), cell-to-cell interactions, and ligand-to-protein interactions.
- Phosphorylation involves the connection of a phosphate group to a protein and is very important in protein links, cellular cycles, signalization pathways, and enzyme regulation.
- Hydroxylation involves the connection of a hydroxyl group to a protein.
- Acetylation involves the introduction of an acetyl group to a protein and is very important in turning on or off proteins and degradation signalling.
- Methylation involves the transfer of one-carbon methyl groups to nitrogen or oxygen and is very important in the increase of the hydrophobicity of the protein and epigenetic regulation.

Integral membrane proteins (e.g. transport proteins) cross the lipid bilayer membrane at least once. In order to do so they present hydrophobic transmembrane segments that cross the highly hydrophobic core of the lipid bilayer [37].

Transport proteins have high numbers of transmembrane segments like alpha-helices and beta-sheets (to a lesser extend) [37] compared to other proteins. When beta-sheets fold on themselves they form a beta-barrel that is present in some transport proteins. Identifying and characterizing these transmembrane domains can help in the classification of the transport proteins.

2.3 Relevant bioinformatics tools and databases

Protein localization tools. In this section, a quick overview of some protein localization tools is performed, namely PSORTb [38], TargetP [39], BaCello [40], ESLpred2 [41], WolfPSORT [42], SubLoc [43], LocTree3 [44], Cell-Ploc2 [45], Cello [46] and PSLPred [44][45].

These tools tackle the subcellular localization prediction problem in different ways, using different models (Table 3), algorithms and datasets for training and testing. The most frequently used models by the predictors are PSSMs, Homology, ANNs, SVMs, KNNs, Naïve Bayes, and Decision Trees. Some of the predictors use more than one model creating a hybrid algorithm. For prokaryotic proteins, the tools PSORTb [38], PSLpred [47], [48] can be used, while the tools TargetP [39], BaCello [40], ESLpred2 [41], Wolf-PSORT [42] can be used for eukaryotic proteins. LocTree3 [44], Cell-Ploc2 [45], Cello [46] and SubLoc [43] are capable of assessing

the subcellular localization of both eukaryotic and prokaryotic proteins. The PSORTb [38] and LocTree3 [44] tools are also capable of predicting the localization of *archaea* proteins.

The same dataset should be used for comparing the performance of these tools.

Table 3-Different Models used by some Localization prediction tools and their respective websites.

Localization Tools	PSSM	Homology	ANN	SVM	KNN	Naïve Bayes	Decision Tree	Website
PSORTb				X		X		http://www.psorbt.org/psortb/
TargetP			X					http://www.cbs.dtu.dk/services/TargetP/
BaCello				X			X	http://gpcr.biocomp.unibo.it/bacello/
ESLpred2	X	X		X				http://www.imtech.res.in/raghava/eslpred2/
WolfPSORT					X			http://www.gencript.com/wolf-psort.html
SubLoc				X				http://www.bioinfo.tsinghua.edu.cn/SubLoc/
LocTree3				X			X	https://roslab.org/services/loctree3/
Cell-Ploc2					X			http://www.csbio.sjtu.edu.cn/bioinf/Cell-Ploc2/
Cello				X				http://cello.life.nctu.edu.tw/
PSLPred		X		X				http://www.imtech.res.in/raghava/pslpred/

Membrane protein topology tools. In this section, a quick overview of some membrane protein topology tools is performed, namely DAS-TMfilter [49], HMMTOP [50], Phobius [51], Predict Protein [52], TMHMM [53], TMPred, BOMP tool [54] and the PRED-TMBB tool [55].

These tools use different models and algorithms to access the topology of membrane proteins (Table 4) and are of great help for identifying and classifying transport proteins. An evaluation

of some of these methods performance can be accessed at [56]. More tools useful for predicting transmembrane domains are provided in <http://www.psort.org/>.

Table 4- Overview of the objective and models used in some Membrane protein topology tools and their respective websites.

Membrane protein Topology tool	Objective	Model/algorithm	Website
DAS-TMfilter	Identify transmembrane helices	Das algorithm (hydrophobicity profile based)	http://www.enzim.hu/DAS/DAS.html
Phobius	Predict transmembrane protein topology and signal peptides	HMM	http://phobius.sbc.su.se/
HMMTOP	Predict helical transmembrane segments and topology of transmembrane proteins	HMM	http://www.enzim.hu/hmmtop/
PredictProtein	Predicts alpha-helical transmembrane proteins topology(TMSEG) and coiled-coil regions (COILS)	ANN	https://www.predictprotein.org/
TMHMM	Predicts transmembrane alpha-helices	HMM	http://www.cbs.dtu.dk/services/TMHMM/
TMPred	Predicts transmembrane alpha-helices	WSM	http://www.ch.embnet.org/software/TMPRED_form.html
BOMP	Predicts transmembrane beta-barrels	C-terminal pattern recognition and integral beta-barrel score	http://services.cbu.uib.no/tools/bomp
PRED-TMBB	Predicts outer membrane beta-barrel proteins	HMM	http://biophysics.biol.uoa.gr/PRED-TMBB/

Transport protein substrate specificity tools. TrSSP (available at: <http://bioinfo.noble.org/TrSSP/>) is a tool that is able to predict the substrate of a transport protein, using a SVM model based on biochemical composition and evolutionary information (i.e. PSSM profile). This tool is able to classify the transport proteins into seven different classes, namely amino acid transporters/oligopeptides, anion transporters, cation transporters, electron transporters, protein/mRNA transporters, sugar transporters, and other transporters [57].

FASTA format. Most of the previously described tools use as input FASTA formatted files containing more than one sequence or simply a sequence in the FASTA format. The FASTA format as described in [58] is a text based format that represents the nucleotide sequence or peptide sequence. This format always begins with “>” followed by the sequence’s description on the first line of text and the sequence in the next line, or lines, in order to separate the description from the sequence.

Databases of transporter proteins. Databases of transporter proteins are essential for the creation of a dataset suitable for a machine learning approach in regard to transporter proteins prediction and annotation. The Transporter Classification Database (TCDB) available at <http://www.tcdb.org/> contains examples of all currently known families (over 800) of transmembrane molecular transport systems [5]. TCDB contains 13788 non-redundant proteins (April, 2016) organized into five levels (TC system), dividing the transporter proteins into class, subclass, family, subfamily, and the transport system [59], [60], [61]. The first division of the TC system divides the proteins into the following categories: 1-Channels/Pores; 2-Electrochemical Potential-driven Transporters; 3-Primary Active Transporters; 4-Group Translocators; 5-Transmembrane Electron Carriers; 8-Accessory Factors Involved in Transport; 9-Incompletely Characterized Transport Systems. The TCDB is recognized by the International Union of Biochemistry and Molecular Biology, making it a good source for extraction of the transport proteins.

The TransportDB (available at: <http://www.membranetransport.org>) [62] has annotations of transport proteins in organisms with fully sequenced genomes. Each organism has its complete set of membrane transport systems identified and classified into different types and families. TransAAP [63] is a tool that can be accessed through the TransportDB website for genome wide transport protein identification and annotation. This tool receives a query genome from the user and then proceeds to identify transporter proteins by searching the query sequence

against different databases and by using the tmhmm tool. The result is a list of predicted transporters annotated automatically that can be curated by the user.

Other relevant Databases. The Universal Protein Resource (Uniprot) is a database that contains reviewed annotations (Swiss-Prot) and automatically generated annotations (TrEMBL) of protein data [64].

Developed by the European Bioinformatics Institute (EMBL-EBI), the SIB Swiss Institute of Bioinformatics and the Protein Information Resource (PIR), Uniprot, and more specifically Swiss-Prot is a great source of reviewed protein data.

The Pfam database [37] is composed by a collection of protein families that are represented by multiple sequence alignments and HMMs. Since proteins are composed by one or more functional domains, identification of these domains within the protein's sequence can lead to the functional characterization of the protein.

2.4 Relevant development environments

Biopython library. In this section a quick review is performed on what the Biopython (web site: http://biopython.org/wiki/Main_Page) libraries can provide.

Biopython has great functionalities that make working with sequences an easier process using the "Seq" object. This is a sequence object containing the sequence's string and the sequence's alphabet. The "Seq" object supports different methods including finding that sequence's complement or reverse complement. A class called "SeqRecord" holds a sequence as a "Seq" object but with additional information, including an identifier, name and description. The "SeqRecord" object can easily be annotated changing his attributes (seq, id, name, description, letter_annotations, annotations, features, and dbxrefs).

The Biopython libraries have a good set of parsers that are able to parse different files in different formats including FASTA, GenBank, PubMed, SwissProt, Unigene and SCOP. These parsers allow access to the information contained in the records of the file.

Biopython can also do sequence alignments using the module "AlignIO".

Extracting information from different biological databases is feasible with Biopython, making it easier to retrieve information from databases as Entrez, PubMed, SwissProt, Prosite, etc. Biopython also allows conducting a BLAST search locally or remotely.

More information and examples about the Biopython functionalities is available at the Biopython Tutorial and Cookbook [65].

Scikit-learn and other scientific computing libraries in python. Scikit-learn (<http://scikit-learn.org/stable/>) is a library containing many resources useful to solve machine learning problems using Python [66]. The scikit-learn uses the numpy, scipy and matplotlib libraries available at <http://www.scipy.org/>. In this section, a quick review on some of the scikit-learn functionalities useful for solving supervised learning problems will be presented.

First, for a dataset to be used in the scikit-learn package it has to be composed by input attribute values (numpy array with $n*m$ dimensions, where n are the examples and m are the in-attributes) and output attribute values (numpy array with 1 dimension of size n). A file containing the dataset can be easily loaded using the “genfromtxt” function from the “numpy” library.

The datasets can be divided into training and test dataset importing the “cross_validation” function. A parameter can be changed to define the proportion of the division

The scikit-learn library has a variety of models for supervised learning problems including KNNs, decision trees, naïve bayes, linear and logistic regression and SVMs.

After creating the model, it can be easily trained by fitting the dataset to the model by the “fit” function after providing the training dataset to the model.

The scikit-learn libraries also have a variety of models for unsupervised learning problems including clustering, principal component analysis (PCA), etc.

Evaluation of the model performance using cross validation can be easily calculated, and other performance estimators like leave-one-out or Kfold cross validation can be created. The scoring parameter is the error metric used by the validation function, if omitted it considers the default method’s estimator as the error metric, but other error metrics like the f1 error metric can be chosen. For regression problems, the scoring parameter can be changed for the R^2 error metric, mean squared error and mean absolute error.

Ensemble methods like bagging, random forests and boosting can be applied as other regular models. Feature selection is also possible. A variance filter can be implemented, while univariate filters using Chi squared and linear regression are also available. Scikit learn can also implement Recursive feature elimination (RFE) as a wrapper method for feature selection. The process of model selection can be implemented using an exhaustive grid search or a randomized parameter optimization.

Scikit learn has many other utilities, including dataset transformations and dataset loading utilities. For more information about the scikit learn functionalities you can access the scikit learn user guide [67].

3. METHODS

3.1 Data

Creating a model capable of separating transport proteins (positive cases) from non-transport proteins (negative cases) using their amino acid sequence as the basis for generating features, requires the usage of well annotated and reviewed proteins.

The positive cases (transport proteins) were obtained by downloading a FASTA file from TCDB containing all the TCDB's proteins, therefore obtaining 13788 positive cases.

The negative cases (non-transport proteins) were obtained by filtering the Swiss-Prot database (total of 551,987 proteins) with the query <NOT "transporter protein", NOT "transmembrane", NOT "transport activity">, getting a total of 467,680 proteins in a FASTA formatted file. The query <NOT "transmembrane"> was used to guarantee that a better elimination of transport proteins from the Swiss-Prot database was achieved. Note that the use of <NOT "transmembrane"> might cause some difficulties for the models generated in this thesis when trying to predict if a transmembrane protein is a transporter protein or not.

3.2 Input attributes and output attributes

For the development of a good model, a good set of input attributes (features) is necessary. In this thesis, 11 different types of features are generated for each protein. A brief description of each feature is provided next.

1. **Amino acid occurrence** is an array of 20 columns where each column contains the number of times a specific amino acid is present (n_i) in the protein's sequence (10).
2. **Amino acid composition** is an array of 20 columns where each column contains the amino occurrence of a specific amino acid divided by the total number of amino acids of the sequence (N) (11).

$$A. Occurrence (i) = n_i \quad (10)$$

$$A. Composition (i) = \frac{n_i}{N} \quad (11)$$

In order to try to obtain more information from the amino acid sequence a division based on their physicochemical properties was created.

3. **Amino acid physicochemical occurrence** is an array of 11 columns containing the sum of the times each amino acid (i) from a specific group (n_j) occurs (12). The groups are as follows: the first one is the amount of charged amino acids (D, E, K, H, and R), followed by Aliphatic amino acids (I, L, and V), Aromatic amino acids (F, H, W, and Y), Polar amino acids (D, E, R, K, Q, and N), Neutral amino acids (A, G, H, P, S, T, and Y), Hydrophobic amino acids (C, F, I, L, M, V, and W), positively charged amino acids (K, R, and H), negatively charged amino acids (D and E), tiny amino acids (A, C, D, G, S, and T), Small amino acids (E, H, I, L, K, M, N, P, Q, and V), and finally, large amino acids (F, R, W, and Y).
4. **Amino acid physicochemical composition** is an array of 11 columns where each column contains the amino acid physicochemical occurrence previously showed divided by the total number of amino acids in the sequence (13).

$$A.PQ.occurrence(j) = \sum n_{ji} \quad (12)$$

$$A.PQ.compostion(j) = \frac{\sum n_{ji}}{N} \quad (13)$$

$$D.comp(i) = \frac{d_i}{N-1} \quad (14)$$

5. **Dipeptide composition** is an array of 400 columns containing the number of times a specific dipeptide occurs (d_i) divided by the total number of dipeptides ($N-1$)(14).
6. The **Alpha helices** feature is an array with only one column containing the number of alpha helices estimated by the Phobius tool [51] [68].
7. The **Signal peptides** feature is an array with only one column where proteins with an estimated signal peptide by the Phobius tool have a “1” and proteins with no estimated signal peptide have a “0”.
8. The **Beta barrel** feature is an array with only one column with “1” or “0” depending if the tool BOMP has predicted a beta barrel formation or not [54], [69].

9. **Subcellular location** feature is an array with one column containing numbers between 0 and 24 depending on the subcellular location predicted by the LocTree3 tool [44], [70]. The numbers represent the following locations: 0 is given when an error occurs; 1 - “chloroplast”; 2 - “chloroplast membrane”; 3 - “cytosol”; 4 - “endoplasmic reticulum”; 5- “endoplasmic reticulum membrane”; 6- “extra-cellular”; 7- “fimbrium”; 8- “golgi apparatus”; 9- “golgi apparatus membrane”; 10- “mitochondrion”; 11- “mitochondrion membrane”; 12- “nucleus”; 13- “nucleus membrane”; 14- “outer membrane”; 15- “periplasmic space”; 16- “peroxisome”; 17- “peroxisome membrane”; 18- “plasma membrane”; 19- “plastid”; 20- “vacuole”; 21- “vacuole membrane”; 22- “secreted”; 23- “cytoplasm”; 24- “inner membrane”.
10. **Number of transporter related Pfam domains** is an array with one column that contains the number of transporter related Pfam domains presented by the proteins. The transporter related Pfam domains were obtained by filtering the Pfam website (<http://pfam.xfam.org/>) with the keywords “transporter”, “channel”, “pump”, and “permease”, obtaining a total of 4042 transporter related Pfam domains.
11. **Single transporter related Pfam domains** is an array with 4042 columns, each representing one of the transporter related Pfam domains obtained. The “1” represents that the protein has that specific Pfam whether “0” represents that the protein does not have that Pfam.

The main purpose of this thesis is to create a model capable of identifying transporter proteins. Creating an output attribute called “**Is transporter**” that assigns “1” or “0” given the features, the model will classify the proteins into “transporter” or “non-transporter” respectively.

As mentioned previously the TCDB divides the transporter proteins into 5 different levels, classifying the proteins based on their TC system classification (Transporter Classification). The TC system incorporates functional and phylogenetic information; hence each level of the TC system is less ambiguous and more specific.

After accessing if a protein is a transporter protein, a more detailed classification can be achieved by trying to classify the protein based on the TCDB TC system.

The “**TCDB1**” output attribute was created to classify the transporter proteins based on the first five categories (see TCDB description in section 2.3) of the first level of the TC system. The “8” category was ignored, as its proteins are accessory factors involved in the transport and not

transporter proteins. Likewise, the “9” category was also disregarded, since such proteins have yet to be reviewed by the TCDB expert staff.

3.3 Data sets

A total of four different datasets were developed to create and evaluate different models.

The first dataset (Table 5) was composed by the features and output attribute (“Is Transporter”) of the total 27576 proteins, half of which were the proteins extracted from the TCDB containing the positive cases (13788), and the other half was a set of randomly chosen 13788 proteins from the negative cases extracted from Swiss-Prot. This dataset is an array with 4510 columns (4509 features and 1 class) and 27576 rows (13788 instances of positive cases and 13788 instances of negative cases).

The second dataset (Table 6) was equal to the first dataset only differing in the negative instances since they are randomly extracted from the negative cases generated using Swiss-Prot. This dataset was created to determine if the negative cases randomly generated had influence in the performance of the model.

The third dataset (Table 7) was created by only selecting the proteins with the first level of the TC system between 1 and 5, eliminating the 8 and 9 categories. Again, the negative instances were extracted by randomly selecting proteins from the negative cases generated by Swiss-Prot. This dataset was composed by a total of 23435 instances (11717 positive cases and 11717 negative cases), each with 4509 features and one output attribute (“Is Transporter”).

The fourth dataset (Table 8) was created by eliminating all instances of the first dataset that were not in the categories 1, 2, 3, 4, and 5 of the TC system. This dataset was composed by 11717 instances, with all the 4509 features and one output attribute (“TCDB1”) and its main purpose was to create and measure the performance of the model created to classify the transporter proteins into the 5 categories of the first level of the TC system.

Table 5- Dataset1 content

Dataset1 (27576x4510)	Features (x4509)	Is Transporter (x1)
Positive cases (13788)	...	1
Negative cases (13788)	...	0

Table 6- Dataset2 content

Dataset2 (27576x4510)	Features (x4509)	Is Transporter (x1)
Positive cases (13788)	...	1
Negative cases (13788)	...	0

Table 7- Dataset3 content

Dataset3 (23434x4510)	Features (x4509)	Is Transporter (x1)
Positive cases (11717)	...	1
Negative cases (11717)	...	0

Table 8- Dataset4 content

Dataset4 (11717x4510)	Features (x4509)	TCDB1 (x1)
Positive cases (11717)	...	1
		2
		3
		4
		5

After being created, the instances of all datasets were mixed randomly, providing a better sample of the dataset after performing cross-validation.

3.4 Dataset pre-processing

The datasets used to train machine learning models are sometimes not optimal, either containing missing values (NaN), not being standardized, or not being scaled. Indeed, before developing the machine learning model, the dataset should be pre-processed and transformed.

The pre-processing of the datasets was achieved using the “scikitlearn” pre-processing features.

All missing values on the datasets were taken off by utilizing the feature “Imputer” that replaces “Nan” values by column mean, Standardization of the datasets was achieved by removing the mean and scaling to unit variance utilizing the feature “StandardScaler”, while the scaling of the dataset was done by setting the maximal absolute value of a feature to 1.0 without shifting or centring the data utilizing the feature “MaxAbsScaler”.

3.5 Feature selection

The larger the number of features in the dataset, the more time it will take for a model to fit the data, and sometimes, some features might lower the performance of a model. Features that present the same value for every single instance are therefore useless for the training of the model and should be eliminated from the dataset. This can be achieved by applying a variance threshold filter to the dataset. Using “sklearn’s” feature selection utilities, namely “VarianceThreshold”, all features with zero variance (i.e. features presenting the same value in every instance) are eliminated.

A recursive feature elimination can be performed to obtain a smaller dataset. Once again, using “sklearn’s” feature selection utilities, namely “RFE”, this can be achieved. Recursive feature elimination uses an external estimator given by the user that will be trained using the initial set of features. The estimator then assigns weights to each feature, and eliminates the ones with the smallest weights, repeating the process, as the name suggests, until the desired number of features set by the user are achieved.

In this thesis, two different RFE filters were created using the external estimators LogisticRegression and SVM, and the number of features to be held was set to 1000 in both cases.

Before training the model, the user can choose the filters to be applied in the dataset. The use of the “VarianceThreshold” filter is recommended before the use of the “RFE” filter since it decreases the time spent in the process.

3.6 Models

Using the training dataset seven different models were developed: **Naïve Bayes** (NB) model using “sklearn’s” “GaussianNB” function; **K-nearest neighbours** (KNN) model using “sklearn’s” “KNeighborsClassifier” function; **Decision tree** model using “sklearn’s” “ExtraTreesClassifier” (ET) function; **Logistic regression** (LR) model using “sklearn’s”

“linear_model.LogisticRegression” function; **Support vector machine** (SVM) model using “sklearn’s” “svm” function; **Gradient Boosting** (GB) using “sklearn’s” “GradientBoostingClassifier” function; **Random Forest** (RF) using “sklearn’s” “RandomForestClassifier” function.

3.7 Ensemble Methods

With the objective of improving the model’s performance, some ensemble methods were implemented. A bagging classifier was used in each model using the “sklearn’s” “BaggingClassifier” function.

Two voting classifiers were implemented using the “sklearn’s” “VotingClassifier” function. The first one used a majority voting, meaning that the final prediction for each protein was the class most predicted by all the classifiers, while the second one used a weighted majority voting process where each of the classifiers have prediction weights given by the user (Figure 3).

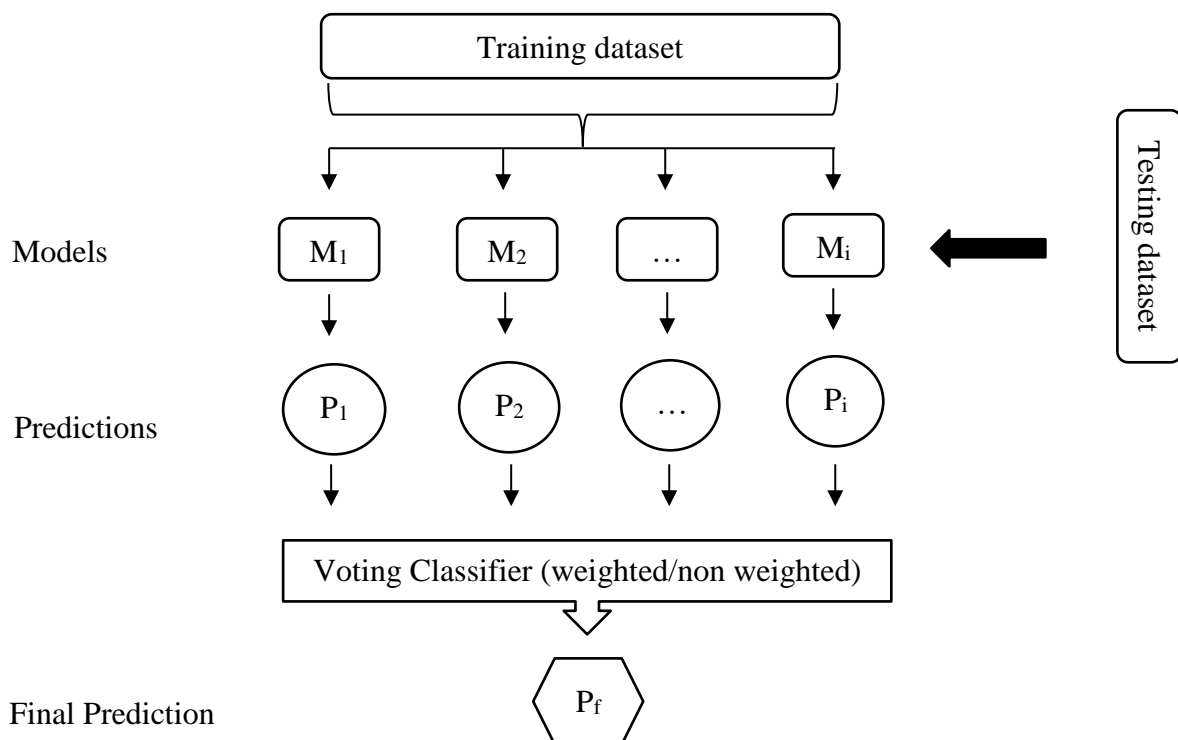


Figure 3- Voting classifier prediction method (Hard Vote classifier’s final prediction is the class predicted by most of the models, while weighted voting classifiers considers that each model has a weight given by the user. When making the final prediction, models with bigger weight have more influence in the final result.

3.8 Cross-validation

To determine the performance of a model, the dataset used for training must be different from the dataset used for testing. Using “sklearn’s” cross validation utilities, namely “cross_val_score”, a five-fold cross validation test can be performed for each model, guaranteeing that the dataset will be divided into 5 equal parts, where 4 parts of the dataset will be used to train the model and the other one for testing the model.

3.9 Performance evaluation

Evaluation of the model’s performance was accessed by a five-fold cross validation process, where the models were trained and tested using five different datasets generated from the original dataset. The results of the performance evaluation were calculated by the mean of the PECC, F1 and ROC-AUC scores for each fold of the cross-validation process. Confusion matrix scores were calculated through the division of the original dataset into training and testing datasets with a proportion of 0.80, meaning that the training dataset will be composed by 80% of the data while the test dataset will be composed by the last 20%.

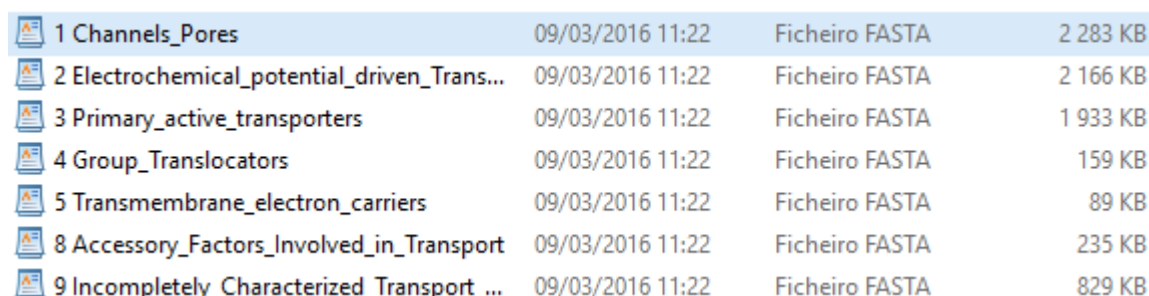
4. DEVELOPMENT

4.1 Code Developed

The code developed in this thesis was created with Python 3.5 using the Python IDE PyDev for Eclipse. It is divided into different scripts that try to mimic the process for the creation of a machine learning algorithm as exemplified in Figure 1.

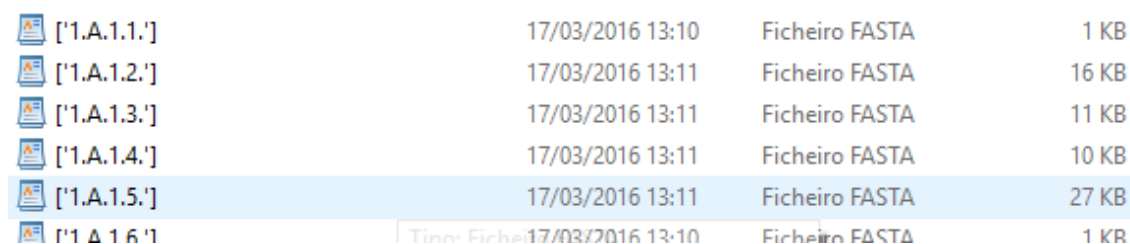
Collecting the Data was done by simply downloading the files containing the positive and negative cases as specified in 3.1.

After that, **Data processing** was implemented with several different scripts. Organization of the positive cases was performed in two different ways using two scripts “TCDB_ProteinType_Division.py” and “TCDB_ProteinType_Division2.py”. “TCDB_ProteinType_Division” purpose was to divide the FASTA records of the positive cases into seven different files according to the first level of the TC system as seen in Figure 4, while “TCDB_ProteinType_Division2” divided the positive cases into the fourth level of the TC system as shown in the example of Figure 5.



File Name	Created	Type	Size
1 Channels_Pores	09/03/2016 11:22	Ficheiro FASTA	2 283 KB
2 Electrochemical_potential_driven_Trans...	09/03/2016 11:22	Ficheiro FASTA	2 166 KB
3 Primary_active_transporters	09/03/2016 11:22	Ficheiro FASTA	1 933 KB
4 Group_Translocators	09/03/2016 11:22	Ficheiro FASTA	159 KB
5 Transmembrane_electron_carriers	09/03/2016 11:22	Ficheiro FASTA	89 KB
8 Accessory_Factors_Involved_in_Transport	09/03/2016 11:22	Ficheiro FASTA	235 KB
9 Incompletely_Characterized_Transport_...	09/03/2016 11:22	Ficheiro FASTA	829 KB

Figure 4- "TCDB_ProteinType_Division" generated files



File Name	Created	Type	Size
['1.A.1.1.']	17/03/2016 13:10	Ficheiro FASTA	1 KB
['1.A.1.2.']	17/03/2016 13:11	Ficheiro FASTA	16 KB
['1.A.1.3.']	17/03/2016 13:11	Ficheiro FASTA	11 KB
['1.A.1.4.']	17/03/2016 13:11	Ficheiro FASTA	10 KB
['1.A.1.5.']	17/03/2016 13:11	Ficheiro FASTA	27 KB
['1.A.1.6.']	17/03/2016 13:10	Ficheiro FASTA	1 KB

Figure 5- "TCDB_ProteinType_Division2" example of the generated files

Secondly, 13788 random sequences were extracted from the negative cases file using the script “**Negative_cases.py**” saving the random sequences into a new file.

From these files (positive and negative cases) a file containing the data necessary to generate the features and classes was created using the script “**Data_creation.py**”. This file is nothing more than an array where the columns are “Fasta ID”, Uniprot Accession number”, “Sequence”, “Is Transporter”, “TCDB ID”, “Taxonomy Domain”. The “Is Transporter” column contains “1” for transporter proteins and “0” for non-transporter proteins, the others are self-explanatory. Utilizing the functionalities of the Biopython library the “Fasta ID” and “Sequence” from each protein were easily obtained using the “SeqIO” parser of FASTA files. “SeqIO” creates a record, and the information on the FASTA file can be extracted from that record. The “Uniprot Accession number” and “TCDB ID” were obtained by a simple division of the “Fasta ID”. Protein Domains were acquired with Biopython’s “Entrez.efetch”, which accesses NCBI using the protein Accession number. From there a record is created using “SeqIO”, and the information about the protein domain on that record can be extracted. For some proteins the “Entrez.efetch” module could not access the proteins information on NCBI, in those cases the protein domain was obtained by accessing remotely to the Uniprot.

From the script “**Features_creation.py**” all the input attributes were created, while the output attributes were created by “**Out_attribute_creation.py**”.

The features “**Amino acid occurrence**”, “**Amino acid composition**”, “**Amino acid physicochemical occurrence**”, “**Amino acid physicochemical composition**” and “**Dipeptide composition**”, were created by taking the sequence of each protein from the “datafile”, and the respective equation for the calculation of the feature was applied as seen in the pseudo code in Figure 6, Figure 7 and Figure 8.

Feature: **Amino acid occurrence**

```
def create_aminoacid_occurrence(datafile):  
    for each protein sequence in datafile:  
        feature= create array ((count AA x in sequence, ...))  
        FEATURES= stack the feature array vertically ((FEATURES, feature))  
    return FEATURES
```

Feature: **Amino acid composition**

```
def create_aminoacid_composition(datafile):  
    for each protein sequence in datafile:  
        feature=create array ((count AA x in sequence/len(sequence), ...))  
        FEATURES= stack the feature array vertically ((FEATURES, feature))  
    return FEATURES
```

Figure 6- Pseudo code for the creation of the amino acid occurrence and composition features

```

Feature: Amino acid physicochemical occurrence
def create_aminoacid_physico_chemical_occurrence(datafile):
    for each protein sequence in datafile:
        feature=create array ((count AAs of group x in sequence, ...))
        FEATURES= stack the feature array vertically ((FEATURES, feature))
    return FEATURES

Feature: amino acid physicochemical composition
def create_aminoacid_physico_chemical_composition(datafile):
    for each protein sequence in datafile:
        feature=creat array ((count AAs of group x in sequence/len(sequence),))
        FEATURES= stack the feature array vertically ((FEATURES, feature))
    return FEATURES

```

Figure 7-Pseudo code for the creation of the amino acid physicochemical occurrence and composition features

```

Feature: Dipeptide composition
def create_dipeptide_composition(datafile):
    for each protein sequence in datafile:
        feature=create array ((count dipeptide x in sequence/len(sequence), ...))
        FEATURES= stack the feature array vertically ((FEATURES, feature))
    return FEATURES

```

Figure 8- Pseudo code for the creation of the dipeptide composition feature

For the feature “**Number of Transporter related Pfam domains**” a file containing many filtered transported related Pfam domains was used. In this feature, for each protein the NCBI or Uniprot (if the previous fails) are accessed to retrieve all the protein’s Pfam domains. The final result is an array with the number of transporter related Pfam domains the protein has. This is obtained by incrementing 1 for each Pfam domain that is equal to a Pfam domain present in the file containing the transporter related Pfam domains as seen in the pseudo code in Figure 9.


```

Feature: Number of Transporter related Pfam domains
def create_transporter_related_pfam_domains(datafile, file with the transporter pfam´s):
    for each protein in datafile:
        n°ProteinPfams=0
        try:
            handle=Access NCBI by Entrez.efetch using protein Accession n°
            record=SeqIO.read(handle, format="genbank")
            ProteinAnnotations= record.annotations
            database_source= (ProteinAnnotations.get("db_source"))
            ProteinPfams= re.findall("(Pfam:.+?),"database_source )

        for pfam in ProteinPfams:
            if pfam in file with the transporter pfam´s:
                n°ProteinPfams+=1
            feature=create array ((n°ProteinPfams))
            FEATURES= stack feature array vertically ((FEATURES, feature))
        except:
            source_code= Protein´s UniProt html page
            links= all links in the source_code
            PfamLinks= re.findall("(http://pfam.xfam.org/family/.{7})",links)
            ProteinPfams= extract Pfams from PfamLinks
            for pfam in ProteinPfams:
                if pfam in file with the transporter pfam´s:
                    n°ProteinPfams+=1
                feature=create array ((n°ProteinPfams))
                FEATURES= stack feature array vertically ((FEATURES, feature))

    Saves a file with each protein transport related pfam´s
    return FEATURES

```

Figure 9-Pseudo code for the creation of the number of transporter related pfam domains feature

The feature “**Single transporter related Pfam domains**” is an array of the size of the number of transporter related Pfam domains. As in the feature “**Number of Transporter related Pfam domains**” the Pfams of each protein are accessed by the NCBI or Uniprot. For each protein an array of zeros (size of the number of transporter related Pfam domains) is then created and the

“0s” are replaced by “1” if the protein has that specific transporter related Pfam domain as represented in the pseudo code on the Figure 10.

```
Feature: Single transporter related Pfam domains
def create_single_pfam_features(datafile, file with each protein's transport related pfam's):
    FEATURES=create array((0s array(size:n°prot in datafile x n° transporter pfam)))
    for each protein in datafile:
        replace “0” for “1” for each transporter related pfam in its respective place
    return FEATURES
```

Figure 10- Pseudo code for the creation of the single transporter related Pfam domains feature

The features “**Number of alpha helices**” and “**Signal peptide**” are both achieved using a Phobius rest API. As seen in the pseudo code in Figure 11, first the sequence of each protein is posted to the Phobius web site and the response URL of each post is retrieved. The response URLs contain the results for each protein. After filtering the results, the number of alpha helices and the presence of signal peptide are obtained.

```
Features: Number of alpha helices and signal peptide
def create_PhobiusJobIds(datafile):
    for each protein in datafile:
        use created Phobius RESTful API to post data (email, format, stype, seq)
    return PhobiusJobIds
def create_num_alphahelices_signalpeptide(datafile, PhobiusJobIds):
    for each protein in datafile:
        get results from PhobiusJobIds
        feature=create array ((n°alphahelices, signalpeptide))
        FEATURES= stack feature array vertically ((FEATURES, feature))
    return FEATURES
```

Figure 11-Pseudo code for the creation of the number of alpha helices and signal peptide features

Using a BOMP rest API the feature “**Beta Barrel**” is created as seen in the pseudo code in Figure 12. First the sequence of each protein is posted to the BOMP website, retrieving the

Feature: **Beta Barrel**

```
def create_BOMPJobIds(datafile):
    for each protein in datafile:
        use created BOMP RESTful API to post data (seq)
    return BOMPJobIds
def create_betabarrel(datafile, BOMPJobIds):
    for each protein in datafile:
        get results from BOMPJobIds
        feature=create array ((betabarrel))
        FEATURES=np.vstack((FEATURES, feature))
    return FEATURES
```

Figure 12-Pseudo code for the creation of the beta barrel feature

response URLs that contain the results. Then the result of each protein is filtered to see if the protein has a beta barrel conformation.

To create the feature “**Location Prediction**” a LocTree3 rest API was used as seen in the pseudo code in Figure 13. First the sequence and domain of each protein are posted to the LocTree3 website, retrieving the response URLs that contain the results. Then the result of each protein is filtered from the response URL to see the protein’s subcellular location.

Feature: **Location Prediction**

```
def create_LocTree3JobIds(datafile):
    for each protein in datafile:
        use created LocTree3 RESTful API to post data (domain, email, seq)
    return LocTree3JobIds
def create_location_prediction(datafile, LocTree3JobIds):
    for each protein in datafile:
        get results from Loctree3JobIds
        feature=create array ((location))
        FEATURES= stack feature array vertically ((FEATURES, feature))
    return FEATURES
```

Figure 13- Pseudo code for the location prediction feature

Out attribute: **Is Transporter**

```
def create_isTransporter(datafile):  
    for each protein in datafile:  
        check if it's a transporter or not in the datafile  
        feature=create array ((1 if transporter, 0 if not))  
        FEATURES= stack feature array vertically ((FEATURES,feature))  
    return FEATURES
```

Out attribute: **TCDB TC system IDs**

```
def create_TCDB_ID(datafile):  
    for each protein in datafile:  
        if transporter check the protein's TC ID  
        outattribute=create array (lvl1, lvl2, lvl3, lvl4)  
    OUTATTRIBUTES=stack array vertically((OUTATTRIBUTES, outattribute))
```

Figure 14-Pseudo code for the creation of the out attributes "Is Transporter" and "TCDB_ID"

To create the out-attribute "**Is transporter**" a simple access to the data file containing the proteins information was sufficient as seen in the pseudo code in Figure 14. "**TCDB TC system IDs**" out attribute is created by splitting the TC system IDs into the first 4 levels if the protein is a transporter protein as seen in the pseudo code in Figure 14.

The final dataset to be used in the training and testing of the models was generated by the script "**Dataset_creation.py**". In this script, the features and out attribute are joined in one dataset. The instances in the dataset are then mixed and the features and out attribute are divided again into two different datasets.

Script: **Dataset_creation.py**

```
def insert_features_outattribute(features,outattribute):  
    finalDataset=create array ((features,))  
    finalDataset= stack outattribute array horizontally((finalDataset,outattribute))  
    return finalDataset  
def mix_Dataset(finalDataset):  
    mixedDataset=finalDataset.iloc[np.random.permutation(len(finalDataset))]  
    return mixedDataset  
def split_mixedDataset(mixedDataset):  
    mixedFeaturesDataset=mixedDataset.delete(outattribute)  
    mixedOutattributeDataset=mixedDataset.delete(features)  
    return mixedFeaturesDataset, mixedOutattributeDataset
```

Figure 15-Pseudo code used for mixing the final dataset and splitting it's in and out attributes

Data Transformation, Model training and testing were performed in the “**Model_creation_and_evaluation.py**” script. As seen in the pseudo code in Figure 17, first the in and out attributes are loaded as Input and Output, then the missing values (NaN) in the Input are removed, followed by a feature selection method that reduces the Input size. The Input can then be standardized, scaled or both, before being used to train and test the models.

Script: **Model_creation_and_evaluation**

Data Transformation:

```
Input,Output = mixedFeaturesDataset, mixedOutattributeDataset
Input = Imputer().fit_transform(Input)
if filtro == "y":
    Filter = VarianceThreshold (Input) or VarianceThreshold+RFE(Input)
else: Filter = Input
if StandarScaler == "y":
    use preprocessing.StandarScaler()
if MaxAbsScaler == "y":
    use preprocessing.MaxAbsScaler()
```

Model Training and testing:

```
if BaggingClassifier == "y":
    use BaggingClassifier()
create models (NB, ET, KNN, LR, GB, RF, SVM)
PECC=cross_validation.cross_val_score(model, Filter, Output, cv=5)
F1= cross_validation.cross_val_score(model, Filter, Output, cv=5, scoring="f1")
AUC=cross_validation.cross_val_score(model,Filter,Output,cv=5,scoring="roc_auc")

Saves all models created and respective statistical analysis files
```

Figure 17-Pseudo code for the data transformation, feature selection and model training and testing

After evaluating the results obtained, **Improving the models** can be achieved by selecting the best transformed data for each model and implementing ensemble methods like Voting classifiers as seen in the pseudo code in Figure 16.

```
HardVoteModel=VoteClassifier(estimator[models], voting="hard")
PECC=cross_validation.cross_val_score(HardVoteModel, Filter, Output, cv=5)
F1= cross_validation.cross_val_score(HardVoteModel,Filter,Output,cv=5,scoring="f1")
SoftVoteModel=VoteClassifier(estimator[models], voting="hard", weights=[weight of each model])
PECC=cross_validation.cross_val_score(HardVoteModel, Filter, Output, cv=5)
F1= cross_validation.cross_val_score(HardVoteModel,Filter,Output,cv=5,scoring="f1")
```

Figure 16-Pseudo code for the ensemble methods "VoteClassifier" with hard voting and weighted hard voting

Applying the model

After creating the models, they can be used to predict if a protein is or not a transporter protein. This can be done with the script “**MakePrediction.py**”. If the protein is a transporter protein a new model can be used to access the first level of the TCDB’s TC system as seen in the pseudo code in Figure 18.

Script: **MakePrediction.py**

```
Given the sequence and domain of a protein
Generates the features for that protein
Loads the models
Predicts if protein is transporter protein (1) or not a transporter protein (0)
If 1:
    Load TC system level 1 models
    Predicts the first level of the TC system (1,2,3,4 or 5)
```

Figure 18-Pseudo code for the prediction method. Predicts if a protein is a transporter protein and if it is, predicts the first level of the TC system from 1 to 5.

4.2 Workflow

The process of creating the datasets to the development of the final models used to make the predictions must follow the workflow presented in Figure 19.

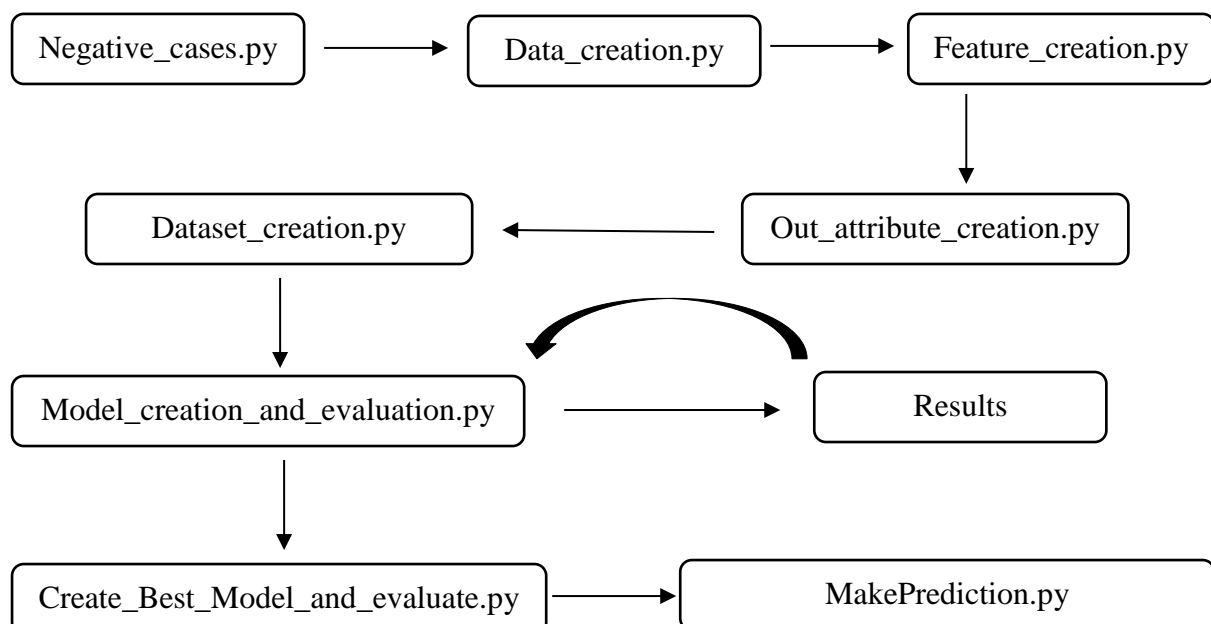


Figure 19- Workflow of the developed algorithm

5. RESULTS AND DISCUSSION

5.1 Case studies

Four different datasets were used (Dataset1, Dataset2, Dataset12345 and Transporter Dataset) to evaluate the developed models.

As detailed before, the first two datasets were used to create models capable of distinguishing transporter proteins and other transport related proteins from non-transporter related proteins and evaluate the performance of the models. The first dataset was used to evaluate the performance of the models when using all the TCDB's proteins. The second dataset was used to determine if the negative cases selected for this study would influence in the performance of the models. The Dataset12345 was used to create models capable of distinguishing transporter from non-transporter proteins and evaluate the performance of the models when using only well-known and curated transporter proteins from the TCDB (categories 8 and 9 of first level of the TC system removed). Finally, the Transporter Dataset was used to create and evaluate models capable of determining the TC system ID of a transporter protein, and eventually characterize the protein.

Before performing the training and testing of the models with the 5-fold cross-validation, the instances of all datasets were randomly mixed to ensure that the training and testing datasets were composed by, nearly, half positive and half negative cases. Thus, the folds were balanced.

5.1.1 Case study: Transport and Transport related protein models performance

To evaluate and improve the models performance, several different tests were performed, using different feature filters and pre-processing methods on Dataset1.

First, the dataset was used with all its 4510 input features and for the pre-processing of the dataset, only the "Imputer" function was used. The results of the five-fold cross validation test are shown in Table 9, in which the Naïve Bayes model was the lowest performing model (below 0.80) in both PECC and F1 scores while all the others were above.

Table 9- Mean of PECC, F1 and ROC-AUC scores after a 5-fold cross validation process using Dataset1 without filters.

<i>Models</i>	<i>PECC scores</i>	<i>F1 scores</i>	<i>ROC AUC</i>
<i>Naïve Bayes</i>	0.76 (+/- 0.00)	0.70 (+/- 0.01)	0.92 (+/- 0.00)
<i>ExtraTreeClassifier</i>	0.86 (+/- 0.00)	0.85 (+/- 0.00)	0.94 (+/- 0.00)
<i>K nearest Neighbours</i>	0.84 (+/- 0.01)	0.83 (+/- 0.01)	0.92 (+/- 0.00)
<i>Logistic Regression</i>	0.90 (+/- 0.00)	0.89 (+/- 0.01)	0.95 (+/- 0.00)
<i>GradientBoosting</i>	0.90 (+/- 0.00)	0.90 (+/- 0.01)	0.96 (+/- 0.00)
<i>RandomForest</i>	0.86 (+/- 0.00)	0.86 (+/- 0.01)	0.94 (+/- 0.00)
<i>SVM</i>	0.85 (+/- 0.01)	0.84 (+/- 0.01)	0.92 (+/- 0.00)

Dataset characteristics: Features- All; Pre-Processing- Imputer(); Filters- None; Using Dataset1 of size (27576,4510).

The second test using dataset1 used a variance threshold filter and the only pre-processing method used was the “Imputer”. The mean of the five-fold cross validation test results on the dataset is presented in Table 10.

Removing the features that had the same value for every protein (zero variance features) returned almost the same results as using the entire dataset1, but the process of creating and evaluating the models was much faster. This was expected since redundant features with the same result for every instance do not contribute to a better discrimination of the classes and will only slow the process. The Logistic Regression and Gradient Boosting models obtained the best results.

Table 10- Mean of PECC, F1 and ROC-AUC scores after a 5-fold cross validation process using a variant of Dataset1 with variance threshold filter.

<i>Models</i>	<i>PECC scores</i>	<i>F1 scores</i>	<i>ROC AUC</i>
<i>Naïve Bayes</i>	0.77 (+/- 0.01)	0.71 (+/- 0.01)	0.93 (+/- 0.00)
<i>ExtraTreeClassifier</i>	0.86 (+/- 0.01)	0.85 (+/- 0.00)	0.94 (+/- 0.00)
<i>K nearest Neighbours</i>	0.84 (+/- 0.01)	0.83 (+/- 0.01)	0.92 (+/- 0.00)
<i>Logistic Regression</i>	0.89 (+/- 0.00)	0.89 (+/- 0.01)	0.95 (+/- 0.00)
<i>GradientBoosting</i>	0.90 (+/- 0.00)	0.90 (+/- 0.00)	0.96 (+/- 0.00)
<i>RandomForest</i>	0.87 (+/- 0.00)	0.86 (+/- 0.00)	0.94 (+/- 0.00)
<i>SVM</i>	0.85 (+/- 0.00)	0.84 (+/- 0.00)	0.93 (+/- 0.00)

Dataset characteristics: Features- All; Pre-Processing- Imputer(); Filters- Variance threshold; Using Dataset1 of size (27576,2873).

To test the influence of a standardization of the values of the features on the performance of the models the “StandardScaler” function was included, maintaining the other pre-processing function “Imputer” and using the filter “Variance threshold”. The results of the five-fold cross-validation process are shown in Table 11, in which improvements signalled in green were obtained in the KNN, LR, and SVM models, while worse results were obtained in the NB model signalled in red.

These results show that the LR, the KNN and SVM (mainly the latter two), benefit greatly with a standardization of the features (e.g. Gaussian with 0 mean and unit variance), while the NB model does not.

Table 11- Mean of PECC, F1 and ROC-AUC scores after a 5-fold cross validation process using a variant of Dataset1 with StandardScaler().

<i>Models</i>	<i>PECC scores</i>	<i>F1 scores</i>	<i>ROC AUC</i>
<i>Naïve Bayes</i>	0.73 (+/- 0.00)	0.63 (+/- 0.01)	0.75 (+/- 0.00)
<i>ExtraTreeClassifier</i>	0.86 (+/- 0.00)	0.85 (+/- 0.01)	0.94 (+/- 0.00)
<i>K nearest Neighbours</i>	0.88 (+/- 0.00)	0.87 (+/- 0.00)	0.95 (+/- 0.00)
<i>Logistic Regression</i>	0.90 (+/- 0.00)	0.90 (+/- 0.00)	0.96 (+/- 0.00)
<i>GradientBoosting</i>	0.90 (+/- 0.00)	0.90 (+/- 0.01)	0.96 (+/- 0.00)
<i>RandomForest</i>	0.87 (+/- 0.00)	0.86 (+/- 0.00)	0.94 (+/- 0.00)
<i>SVM</i>	0.89 (+/- 0.00)	0.88 (+/- 0.01)	0.95 (+/- 0.00)

Dataset characteristics: Features- All; Pre-Processing- Imputer() and StandardScaler(); Filters- Variance threshold; Using Dataset1 of size (27576,2873).

The influence of scaling the values of the features on the performance of the models was assessed by the “MaxAbsScaler” function, keeping the other pre-processing function “Imputer” and using the filter “Variance threshold”. The results of the five-fold cross-validation process are presented in Table 12 where improvements in the KNN model were signalled in green while worse results in the NB, LR, and SVM models were signalled in red. The performance of the LR and SVM models reported an overall better performance when using the pre-processing function “StandardScaler” compared to the use of “MaxAbsScaler”. The only model that benefited from scaling the dataset with “MaxAbsScaler” was the KNN model reaching the same performance as when using the “StandardScaler” function.

Table 12- Mean of PECC, F1 and ROC-AUC scores after a 5-fold cross validation process using a variant of Dataset1 with MaxAbsScaler().

<i>Models</i>	<i>PECC scores</i>	<i>F1 scores</i>	<i>ROC AUC</i>
<i>Naïve Bayes</i>	0.73 (+/- 0.00)	0.64 (+/- 0.01)	0.75 (+/- 0.00)
<i>ExtraTreeClassifier</i>	0.86 (+/- 0.00)	0.85 (+/- 0.01)	0.94 (+/- 0.00)
<i>K nearest Neighbours</i>	0.88 (+/- 0.00)	0.87 (+/- 0.00)	0.95 (+/- 0.00)
<i>Logistic Regression</i>	0.89 (+/- 0.00)	0.88 (+/- 0.01)	0.95 (+/- 0.00)
<i>GradientBoosting</i>	0.90 (+/- 0.00)	0.90 (+/- 0.00)	0.96 (+/- 0.00)
<i>RandomForest</i>	0.87 (+/- 0.00)	0.86 (+/- 0.01)	0.94 (+/- 0.00)
<i>SVM</i>	0.82 (+/- 0.01)	0.79 (+/- 0.01)	0.89 (+/- 0.00)

Dataset characteristics: Features- All; Pre-Processing- Imputer() and MaxAbsScaler(); Filters- Variance threshold; Using Dataset1 of size (27576,2873).

Even though all features with zero variance were eliminated the number of features is still quite high (2873). Thus, a smaller dataset was used to reduce the computation time, to train and test the models, using an RFE filter after the “VarianceThreshold” filter that retained the best 1000 features using a LR model as base.

Again, the pre-processing function “Imputer” was employed and the results of this test are shown in Table 13. This time every single model got worse results when compared to the first test. This means that a lot of information about the proteins is lost when removing so many features.

Table 13- Mean of PECC, F1 and ROC-AUC scores after a 5-fold cross validation process using a variant of Dataset1 with a Variance threshold followed by a RFE filter.

<i>Models</i>	<i>PECC scores</i>	<i>F1 scores</i>	<i>ROC AUC</i>
<i>Naïve Bayes</i>	0.61 (+/- 0.00)	0.38 (+/- 0.01)	0.88 (+/- 0.00)
<i>ExtraTreeClassifier</i>	0.84 (+/- 0.01)	0.85 (+/- 0.00)	0.92 (+/- 0.00)
<i>K nearest Neighbours</i>	0.84 (+/- 0.00)	0.84 (+/- 0.00)	0.92 (+/- 0.00)
<i>Logistic Regression</i>	0.83 (+/- 0.00)	0.82 (+/- 0.00)	0.90 (+/- 0.00)
<i>GradientBoosting</i>	0.83 (+/- 0.01)	0.82 (+/- 0.01)	0.91 (+/- 0.00)
<i>RandomForest</i>	0.83 (+/- 0.01)	0.81 (+/- 0.01)	0.91 (+/- 0.00)
<i>SVM</i>	0.75 (+/- 0.01)	0.68 (+/- 0.01)	0.85 (+/- 0.00)

Dataset characteristics: Features- All; Pre-Processing- Imputer(); Filters- Variance threshold and RFE with LogisticRegression; Using Dataset1 of size (27576,1000).

As shown in Table 14, the analysis of all the previous tests' results demonstrates that the best outcomes are obtained when using the "Imputer" pre-processing function and the filter "Variance Threshold" for the models NB, ET, GB and RF and using the "Imputer" and "StandardScaler" pre-processing functions and the filter "Variance Threshold" for the models KNN, LR and SVM.

Table 14- Mean of PECC, F1 and ROC-AUC scores after a 5-fold cross validation process using a variant of Dataset1 with the best pre-processing parameters.

<i>Models</i>	<i>PECC scores</i>	<i>F1 scores</i>	<i>ROC AUC</i>
<i>Naïve Bayes</i>	0.77 (+/- 0.00)	0.71 (+/- 0.01)	0.93 (+/- 0.00)
<i>ExtraTreeClassifier</i>	0.86 (+/- 0.00)	0.85 (+/- 0.01)	0.94 (+/- 0.00)
<i>K nearest Neighbours</i>	0.88 (+/- 0.00)	0.87 (+/- 0.00)	0.95 (+/- 0.00)
<i>Logistic Regression</i>	0.90 (+/- 0.00)	0.90 (+/- 0.00)	0.96 (+/- 0.00)
<i>GradientBoosting</i>	0.90 (+/- 0.00)	0.90 (+/- 0.01)	0.96 (+/- 0.00)
<i>RandomForest</i>	0.87 (+/- 0.00)	0.86 (+/- 0.00)	0.94 (+/- 0.00)
<i>SVM</i>	0.89 (+/- 0.00)	0.88 (+/- 0.01)	0.95 (+/- 0.00)

Dataset characteristics for NB, ET, GB and RF models: Features- All; Pre-Processing- Imputer(); Filters- Variance threshold. Dataset characteristics for KNN, LR and SVM models: Features- All; Pre-Processing- Imputer() and StandardScaler; Filters- Variance threshold. Using Dataset1 of size (27576, 2873).

After obtaining the best results for each model, an ensemble method was implemented on them. The purpose of the implementation of an ensemble of the models was to combine their prediction capabilities into a better suited model. Two hard voting classifiers and two weighted

voting classifiers were tested on a five-fold cross validation process and the results are shown in Table 15. The ensemble method that used a hard vote process with only the three best performing models obtained the best scores with PECC score of 0.91(0/- 0.00) and F1 scores of 0.90 (+/- 0.00).

The fact that the best performing ensemble model (ensemble of the LR, GB and SVM models) showed little improvement compared to the performance of the best models (LR, GB and SVM) indicates that these models are not that diverse [13], meaning that they make errors in the same spaces of the test dataset. Identifying this “error space” is key to understanding why the models show nearly 10% of erroneous predictions.

Table 15- Mean performance of the ensemble methods: “VotingClassifier” with and without weights on the models using a variant of Dataset1.

<i>Ensemble</i>	<i>Models used</i>	<i>PECC scores</i>	<i>F1 scores</i>
<i>Hard Vote</i>	All	0.90 (+/- 0.00)	0.89 (+/- 0.00)
<i>Weighted Vote</i> <i>[1,2,2,3,3,2,3]</i>	All	0.90 (+/- 0.00)	0.89 (+/- 0.00)
<i>Weighted Vote</i> <i>[1,3,3,10,10,3,10]</i>	All	0.90 (+/- 0.00)	0.90 (+/- 0.00)
<i>Hard Vote</i>	LR, GB, SVM	0.91 (+/- 0.00)	0.90 (+/- 0.00)

Dataset characteristics for NB, ET, GB and RF models: Features- All; Pre-Processing- Imputer(); Filters- Variance threshold. Dataset characteristics for KNN, LR and SVM models: Features- All; Pre-Processing- Imputer() and StandarScaler; Filters- Variance threshold. Using Dataset1 of size (27576, 2873). Weights are according to the following order: NB, ET, KNN, LR, GB, RF, SVM.

Although some models reach 0.90 in accuracy and F1 scores and have a ROC-AUC score of more than 0.95, meaning they can easily differentiate negative from positive cases, they still wrongly predict 10% of the data. In order to try to evaluate where these 10% fit in the dataset, confusion matrices were created using a 5-fold cross validation process. The results for the first fold are shown below (Table 16) while the results for the other folds can be seen in Attachment I.

Table 16-Confusion Matrix for all models after a cross-validation process with training dataset representing 80% of the dataset1 and the test dataset the other 20%.

Predicted	NB model	0	1	__all__	Predicted	ET model	0	1	__all__
Actual					Actual				
0		TN (2758)	FP (38)	2796	0		TN (2624)	FP (172)	2796
1		FN (1240)	TP (1479)	2719	1		FN (617)	TP (2102)	2719
__all__		3998	1517	5515	__all__		3241	2274	5515
Predicted	KNN model	0	1	__all__	Predicted	LR model	0	1	__all__
Actual					Actual				
0		TN (2591)	FP (205)	2796	0		TN (2638)	FP (158)	2796
1		FN (456)	TP (2263)	2719	1		FN (396)	TP (2323)	2719
__all__		3047	2468	5515	__all__		3034	2481	5515
Predicted	GB model	0	1	__all__	Predicted	RF model	0	1	__all__
Actual					Actual				
0		TN (2663)	FP (133)	2796	0		TN (2615)	FP (181)	2796
1		FN (425)	TP (2294)	2719	1		FN (534)	TP (2185)	2719
__all__		3088	2427	5515	__all__		3149	2366	5515
Predicted	SVM model	0	1	__all__	Predicted				
Actual					Actual				
0		TN (2648)	FP (148)	2796	0				
1		FN (490)	TP (2229)	2719	1				
__all__		3138	2377	5515	__all__				

After a 5-fold cross validation process the results of each fold were identical to the results of the first fold observed in Table 16. Every model shows a higher number of FNs when compared to the number of FP, meaning that the models are misclassifying transport proteins as negative cases. The classification errors can be divided in about 7% of FNs and 3% FP.

Changing the UniProt queries used to create the negative cases might also help in the improvement of the models overall performance and decreasing the FN cases since there might

be some transporter proteins in the negative cases that cause the model to predict a non-transporter protein when the protein is a transporter protein.

Errors in the creation of the features might also be the cause of the high error value. When using the REST APIs for the creation of the features, errors thrown by the server lead to a classification of the feature as “0”. Also, if the script fails to get the Pfams for a positive case, all the features for the transport related Pfam domains will be considered “0” and the protein might be mistaken by a negative case increasing the FN error. This could be possibly solved by using all existing Pfams as a feature, marking “1” if the protein had that Pfam domain or “0” if it did not. The problem with the latter approach was the computation time involved in the process due to the size of the dataset.

5.1.2 Case study: Influence of negative cases in model performance

As previously mentioned, Dataset2 was created to assess whether the randomly chosen negative cases had impact in the performance of the models. The models were created using the pre-processing function “Imputer” and the filter “Variance Threshold”, evaluated using a five-fold cross-validation and the results are presented in Table 17. This results allow assuming, that although there are some improvements and some impairments in models when compared to the usage of both datasets, these differences are not significant thus suggesting that the randomly chosen negative cases have no influence in the models performance.

Table 17- Mean of PECC, F1 and ROC-AUC scores after a 5-fold cross validation process using a variant of Dataset2.

<i>Models</i>	<i>PECC scores</i>	<i>F1 scores</i>	<i>ROC AUC</i>
<i>Naïve Bayes</i>	0.77 (+/- 0.01)	0.71 (+/- 0.01)	0.92 (+/- 0.00)
<i>ExtraTreeClassifier</i>	0.86 (+/- 0.01)	0.85 (+/- 0.01)	0.94 (+/- 0.00)
<i>K nearest Neighbours</i>	0.84 (+/- 0.01)	0.82 (+/- 0.01)	0.91 (+/- 0.00)
<i>Logistic Regression</i>	0.90 (+/- 0.00)	0.89 (+/- 0.00)	0.96 (+/- 0.00)
<i>GradientBoosting</i>	0.90 (+/- 0.01)	0.90 (+/- 0.01)	0.96 (+/- 0.00)
<i>RandomForest</i>	0.87 (+/- 0.01)	0.86 (+/- 0.01)	0.94 (+/- 0.00)
<i>SVM</i>	0.85 (+/- 0.01)	0.84 (+/- 0.01)	0.92 (+/- 0.00)

Dataset characteristics: Features- All; Pre-Processing- Imputer(); Filters- Variance threshold; Using Dataset2 of size (27576,2873). Red- worse results when comparing with dataset1; Green- better results when comparing with dataset1

5.1.3 Case study: Transport protein models performance

Using the best method to pre-process and filter the features determined in the case study Dataset1, the models were created and evaluated by a 5-fold cross validation process and the results are shown in Table 18. Once again the LR and GB models obtained the best results.

Table 18- Mean of PECC, F1 and ROC-AUC scores after a 5-fold cross validation process using a variant of dataset12345.

<i>Models</i>	<i>PECC scores</i>	<i>F1 scores</i>	<i>ROC AUC</i>
<i>Naïve Bayes</i>	0.79 (+/- 0.01)	0.74 (+/- 0.02)	0.92 (+/- 0.00)
<i>ExtraTreeClassifier</i>	0.86 (+/- 0.00)	0.85 (+/- 0.00)	0.94 (+/- 0.00)
<i>K nearest Neighbours</i>	0.84 (+/- 0.00)	0.83 (+/- 0.00)	0.91 (+/- 0.00)
<i>Logistic Regression</i>	0.89 (+/- 0.01)	0.88 (+/- 0.00)	0.95 (+/- 0.00)
<i>GradientBoosting</i>	0.90 (+/- 0.01)	0.90 (+/- 0.01)	0.95 (+/- 0.00)
<i>RandomForest</i>	0.87 (+/- 0.00)	0.86 (+/- 0.01)	0.94 (+/- 0.00)
<i>SVM</i>	0.85 (+/- 0.00)	0.84 (+/- 0.00)	0.92 (+/- 0.00)

Dataset characteristics for NB, ET, GB and RF models: Features- All; Pre-Processing- Imputer(); Filters- Variance threshold. Dataset characteristics for KNN, LR and SVM models: Features- All; Pre-Processing- Imputer() and StandarScaler; Filters- Variance threshold. Using Dataset12345 of size (23434, 2873).

The creation and evaluation of different ensemble voting classifiers returned the following results (Table 19).

Table 19- Mean performance of the ensemble methods: "VotingClassifier" with and without weights on the models.

<i>Ensemble</i>	<i>Models used</i>	<i>PECC scores</i>	<i>F1 scores</i>
<i>Hard Vote</i>	All	0.90 (+/- 0.00)	0.89 (+/- 0.00)
<i>Hard Vote</i>	LR, GB, SVM	0.91 (+/- 0.00)	0.90 (+/- 0.00)
<i>Hard Vote</i>	LR, GB, RF	0.90 (+/- 0.01)	0.90 (+/- 0.00)
<i>Weighted Hard Vote</i> <i>[1,2,2,3,3,2,3]</i>	All	0.90 (+/- 0.00)	0.90 (+/- 0.00)
<i>Weighted Hard Vote</i> <i>[1,3,3,10,10,3,10]</i>	All	0.91 (+/- 0.00)	0.90 (+/- 0.00)
<i>Weighted Hard Vote</i> <i>[1,6,3,10,10,7,4]</i>	All	0.90 (+/- 0.00)	0.89 (+/- 0.00)

Dataset characteristics for NB, ET, GB and RF models: Features- All; Pre-Processing- Imputer(); Filters- Variance threshold. Dataset characteristics for KNN, LR and SVM models: Features- All; Pre-Processing- Imputer() and StandarScaler; Filters- Variance threshold. Using Dataset12345 of size (23434, 2873). Weights are according to the following order: NB, ET, KNN, LR, GB, RF, SVM.

The best ensemble of the models used with Dataset12345 showed the same PECC and F1 scores as the best ensemble using Dataset1. These results suggest that adding proteins from classes 8 and 9 of the first level of the TC system does not improve the overall performance of the final ensemble model, although having some impact in the performance of some individual models (NB, KNN and SVM).

5.1.4 Case study: Transport proteins characterization models performance

To characterize the transport proteins the best method to pre-process and filter the features determined in the case study Dataset1 was used, and the models were created and evaluated by a 5-fold cross validation process and the results are presented in Table 20, where the LR showed the best performance out of all the other models with a PECC score of 0.84 (+/- 0.01).

Table 20-- Mean of PECC scores after a 5-fold cross validation process using a variation of the transporter dataset.

<i>Models</i>	<i>PECC scores</i>
<i>Naïve Bayes</i>	0.55 (+/- 0.00)
<i>ExtraTreeClassifier</i>	0.81 (+/- 0.00)
<i>K nearest Neighbours</i>	0.74 (+/- 0.00)
<i>Logistic Regression</i>	0.84 (+/- 0.01)
<i>GradientBoosting</i>	0.82 (+/- 0.01)
<i>RandomForest</i>	0.79 (+/- 0.00)
<i>SVM</i>	0.74 (+/- 0.01)

Dataset characteristics for NB, ET, GB and RF models: Features- All; Pre-Processing- Imputer(); Filters- Variance threshold. Dataset characteristics for KNN, LR and SVM models: Features- All; Pre-Processing- Imputer() and StandarScaler; Filters- Variance threshold. Using Dataset12345 of size (11717, 2873).

Creating and evaluating different ensemble voting classifiers returned the following results (Table 21), in which the hard vote with all the models and the weighted hard vote with weights [1,3,3,10,10,3,10] showed the best performance with PECC score of 0.87 (+/- 0.01).

Table 21-- Mean performance of the ensemble methods: "VotingClassifier" with and without weights on the models.

<i>Ensemble</i>	<i>Models used</i>	<i>PECC scores</i>
<i>Hard Vote</i>	All	0.87 (+/- 0.01)
<i>Hard Vote</i>	LR, GB, SVM	0.86 (+/- 0.01)
<i>Hard Vote</i>	LR, GB, ET	0.84 (+/- 0.01)
<i>Weighted Vote</i> <i>[1,2,2,3,3,2,3]</i>	All	0.87 (+/- 0.01)
<i>Weighted Vote</i> <i>[1,3,3,10,10,3,10]</i>	All	0.86 (+/- 0.01)
<i>Weighted Vote</i> <i>[1,8,3,10,9,6,3]</i>	All	0.85 (+/- 0.01)

Dataset characteristics for NB, ET, GB and RF models: Features- All; Pre-Processing- Imputer(); Filters- Variance threshold. Dataset characteristics for KNN, LR and SVM models: Features- All; Pre-Processing- Imputer() and StandarScaler; Filters- Variance threshold. Using Dataset12345 of size (11717, 2873). Weights are according to the following order: NB, ET, KNN, LR, GB, RF, SVM.

Confusion matrices were created, using a five-fold cross validation process, to evaluate the wrong classifications in the dataset. The results for the first fold are presented in Table 22,

Table 23, Table 24, Table 25, Table 26, Table 27 and Table 28 while the results for the other folds are shown in Attachment II.

Table 22- Confusion Matrix for the NB model after a cross-validation process with training dataset representing 80% of the transporter dataset and the test dataset the other 20%.

NB model	Predicted	1	2	3	4	5	__all__
Actual							
1		320	7	0	0	480	807
2		1	471	3	1	223	699
3		10	9	425	1	299	744
4		0	0	0	29	22	51
5		0	0	1	0	41	42
__all__		331	487	429	31	1065	2343

Table 23- Confusion Matrix for the ET model after a cross-validation process with training dataset representing 80% of the transporter dataset and the test dataset the other 20%.

ET model	Predicted	1	2	3	4	5	__all__
Actual							
1		737	38	32	0	0	807
2		65	623	11	0	0	699
3		166	78	499	1	0	744
4		10	16	6	19	0	51
5		22	14	3	0	3	42
__all__		1000	769	551	20	3	2343

Table 24- Confusion Matrix for the KNN model after a cross-validation process with training dataset representing 80% of the transporter dataset and the test dataset the other 20%.

KNN model	Predicted	1	2	3	4	5	__all__
Actual							
1		708	43	54	1	1	807
2		30	645	23	1	0	699
3		98	49	591	2	4	744
4		4	5	9	33	0	51
5		8	10	6	0	18	42
__all__		848	752	683	37	23	2343

Table 25- Confusion Matrix for the LR model after a cross-validation process with training dataset representing 80% of the transporter dataset and the test dataset the other 20%.

LR model	Predicted	1	2	3	4	5	__all__
Actual							
1		729	31	45	1	1	807
2		47	630	21	1	0	699
3		132	39	567	1	5	744
4		5	8	8	30	0	51
5		16	5	4	1	16	42
__all__		929	713	645	34	22	2343

Table 26- Confusion Matrix for the GB model after a cross-validation process with training dataset representing 80% of the transporter dataset and the test dataset the other 20%.

GB model	Predicted	1	2	3	4	5	__all__
Actual							
1		701	43	62	0	1	807
2		53	616	30	0	0	699
3		143	56	539	1	5	744
4		6	4	12	29	0	51
5		16	7	4	0	15	42
__all__		919	729	647	30	21	2343

Table 27-Confusion Matrix for the RF model after a cross-validation process with training dataset representing 80% of the transporter dataset and the test dataset the other 20%.

RF model	Predicted	1	2	3	4	5	__all__
Actual							
1		711	44	52	0	0	807
2		85	592	22	0	0	699
3		173	66	505	0	0	744
4		11	15	15	10	0	51
5		24	11	6	0	1	42
__all__		1004	728	600	10	1	2343

Table 28- Confusion Matrix for the SVM model after a cross-validation process with training dataset representing 80% of the transporter dataset and the test dataset the other 20%.

SVM model	Predicted	1	2	3	4	5	__all__
Actual							
1		761	43	3	0	0	807
2		55	639	4	1	0	699
3		232	60	446	1	5	744
4		12	8	2	29	0	51
5		17	7	4	0	14	42
__all__		1077	757	459	31	19	2343

To better evaluate the models a table containing the sensitivity (recall) and specificity for each class was created (Table 29). Maximum (green) and minimum values (red) of sensibility and specificity are marked for each class.

As seen in Table 29 for class 1, the model with highest sensibility (0.943) was the SVM model but it also had the lowest specificity (0.794). On the contrary, the NB model showed the lowest sensitivity (0.396) while having the highest specificity (0.993).

For class 2 the KNN model showed the highest sensitivity (0.923) and the NB model the lowest (0,674). The NB model reached the highest specificity (0,990) while the ET model had the lowest (0,911).

For class 3 the KNN model showed the highest sensitivity (0,794) and the NB model the lowest (0,571). The NB model had the highest specificity (0,997) and the GB model the lowest (0,932).

For class 4 the KNN showed the highest sensitivity (0,647) and the RF model the lowest (0,196). The RF model had the highest specificity (1) and the KNN and LR had the lowest (0,998).

For class 5 the NB model had the highest sensitivity (0,976) and the RF model the lowest (0,024). The ET and RF models had the highest specificity (1) and the NB model the lowest (0,555).

Table 29-Table of sensibility and Specificity for each class predicted by every model

NB model	Sensitivity	Specificity	ET model	Sensitivity	Specificity
1	0,39653036	0,99283854	1	0,91325898	0,828776
2	0,67381974	0,99026764	2	0,89127325	0,9111922
3	0,57123656	0,99749844	3	0,67069892	0,9674797
4	0,56862745	0,9991274	4	0,37254902	0,9995637
5	0,97619048	0,5549761	5	0,07142857	1
KNN model	Sensitivity	Specificity	LR model	Sensitivity	Specificity
1	0,87732342	0,90885417	1	0,90334572	0,8697917
2	0,92274678	0,93491484	2	0,90128755	0,9495134
3	0,79435484	0,94246404	3	0,76209677	0,9512195
4	0,64705882	0,9982548	4	0,58823529	0,9982548
5	0,42857143	0,99782703	5	0,38095238	0,9973924
GB model	Sensitivity	Specificity	RF model	Sensitivity	Specificity
1	0,86864932	0,85807292	1	0,88104089	0,8092448
2	0,88125894	0,93309002	2	0,84692418	0,9172749
3	0,72446237	0,93245779	3	0,67876344	0,9405879
4	0,56862745	0,9995637	4	0,19607843	1
5	0,35714286	0,99739244	5	0,02380952	1
SVM model	Sensitivity	Specificity			
1	0,94299876	0,79427083			
2	0,91416309	0,92822384			
3	0,59946237	0,99186992			
4	0,56862745	0,9991274			
5	0,33333333	0,99782703			

High sensitivity and specificity values are representative of few FN and FP respectively. Although some models in Table 29 showed high specificity for some classes (NB model for classes 1,2,3 and 4; RF model for classes 4 and 5; ET model for class 5) they also showed low

sensitivity for the same class, meaning that when they classify a protein to be in that class the number of FP is almost 0, but the number of FN for that class is high too. Preferring classifications from models with high specificity for a specific class might lower the number of FP.

6. CONCLUSIONS AND FURTHER WORK

For this work, seven different machine learning models as well as ensembles of the models were trained and tested, using a five-fold cross validation process, on different datasets to identify and classify transport proteins.

The results for the performance of the models using datasets with different selections of negative cases showed little variance meaning that the chosen negative cases have little influence in the ability of the models to distinguish between transporter and non-transporter proteins.

Evaluation of the results for the Dataset1 shows that, to obtain the best performance, the pre-processing functions “Imputer” and “StandardScaler” and the filter “Variance Threshold” must be used for models KNN, LR and SVM. “StandardScaler” only worsened the performance of the NB model. Hence, for models NB, ET, GB and RF the best performance is achieved with the pre-processing function “Imputer” and the “Variance Threshold” filter. Using an ensemble of the best performing models, namely LR, GB and SVM, a hard voting classifier was implemented achieving a PECC score of 0.91 (+/- 0.00) and a F1score of 0.90 (+/- 0.00).

Creating confusion matrices in a 5-fold cross-validation process of the dataset, the models show a high number of FNs compared to the FPs (almost 3 times bigger), meaning that the models are having more trouble correctly identifying transporter proteins than correctly identifying non-transporter proteins but the confidence level of the classification of a protein as transporter is higher than the confidence level associated with the classification of the protein as non-transporter. Proteins classified as non-transporters should be further analysed.

The NB model presented the lowest FP value, meaning that whenever this model predicts a protein to be a transporter protein the error associated with that prediction is low, when compared to the other models.

The best performance using the Dataset12345 was obtained with the ensemble method hard voting classifier, using the models LR, GB and SVM achieving a PECC score of 0.91 (+/- 0.00) and a F1score of 0.90 (+/- 0.00). Although this model obtained the same performance scores as the best performing model for Dataset1, this model only contains transport proteins as positive cases (TCDB’s classes 8 and 9 were removed) and is thus better suited for the classification of transport proteins, whilst Dataset1 (which contains all TCDB records) should be used for classifying all proteins associated to the transport phenomena.

Although these models show an error of 9%, due to the low number of FP when the model predicts a protein to be transporter protein the error is much lower and the level of confidence of the prediction much higher. In the case where the model predicts a protein to be a negative case the error associated with that prediction is higher because of the number of FN presented by the models.

For the transporter Dataset, although the models obtained lower PECC scores than the other datasets (which was expected since this model has five different classes) the ensemble of all the models created with a hard voting classifier obtained a PECC score of 0.87 (+/- 0.01).

Therefore, after a protein is classified as transporter protein, with the previous dataset's models, it can be characterized in the first level of the TC system with an accuracy of 87% using the voting classifier model developed with the Transporter Dataset.

Although some models showed almost 100% specificity for a class they also had low sensitivity values for that class.

Implementing algorithms that combine the best performing models for each case can greatly reduce the error associated with the classifications.

Finally, the models developed in this thesis allowed for the identification of transport and transport related proteins with an accuracy of 91% and the following characterization into the first level of the TC system with an accuracy of 87% given only the protein's sequence making them a quick and good solution for the identification and characterization of transport proteins given a new unannotated genome.

The code of the algorithm developed in this work is freely available in GitHub at "<https://github.com/DanielVarzim/Master-s-Thesis->".

Although most goals proposed in this thesis were accomplished, there is always room for improvement. These points can be divided into improvement of the work done and integration of the tool with *merlin* software:

- Optimize the code for a faster creation of the features.
- Search for new protein features to improve the overall performance of the models.
- Test new filters and wrapper methods for the dataset as well as more pre-processing utilities.
- Test the models with different sets of parameters using "sklearn's" "GridSearch" method to improve their performances.

- Create models to further characterize the transporter proteins into the several levels of the TC system.
- Integrate the tool with the open-source software framework *merlin*: Create wrappers between these models and *merlin* structures; Create operations for applying classifiers to new sequences, enriching genome scale metabolic models.

BIBLIOGRAPHY

- [1] O. Dias, M. Rocha, E. C. Ferreira, and I. Rocha, “Reconstructing genome-scale metabolic models with merlin.,” *Nucleic Acids Res.*, vol. 43, no. 8, pp. 3899–910, Apr. 2015.
- [2] S. C. Schuster, “Next-generation sequencing transforms today ’ s biology,” *Nat. Methods*, vol. 5, no. 1, pp. 16–18, 2008.
- [3] A. M. Feist, C. S. Henry, J. L. Reed, M. Krummenacker, A. R. Joyce, P. D. Karp, L. J. Broadbelt, V. Hatzimanikatis, and B. Ø. Palsson, “A genome-scale metabolic reconstruction for Escherichia coli K-12 MG1655 that accounts for 1260 ORFs and thermodynamic information.,” *Mol. Syst. Biol.*, vol. 3, no. 121, p. 121, Jun. 2007.
- [4] M. Punta, L. R. Forrest, H. Bigelow, A. Kernytsky, J. Liu, and B. Rost, “Membrane protein prediction methods.,” *Methods*, vol. 41, no. 4, pp. 460–74, Apr. 2007.
- [5] M. H. Saier, V. S. Reddy, D. G. Tamang, and A. Västermark, “The transporter classification database.,” *Nucleic Acids Res.*, vol. 42, no. Database issue, pp. D251-8, Jan. 2014.
- [6] I. Thiele and B. Ø. Palsson, “A protocol for generating a high-quality genome-scale metabolic reconstruction.,” *Nat. Protoc.*, vol. 5, no. 1, pp. 93–121, Jan. 2010.
- [7] J. J. Hamilton and J. L. Reed, “Software platforms to facilitate reconstructing genome-scale metabolic networks.,” *Environ. Microbiol.*, vol. 16, no. 1, pp. 49–59, Jan. 2014.
- [8] A. M. Feist, M. J. Herrgård, I. Thiele, J. L. Reed, and B. Ø. Palsson, “Reconstruction of biochemical networks in microorganisms.,” *Nat. Rev. Microbiol.*, vol. 7, no. 2, pp. 129–43, Feb. 2009.
- [9] O. Dias, D. G. Gomes, P. Vilaça, J. Cardoso, M. Rocha, E. C. Ferreira, and I. Rocha, “Genome-wide Semi-automated Annotation of Transporter Systems,” *Submitted*, vol. XX, no. X, pp. 1–14, 2014.
- [10] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, “Machine learning: An artificial intelligence approach. Vol. III,” vol. III, 1986.
- [11] M. Awad and R. Khanna, *Efficient Learning Machines*. Berkeley, CA: Apress, 2015.
- [12] T. M. Mitchell, *Machine Learning*. McGraw Hill, 1997.
- [13] M. Rocha, P. Cortez, and J. M. Neves, *Análise Inteligente de Dados algoritmos e implementação em JAVA*. FCA, 2008.
- [14] “Glossary of Terms Journal of Machine Learning.” [Online]. Available: <http://robotics.stanford.edu/~ronnyk/glossary.html>. [Accessed: 19-Jan-2016].
- [15] D. R. Wilson and T. R. Martinez, “Value difference metrics for continuously valued attributes,” *Proc. Int. Conf. Artif. Intell. Expert Syst. Neural Networks*, no. Vdm, pp. 11–14, 1996.
- [16] A. Smola and S. V. N. Vishwanathan, *Introduction to Machine Learning*. Cambridge University Press, 2008.
- [17] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Linear Regression Analysis*, 5th ed. John Wiley & Sons, Inc, 2012.
- [18] M. L. Fisher, “The Lagrangian relaxation method for solving integer programming problems,” *INFORMS*, pp. 1861–1871, Dec-2004.
- [19] “1.11. Ensemble methods — scikit-learn 0.17 documentation.” [Online]. Available: <http://scikit-learn.org/stable/modules/ensemble.html>. [Accessed: 29-Jan-2016].
- [20] R. Kohavi, “A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection,” *Int. Jt. Conf. Artif. Intell.*, vol. 14, no. 12, pp. 1137–1143, 1995.
- [21] Y. Saeys, I. Inza, and P. Larranaga, “A review of feature selection techniques in bioinformatics,” *Bioinformatics*, vol. 23, no. 19, pp. 2507–2517, Oct. 2007.

- [22] W. R. Pearson, “An introduction to sequence similarity (‘homology’) searching.,” *Curr. Protoc. Bioinformatics*, vol. Chapter 3, no. SUPPL.42, p. Unit3.1, Jun. 2013.
- [23] T. Madden, “The BLAST Sequence Analysis Tool.” National Center for Biotechnology Information (US), 15-Mar-2013.
- [24] “BLAST Homepage and Selected Search Pages.” [Online]. Available: ftp://ftp.ncbi.nlm.nih.gov/pub/factsheets/HowTo_BLASTGuide.pdf. [Accessed: 02-Feb-2016].
- [25] R. D. Finn, J. Clements, W. Arndt, B. L. Miller, T. J. Wheeler, F. Schreiber, A. Bateman, and S. R. Eddy, “HMMER web server: 2015 update.,” *Nucleic Acids Res.*, vol. 43, no. W1, pp. W30-8, Jul. 2015.
- [26] A. Ben-hur and D. Brutlag, “Chapter 31 Sequence Motifs: Highly Predictive Features of Protein Function,” *Enzyme*, vol. 645, pp. 625–645, 2006.
- [27] “Box 2 : Applied bioinformatics for the identification of regulatory elements : Nature Reviews Genetics.” [Online]. Available: http://www.nature.com/nrg/journal/v5/n4/box/nrg1315_BX2.html. [Accessed: 03-Feb-2016].
- [28] X. Chen, T. Jiang, and Z. Fan, “LEARNING POSITION WEIGHT MATRICES FROM SEQUENCE AND EXPRESSION DATA The discovery of regulatory motifs in DNA sequences is very important in systems biology as it is the first and important step towards understanding the mechanisms that regulate the expre,” *Sci. Technol.*, pp. 1–12, 2007.
- [29] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm,” *ournal R. Stat. Soc. Ser. B (Statistical Methodol.*, vol. 39, no. 1, pp. 1–38, 1977.
- [30] T. L. Bailey and C. Elkan, “Fitting a Mixture Model by Expectation Maximization to Discover Motifs in Bipolymers,” pp. 1–33, 1994.
- [31] M. K. Das and H.-K. Dai, “A survey of DNA motif finding algorithms.,” *BMC Bioinformatics*, vol. 8 Suppl 7, no. Suppl 7, p. S21, 2007.
- [32] S. Eddy, “Profile hidden Markov models.,” *Bioinformatics*, vol. 14, no. 9, pp. 755–763, 1998.
- [33] R. D. Finn, J. Clements, and S. R. Eddy, “HMMER web server: interactive sequence similarity searching.,” *Nucleic Acids Res.*, vol. 39, no. Web Server issue, pp. W29-37, Jul. 2011.
- [34] R. Franklin Weaver, *Molecular biology*. 2005.
- [35] Christopher Walsh, *Posttranslational Modification of Proteins: Expanding Nature’s Inventory*. Roberts and Company Publishers, 2006.
- [36] “Overview of Post-Translational Modification,” *ThermoFisher Scientific*. [Online]. Available: <https://www.thermofisher.com/pt/en/home/life-science/protein-biology/protein-biology-learning-center/protein-biology-resource-library/pierce-protein-methods/overview-post-translational-modification.html>. [Accessed: 04-Feb-2016].
- [37] A. Kessel and N. Ben-Tal, *Introduction to proteins*. 2011.
- [38] N. Y. Yu, J. R. Wagner, M. R. Laird, G. Melli, S. Rey, R. Lo, P. Dao, S. Cenk Sahinalp, M. Ester, L. J. Foster, and F. S. L. Brinkman, “PSORTb 3.0: Improved protein subcellular localization prediction with refined localization subcategories and predictive capabilities for all prokaryotes,” *Bioinformatics*, vol. 26, no. 13, pp. 1608–1615, 2010.
- [39] S.-A. Chen, Y.-Y. Ou, T.-Y. Lee, and M. M. Gromiha, “Prediction of transporter targets using efficient RBF networks with PSSM profiles and biochemical properties.,” *Bioinformatics*, vol. 27, no. 15, pp. 2062–7, Aug. 2011.
- [40] A. Pierleoni, P. L. Martelli, P. Fariselli, and R. Casadio, “BaCelLo: A balanced

- subcellular localization predictor,” *Bioinformatics*, vol. 22, no. 14, pp. 408–416, 2006.
- [41] A. Garg and G. P. S. Raghava, “ESLpred2: improved method for predicting subcellular localization of eukaryotic proteins,” *BMC Bioinformatics*, vol. 9, p. 503, 2008.
- [42] P. Horton, K. J. Park, T. Obayashi, N. Fujita, H. Harada, C. J. Adams-Collier, and K. Nakai, “WoLF PSORT: Protein localization predictor,” *Nucleic Acids Res.*, vol. 35, no. SUPPL.2, pp. 585–587, 2007.
- [43] H. Chen, N. Huang, and Z. Sun, “SubLoc: a server/client suite for protein subcellular location based on SOAP,” *Bioinformatics*, vol. 22, no. 3, pp. 376–7, Feb. 2006.
- [44] T. Goldberg, M. Hecht, T. Hamp, T. Karl, G. Yachdav, N. Ahmed, U. Altermann, P. Angerer, S. Ansorge, K. Balasz, M. Bernhofer, A. Betz, L. Cizmadija, K. T. Do, J. Gerke, R. Greil, V. Joerdens, M. Hastreiter, K. Hembach, M. Herzog, M. Kalemantov, M. Kluge, A. Meier, H. Nasir, U. Neumaier, V. Prade, J. Reeb, A. Sorokoumov, I. Troshani, S. Vorberg, S. Waldruff, J. Zierer, H. Nielsen, and B. Rost, “LocTree3 prediction of localization,” *Nucleic Acids Res.*, vol. 42, no. Web Server issue, pp. W350-5, Jul. 2014.
- [45] K.-C. Chou and H.-B. Shen, “Cell-PLoc: a package of Web servers for predicting subcellular localization of proteins in various organisms,” *Nat. Protoc.*, vol. 3, no. 2, pp. 153–62, Jan. 2008.
- [46] C.-S. Yu, 1, Y.-C. Chen, 2, C.-H. Lu, 2, J.-K. Hwang, and 3 1, 2, “Prediction of Protein Subcellular Localization,” *Proteins*, vol. 70, no. 2, pp. 311–319, 2008.
- [47] M. Bhasin, A. Garg, and G. P. S. Raghava, “PSLpred: prediction of subcellular localization of bacterial proteins,” *Bioinformatics*, vol. 21, no. 10, pp. 2522–4, May 2005.
- [48] G. Aarti, Bhasin Manoj, and Raghava, “Pslpred: A svm based method for the subcellular localization of Prokaryotic proteins,” *imtech*. [Online]. Available: <http://www.imtech.res.in/raghava/pslpred/algo.html>. [Accessed: 25-Apr-2015].
- [49] M. Cserzo, F. Eisenhaber, B. Eisenhaber, and I. Simon, “TM or not TM: transmembrane protein prediction with low false positive rate using DAS-TMfilter,” *Bioinformatics*, vol. 20, no. 1, pp. 136–7, Jan. 2004.
- [50] G. E. Tusnády and I. Simon, “The HMMTOP transmembrane topology prediction server,” *Bioinformatics*, vol. 17, no. 9, pp. 849–50, Sep. 2001.
- [51] L. Käll, A. Krogh, and E. L. L. Sonnhammer, “Advantages of combined transmembrane topology and signal peptide prediction—the Phobius web server,” *Nucleic Acids Res.*, vol. 35, no. SUPPL.2, pp. 429–432, 2007.
- [52] G. Yachdav, E. Kloppmann, L. Kaján, M. Hecht, T. Goldberg, T. Hamp, P. Hönigschmid, A. Schafferhans, M. Roos, M. Bernhofer, L. Richter, H. Ashkenazy, M. Punta, A. Schlessinger, Y. Bromberg, R. Schneider, G. Vriend, C. Sander, N. Ben-Tal, and B. Rost, “PredictProtein—an open resource for online prediction of protein structural and functional features,” *Nucleic Acids Res.*, vol. 42, no. Web Server issue, pp. W337-43, Jul. 2014.
- [53] L. L. S. Erik, V. H. Gunnar, and K. Anders, “A hidden Markov model for predicting transmembrane helices in protein sequences,” pp. 175–182, 1998.
- [54] F. S. Berven, K. Flikka, H. B. Jensen, and I. Eidhammer, “BOMP: a program to predict integral beta-barrel outer membrane proteins encoded within genomes of Gram-negative bacteria,” *Nucleic Acids Res.*, vol. 32, no. Web Server issue, pp. W394-9, Jul. 2004.
- [55] P. G. Bagos, T. D. Liakopoulos, I. C. Spyropoulos, and S. J. Hamodrakas, “PRED-TMBB: A web server for predicting the topology of ??-barrel outer membrane proteins,” *Nucleic Acids Res.*, vol. 32, no. WEB SERVER ISS., pp. 400–404, 2004.
- [56] S. Möller, M. D. Croning, and R. Apweiler, “Evaluation of methods for the prediction of membrane spanning regions,” *Bioinformatics*, vol. 17, no. 7, pp. 646–53, Jul. 2001.
- [57] N. K. Mishra, J. Chang, and P. X. Zhao, “Prediction of Membrane Transport Proteins

- and Their Substrate Specificities Using Primary Sequence Information,” *PLoS One*, vol. 9, no. 6, p. e100278, Jun. 2014.
- [58] “BLAST TOPICS.” [Online]. Available: https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=BlastHelp.
- [59] M. H. Saier, V. S. Reddy, D. G. Tamang, and A. Vastermark, “The Transporter Classification Database,” *Nucleic Acids Res.*, vol. 42, no. D1, pp. D251–D258, 2014.
- [60] M. H. Saier, “TCDB: the Transporter Classification Database for membrane transport protein analyses and information,” *Nucleic Acids Res.*, vol. 34, no. 90001, pp. D181–D186, 2006.
- [61] “TCDB » HOME.” [Online]. Available: <http://tcdb.org/>.
- [62] Q. Ren, K. Chen, and I. T. Paulsen, “TransportDB: a comprehensive database resource for cytoplasmic membrane transport systems and outer membrane channels,” *Nucleic Acids Res.*, vol. 35, no. Database, pp. D274–D279, Jan. 2007.
- [63] Q. Ren, “TransAAP : A Web-based Transporter Annotation Tool User Guide,” no. 301, 2007.
- [64] “UniProt.” [Online]. Available: <http://www.uniprot.org/>.
- [65] “Biopython Tutorial and Cookbook.” [Online]. Available: <http://biopython.org/DIST/docs/tutorial/Tutorial.html#htoc6>. [Accessed: 06-Feb-2016].
- [66] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, “Scikit-learn: Machine Learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2001.
- [67] “User guide: contents — scikit-learn 0.17 documentation.” [Online]. Available: http://scikit-learn.org/stable/user_guide.html. [Accessed: 07-Feb-2016].
- [68] “Phobius.” [Online]. Available: <http://phobius.sbc.su.se/index.html>.
- [69] “CBU - Computational Biology Unit- BOMP.” [Online]. Available: <http://services.cbu.uib.no/tools/bomp>.
- [70] “LocTree3 - Protein Subcellular Localization Prediction Server.” [Online]. Available: <https://roslab.org/services/loctree3/>.

ATTACHMENT I

AI₁- Confusion Matrix for all models after the second fold of the cross-validation process.

<i>Predicted</i>		NB model			<i>Predicted</i>		ET model		
		0	1	__all__			0	1	__all__
<i>Actual</i>					<i>Actual</i>				
0		TN (2666)	FP (56)	2722	0		TN (2570)	FP (152)	2722
1		FN (1211)	TP (1582)	2793	1		FN (612)	TP (2181)	2793
__all__		3877	1638	5515	__all__		3182	2333	5515
<i>Predicted</i>		KNN model			<i>Predicted</i>		LR model		
		0	1	__all__			0	1	__all__
<i>Actual</i>					<i>Actual</i>				
0		TN (2516)	FP (206)	2722	0		TN (2546)	FP (176)	2796
1		FN (457)	TP (2336)	2793	1		FN (356)	TP (2437)	2719
__all__		2973	2542	5515	__all__		2902	2613	5515
<i>Predicted</i>		GB model			<i>Predicted</i>		RF model		
		0	1	__all__			0	1	__all__
<i>Actual</i>					<i>Actual</i>				
0		TN (2591)	FP (131)	2722	0		TN (2559)	FP (163)	2722
1		FN (384)	TP (2409)	2793	1		FN (533)	TP (2260)	2793
__all__		2975	2540	5515	__all__		3092	2423	5515
<i>Predicted</i>		SVM model			<i>Predicted</i>				
		0	1	__all__					
<i>Actual</i>					<i>Actual</i>				
0		TN (2582)	FP (140)	2722	0				
1		FN (436)	TP (2357)	2793	1				
__all__		3018	2497	5515	__all__				

AI₂- Confusion Matrix for all models after the third fold of the cross-validation process.

<i>Predicted</i>		NB model			<i>Predicted</i>		ET model		
		0	1	__all__			0	1	__all__
<i>Actual</i>					<i>Actual</i>				
0		TN (2733)	FP (44)	2777	0		TN (2635)	FP (142)	2777
1		FN (1189)	TP (1549)	2738	1		FN (614)	TP (2124)	2738
__all__		3922	1593	5515	__all__		3249	2266	5515
<i>Predicted</i>		KNN model			<i>Predicted</i>		LR model		
<i>Actual</i>					<i>Actual</i>				
0		TN (2577)	FP (200)	2777	0		TN (2627)	FP (150)	2777
1		FN (484)	TP (2254)	2738	1		FN (410)	TP (2328)	2738
__all__		3061	2454	5515	__all__		3037	2478	5515
<i>Predicted</i>		GB model			<i>Predicted</i>		RF model		
<i>Actual</i>					<i>Actual</i>				
0		TN (2672)	FP (105)	2777	0		TN (2610)	FP (167)	2777
1		FN (426)	TP (2312)	2738	1		FN (559)	TP (2179)	2738
__all__		3098	2417	5515	__all__		3169	2346	5515
<i>Predicted</i>		SVM model			<i>Predicted</i>				
<i>Actual</i>					<i>Actual</i>				
0		TN (2645)	FP (132)	2777	0				
1		FN (474)	TP (2264)	2738	1				
__all__		3119	2396	5515	__all__				

AI₃- Confusion Matrix for all models after the fourth fold of the cross-validation process.

<i>Predicted</i>	NB model	0	1	__all__	<i>Predicted</i>	ET model	0	1	__all__
<i>Actual</i>					<i>Actual</i>				
0		TN (2733)	FP (44)	2777	0		TN (2640)	FP (137)	2777
1		FN (1187)	TP (1551)	2738	1		FN (601)	TP (2137)	2738
__all__		3920	1595	5515	__all__		3241	2274	5515
<i>Predicted</i>	KNN model	0	1	__all__	<i>Predicted</i>	LR model	0	1	__all__
<i>Actual</i>					<i>Actual</i>				
0		TN (2559)	FP (218)	2777	0		TN (2632)	FP (145)	2777
1		FN (422)	TP (2534)	2738	1		FN (376)	TP (2362)	2738
__all__		3061	2454	5515	__all__		3008	2507	5515
<i>Predicted</i>	GB model	0	1	__all__	<i>Predicted</i>	RF model	0	1	__all__
<i>Actual</i>					<i>Actual</i>				
0		TN (2647)	FP (130)	2777	0		TN (2610)	FP (167)	2777
1		FN (449)	TP (2289)	2738	1		FN (512)	TP (2226)	2738
__all__		3096	2419	5515	__all__		3122	2393	5515
<i>Predicted</i>	SVM model	0	1	__all__	<i>Predicted</i>				
<i>Actual</i>					<i>Actual</i>				
0		TN (2627)	FP (150)	2777	0				
1		FN (432)	TP (2306)	2738	1				
__all__		3059	2456	5515	__all__				

AI4- Confusion Matrix for all models after the fifth fold of the cross-validation process.

<i>Predicted</i>	NB model	0	1	__all__	<i>Predicted</i>	ET model	0	1	__all__
<i>Actual</i>					<i>Actual</i>				
0		TN (2667)	FP (48)	2715	0		TN (2593)	FP (122)	2715
1		FN (1328)	TP (1472)	2800	1		FN (580)	TP (2220)	2800
__all__		3995	1520	5515	__all__		3173	2342	5515
<i>Predicted</i>	KNN model	0	1	__all__	<i>Predicted</i>	LR model	0	1	__all__
<i>Actual</i>					<i>Actual</i>				
0		TN (2504)	FP (211)	2715	0		TN (2537)	FP (178)	2715
1		FN (429)	TP (2371)	2800	1		FN (361)	TP (2439)	2800
__all__		2933	2582	5515	__all__		2898	2617	5515
<i>Predicted</i>	GB model	0	1	__all__	<i>Predicted</i>	RF model	0	1	__all__
<i>Actual</i>					<i>Actual</i>				
0		TN (2604)	FP (111)	2715	0		TN (2572)	FP (143)	2715
1		FN (411)	TP (2389)	2800	1		FN (528)	TP (2272)	2800
__all__		3015	2500	5515	__all__		3100	2415	5515
<i>Predicted</i>	SVM model	0	1	__all__	<i>Predicted</i>				
<i>Actual</i>					<i>Actual</i>				
0		TN (2557)	FP (158)	2715	0				
1		FN (434)	TP (2366)	2800	1				
__all__		2991	2524	5515	__all__				

ATTACHMENT II

AI1- Confusion Matrix for all models after the second fold of the cross-validation process.

Confusion Matrix: NB model

Predicted	1.0	2.0	3.0	4.0	5.0	__all__
Actual						
1.0	320	7	0	0	480	807
2.0	1	471	3	1	223	699
3.0	10	9	425	1	299	744
4.0	0	0	0	29	22	51
5.0	0	0	1	0	41	42
__all__	331	487	429	31	1065	2343

Confusion Matrix: ExtraTree model

Predicted	1.0	2.0	3.0	4.0	5.0	__all__
Actual						
1.0	737	38	32	0	0	807
2.0	65	623	11	0	0	699
3.0	166	78	499	1	0	744
4.0	10	16	6	19	0	51
5.0	22	14	3	0	3	42
__all__	1000	769	551	20	3	2343

Confusion Matrix: KNN model

Predicted	1.0	2.0	3.0	4.0	5.0	__all__
Actual						
1.0	708	43	54	1	1	807
2.0	30	645	23	1	0	699
3.0	98	49	591	2	4	744
4.0	4	5	9	33	0	51
5.0	8	10	6	0	18	42
__all__	848	752	683	37	23	2343

Confusion Matrix: Logistic Regression model

Predicted	1.0	2.0	3.0	4.0	5.0	__all__
Actual						
1.0	729	31	45	1	1	807
2.0	47	630	21	1	0	699
3.0	132	39	567	1	5	744
4.0	5	8	8	30	0	51
5.0	16	5	4	1	16	42
__all__	929	713	645	34	22	2343

Confusion Matrix: GradientBoosting model

Predicted	1.0	2.0	3.0	4.0	5.0	__all__
Actual						
1.0	701	43	62	0	1	807
2.0	53	616	30	0	0	699
3.0	143	56	539	1	5	744
4.0	6	4	12	29	0	51
5.0	16	7	4	0	15	42
__all__	919	726	647	30	21	2343

```

-----
Confusion Matrix: RandomForest model
Predicted   1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         711  44  52   0   0    807
2.0         85 592  22   0   0    699
3.0        173  66 505   0   0    744
4.0         11  15  15  10   0    51
5.0         24  11   6   0   1    42
__all__    1004 728 600  10   1   2343
-----

```

```

-----
Confusion Matrix: SVM model
Predicted   1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         761  43   3   0   0    807
2.0         55 639   4   1   0    699
3.0        232  60 446   1   5    744
4.0         12   8   2  29   0    51
5.0         17   7   4   0  14    42
__all__    1077 757 459  31  19   2343
-----

```

III₂- Confusion Matrix for all models after the third fold of the cross-validation process.

```

-----
Confusion Matrix: NB model
Predicted   1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         325   6   3   0  501    835
2.0          8 484   9   0  233    734
3.0          6   6 402   1  267    682
4.0          0   0   0  58   8    66
5.0          1   0   1   0  24    26
__all__     340 496 415  59 1033   2343
-----

```

```

-----
Confusion Matrix: ExtraTree model
Predicted   1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         737  38  60   0   0    835
2.0          56 655  23   0   0    734
3.0         130  59 493   0   0    682
4.0          13  13   7  33   0    66
5.0           5  11   5   0   5    26
__all__     941 776 588  33   5   2343
-----

```

```

-----
Confusion Matrix: KNN model
Predicted   1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         711  50  73   1   0    835
2.0          26 676  32   0   0    734
3.0          62  51 563   3   3    682
4.0           3   4   1  58   0    66
5.0           1   6   4   0  15    26
__all__     803 787 673  62  18   2343
-----

```

```

-----
Confusion Matrix: Logistic Regression model
Predicted  1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         744  26  64   0   1   835
2.0         34 677  23   0   0   734
3.0         94  46 539   0   3   682
4.0          2  3  3  58   0   66
5.0          5  4  1  0  16   26
__all__     879 756 630  58  20  2343
-----

```

```

-----
Confusion Matrix: GradientBoosting model
Predicted  1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         716  33  86   0   0   835
2.0         42 655  37   0   0   734
3.0        113  64 502   1   2   682
4.0          6  5  2  53   0   66
5.0          1  5  6  0  14   26
__all__     878 762 633  54  16  2343
-----

```

```

-----
Confusion Matrix: RandomForest model
Predicted  1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         730  37  68   0   0   835
2.0         65 628  41   0   0   734
3.0        143  46 493   0   0   682
4.0         20  14  13  19   0   66
5.0          7  11  5  0  3   26
__all__     965 736 620  19  3  2343
-----

```

```

-----
Confusion Matrix: SVM model
Predicted  1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         795  28  12   0   0   835
2.0         41 677  16   0   0   734
3.0        186  66 428   0   2   682
4.0          6  4  3  53   0   66
5.0          5  5  2  1  13   26
__all__    1033 780 461  54  15  2343
-----

```

AI3- Confusion Matrix for all models after the fourth fold of the cross-validation process.

```

-----
Confusion Matrix: NB model
Predicted  1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         287  2  3  0  502   794
2.0          1 494  4  3  226   728
3.0          8  5 441  0  267   721
4.0          0  0  1 49  17   67
5.0          0  0  3  0  30   33
__all__     296 501 452  52 1042  2343
-----

```



```

Confusion Matrix: ExtraTree model
Predicted  1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         730   21   43    0    0     794
2.0          70  643   15    0    0     728
3.0         141   83  497    0    0     721
4.0          14   13    9   31    0     67
5.0          10    9   10    0    4     33
__all__     965  769  574   31    4    2343

```

```

-----
Confusion Matrix: KNN model
Predicted  1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         688   28   76    2    0     794
2.0          34  660   29    5    0     728
3.0          80   32  602    4    3     721
4.0           6    4    5   52    0     67
5.0           3    9    7    0   14     33
__all__     811  733  719   63   17    2343

```

```

-----
Confusion Matrix: Logistic Regression model
Predicted  1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         715   26   53    0    0     794
2.0          49  652   23    4    0     728
3.0         110   41  570    0    0     721
4.0           9    4    4   49    1     67
5.0           6    4    2    0   21     33
__all__     889  727  652   53   22    2343

```

```

-----
Confusion Matrix: GradientBoosting model
Predicted  1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         700   21   73    0    0     794
2.0          55  633   39    1    0     728
3.0         119   77  525    0    0     721
4.0           5    5    7   49    1     67
5.0           4    5    6    0   18     33
__all__     883  741  650   50   19    2343

```

```

-----
Confusion Matrix: RandomForest model
Predicted  1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         720   22   52    0    0     794
2.0          88  614   26    0    0     728
3.0         153   83  485    0    0     721
4.0          20   19   14   14    0     67
5.0          15    9    9    0    0     33
__all__     996  747  586   14    0    2343

```

Confusion Matrix: SVM model

Predicted	1.0	2.0	3.0	4.0	5.0	__all__
Actual						
1.0	761	26	7	0	0	794
2.0	65	651	8	4	0	728
3.0	199	55	467	0	0	721
4.0	12	4	1	49	1	67
5.0	9	5	7	0	12	33
__all__	1046	741	490	53	13	2343

III4- Confusion Matrix for all models after the fifth fold of the cross-validation process.

Confusion Matrix: NB model

Predicted	1.0	2.0	3.0	4.0	5.0	__all__
Actual						
1.0	327	10	4	0	513	854
2.0	1	499	3	1	215	719
3.0	6	11	408	0	254	679
4.0	0	0	0	47	18	65
5.0	0	0	0	0	26	26
__all__	334	520	415	48	1026	2343

Confusion Matrix: ExtraTree model

Predicted	1.0	2.0	3.0	4.0	5.0	__all__
Actual						
1.0	758	34	62	0	0	854
2.0	61	643	15	0	0	719
3.0	114	82	483	0	0	679
4.0	17	19	3	26	0	65
5.0	9	8	7	0	2	26
__all__	959	786	570	26	2	2343

Confusion Matrix: KNN model

Predicted	1.0	2.0	3.0	4.0	5.0	__all__
Actual						
1.0	738	48	68	0	0	854
2.0	26	671	22	0	0	719
3.0	66	47	563	2	1	679
4.0	4	5	4	52	0	65
5.0	4	3	9	0	10	26
__all__	838	774	666	54	11	2343

Confusion Matrix: Logistic Regression model

Predicted	1.0	2.0	3.0	4.0	5.0	__all__
Actual						
1.0	768	25	59	1	1	854
2.0	41	661	16	1	0	719
3.0	104	47	527	0	1	679
4.0	8	6	5	45	1	65
5.0	6	4	7	0	9	26
__all__	927	743	614	47	12	2343

```

-----
Confusion Matrix: GradientBoosting model
Predicted  1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         723   32   96    0    3    854
2.0          45  637   37    0    0    719
3.0         111   66  502    0    0    679
4.0           6    8    7   44    0    65
5.0           4    5    6    0   11    26
__all__     889  748  648   44   14   2343
-----

```

```

-----
Confusion Matrix: RandomForest model
Predicted  1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         749   36   69    0    0    854
2.0          73  617   29    0    0    719
3.0         132   70  477    0    0    679
4.0          25   19   11   10    0    65
5.0          11    5    9    0    1    26
__all__     990  747  595   10    1   2343
-----

```

```

-----
Confusion Matrix: SVM model
Predicted  1.0  2.0  3.0  4.0  5.0  __all__
Actual
1.0         798   41   15    0    0    854
2.0          50  662    6    0    1    719
3.0         159   67  452    0    1    679
4.0           9    8    3   45    0    65
5.0           9    4    5    0    8    26
__all__    1025  782  481   45   10   2343
-----

```