Universidade do Minho
Escola de Engenharia
Departamento de Informática

Andrea Ferreira Meireles Silva

Identification and classification of transporter proteins using deep learning models

October 2019

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Andrea Ferreira Meireles Silva

Identification and classification of transporter proteins using deep learning models

Master dissertation

Master Degree in Bioinformatics

Dissertation supervised by

Professor Doutor Miguel Rocha

Professor Doutor Óscar Dias

October 2019

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Acknowledgements / Agradecimentos

Apesar deste trabalho ser uma conquista pessoal, tal não seria possível sem algumas pessoas que me apoiaram durante todo o ano e sempre me souberam dar uma palavra de conforto e força nos momentos mais difíceis. Por esse motivo, guardo a primeira secção para agradecer o apoio incondicional de todas estas pessoas, realçando que sem as mesmas certamente não teria conseguido chegar tão longe.

Gostaria de agradecer aos meus orientadores, professor Miguel Rocha e professor Óscar Dias, pelos conselhos e orientação, assim como por todo o tempo dispensado para me ajudar e rever o meu trabalho.

Aos meus colegas de mestrado, especialmente às minhas colegas Cláudia, Inês, Sofia e Bárbara, por me terem acolhido tão bem, pela amizade e por todos os momentos de diversão, união e descontração que me proporcionaram ao longo deste mestrado, que irei guardar para sempre.

Um agradecimento especial aos meus pais, pela constante dedicação e preocupação, pela educação e liberdade que me deram, deixando-me e encorajando-me a seguir os meus sonhos e a dar o meu melhor. São os meus maiores exemplos, e orgulho-me todos os dias dos exemplos que são a seguir e de tudo o que fizeram e fazem por mim.

Um agradecimento igualmente especial à minha irmã, que foi uma constante prova de determinação e força de vontade, que lutou e me fez sempre ver que devemos sempre lutar pelos nossos sonhos e nunca desistir.

Ao resto da minha família, avós e tios, que constantemente me perguntavam e apoiavam a minha vida de estudante, sempre me encorajando e dando forças.

Quero ainda agradecer ao meu namorado e amigos, que sempre me ajudaram nos momentos de maior fraqueza e foram um apoio incondicional, sempre dispostos a ajudar e que sempre me incentivaram com palavras de força e de apoio nos momentos certos.

A todos vocês, obrigada pela força que sempre me deram e pelo exemplo que têm sido, sem vocês eu não estaria aqui.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Resumo

Nos últimos anos a identificação e sequenciação de proteínas transportadoras tem crescido, uma vez que estas são de extrema importância no corpo humano e em todos os seres vivos, sendo responsáveis pela absorção e movimentação de moléculas essenciais às células e ainda pela excreção de produtos do metabolismo celular. A identificação de genes que codificam proteínas transportadoras é muito importante em várias áreas, como farmacocinética e reconstrução de modelos metabólicos em escala genómica que permitem perceber a relação entre genótipos-fenótipos.

De forma a tentar diferenciar proteínas transportadoras de não transportadoras duas abordagens foram realizadas, treinando e testando modelos de machine learning e de deep learning. Os dados utilizados provêm da base de dados TCDB, que contém proteínas transportadoras, e da base de dados Swiss-Prot, onde as proteínas foram filtradas para serem obtidas proteínas não transportadoras, obtendo no final um conjunto de dados equilibrado. De seguida, através desses dados foram obtidas características das proteínas através das suas sequências, sendo assim utilizado para treinar diferentes modelos de machine learning e deep neural networks. Nesta abordagem os modelos apresentaram um bom desempenho global, atingindo 89% de acerto na identificação de proteínas transportadoras. Todos os modelos treinados apresentam um elevado número de falsos negativos em comparação com o número de falsos positivos, indicando que a maior falha nos modelos prende-se na identificação de proteínas transportadoras como não transportadoras.

O principal objetivo deste projeto prendia-se com a utilização de métodos de deep learning para identificar proteínas transportadoras, apenas utilizando as suas sequências de aminoácidos como entrada, comparando assim as duas abordagens realizadas. Desta forma, utilizando apenas as sequencias das proteínas, diferentes redes neuronais foram treinadas e testadas, desde redes neuronais recorrentes a convolucionais, obtendo um desempenho global muito semelhante ao da abordagem anterior, atingindo também um valor de 89% de acerto na identificação de proteínas transportadoras. Assim, foram alcançados modelos de desempenho preditivo semelhante sem a necessidade de calcular características.

Palavras chave: Deep Learning; Machine Learning; Modelos; Proteínas transportadoras.

Abstract

In the last years, the identification and sequencing of transport proteins has grown, once they are extremely important in the human body and in all living beings, being responsible for the absorption and movement of molecules essential to cells and also for the excretion of cellular metabolism products. Identification of genes that encode transport proteins is very important in areas, such as pharmacokinetics and genome-scale metabolic models reconstruction, which allow us to understand the relationship between genotypes and phenotypes.

In order to try to differentiate transport proteins from non-transport ones, two approaches were taken, training and testing machine learning and deep learning models. The data used came from the TCDB database, which contains transport proteins, and from the Swiss-Prot database, where the proteins were filtered to obtain non-transport proteins, obtaining at the end a balanced dataset. Next, using this dataset, features were created from the protein sequences and used to train different machine learning models and deep neural networks. In this approach the models presented a good overall performance, reaching 89% accuracy in the identification of transport proteins. All trained models have a high number of false negatives compared to the number of false positives, indicating that the major failure in the models is the identification of transport proteins as non-transport proteins.

The main objective of this project was to use deep learning methods to identify transport proteins, only using their aminoacid sequences as inputs, thus comparing the two approaches. Thus, using only the protein sequences, different neural networks were trained and tested, from recurrent to convolutional neural networks, obtaining an overall performance very similar to that of the previous approach, reaching once more 89% accuracy in the identification of transport proteins. Thus, we have attained models of similar predictive performance without the need to compute features.

Keywords: Deep learning; Machine Learning; Models; Transport proteins.

Index

List of Figures

List of Tables

List of Abbreviations and Acronyms

ADE	Absorption, Distribution and Excretion
ANN	Artificial Neural Networks
AUC	Area Under the ROC curve
BBB	Blood-Brain Barrier
BLAST	Basic Local Alignment Search Tool
CNF	Conditional Neural Fields
CNN	Convolutional Neural Networks
CNS	Central Nervous System
CRF	Conditional Random Field
CSV	Comma Separated Values
CV	Cross-Validation
DBN	Deep Belief Network
DL	Deep Learning
DNA	Deoxyribonucleic acid
DNN	Deep Neural Networks
EBI	European Bioinformatics Institute
EC	Enzyme Commission
FN	False Negative
FP	False Positive
GRU	Gated Recurrent Unit
GSMM	Genome-Scale Metabolic Models
HMM	Hidden Markov models
IUBMB	International Union of Biochemistry and Molecular Biology
LOO	Leave-One-Out
LR	Logistic Regression
LSTM	Long Short Term Memory
MAD	Mean of Absolute Deviation

Merlin	Metabolic Models Reconstruction Using Genome-Scale Information
ML	Machine Learning
MRC	Medical Research Council
PCA	Principal Component Analysis
PIR	Protein Information Resource
Pr	Precision
PSSM	Position Specific Scoring Matrix
PTM	Post-Translation Modifications
RBM	Restricted Boltzmann Machine
Re	Recall
ReLU	Rectified Linear Unit
RF	Random Forests
RMSE	Square Root of the mean of Squared Errors
RNA	Ribonucleic acid
RNN	Recurrent Neural Networks
ROC	Receiver Operating Characteristics
SIB	Swiss Institute of Bioinformatics
SGD	Stochastic Gradient Descent
SSE	Sum of Squared Errors
SVM	Support Vector Machines
TANH	Hyperbolic Tangent
TC	Transporter Classification
TCDB	Transporter Classification Database
TN	True Negative
TP	True Positive
TrSSP	Transporter Substrate Specificity Prediction Server

UniProt Universal Protein Resource

1. Introduction

1.1. Context and motivation

Transport proteins are involved in the transportation of amino acids, sugars, proteins, cations, anions, mRNAs, water, hormones, electrons and substrates, playing an important role in the human body, especially in cellular functions like cell metabolism, ion homeostasis, signal and energy transduction, osmoregulation and several other process [31]. Transport proteins facilitate the absorption, uptake and efflux of many drugs, and are also responsible for the oxygen distribution, being vital to the growth and life of all living things. Due to transporters importance, their sequencing and analyses are very important on structural and functional biology, but also on fields like drug metabolism and pharmacokinetics, being also very relevant to genome-scale metabolic models reconstruction [12].

Nowadays, sequence analysis is a very common task used to predict proteins' characteristics, mainly their functional annotation [3]. In this field, the molecular characterization of transporter proteins has seen a relevant growth. These are extremely important since they have a crucial role in the absorption, distribution and excretion of therapeutic drugs within the human body [4]. The accurate and automatic identification of genes encoding transport proteins and the carried metabolites are also essential for the development of accurate Genome-Scale Metabolic Models (GSMM's) [5].

GSMM's are mathematical representations of living organisms that link genomic information to metabolic reaction networks allowing to understand genotype-phenotype relationships [1]. Thus, GSMM's are used to predict, *in silico*, microorganisms' responses to different genetic or environmental stressors, and, when containing transport proteins, they can associate genes to reactions [2].

On the other hand, Deep Learning is beginning to impact biological research and biomedical applications, as a result of its ability to integrate vast datasets, learn arbitrarily complex relationships and incorporate existing knowledge. Deep learning models are already used to predict, with varying degrees of success, how genetic variation alters cellular processes involved in pathogenesis, which small molecules will modulate the activity of therapeutically relevant proteins, and whether radiographic images are indicative of disease, among many other biomedical applications [6].

Several machine learning (ML) and deep learning (DL) supervised models, widely used to analyse biological sequence data, based on common features extracted from the sequences, can be used to identify and characterize transport proteins [8]. In the neural networks used for deep learning, inputs are fed into an input layer and this layer feeds into one or more hidden layers. In the end, the model eventually

produces an output layer [9]. The performance of these models can be improved by introducing convolutional filters [3]. Also, recurrent neural networks, such as the Long Short Term Memory (LSTM) model were previously designed to handle sequences and were already used to predict, with high accuracy, the subcellular location of proteins using only the protein sequence information [3].

This dissertation emerges to address the development of accurate computational prediction methods, which will be able to predict if a protein is a transporter, from its amino acid sequence. One of the clinical applications of these methods is the prediction of potential inhibitors of multi-drug resistance transporters, as well as transporters that grant survival advantages to pathogenic microbes, which would help in the design of novel anti-microbial drugs [7].

1.2. Objectives

The main aim of this work is to develop methods based in the use of deep learning models to address the identification of transporter proteins solely from their amino acid sequences, comparing their results with previous approaches for this task. This encompasses a number of scientific/technological objectives:

- Review the state of the art Machine Learning and Deep Learning models, as well as sequence analysis algorithms and tools, and their applications in transport protein identification and related tasks.
- Develop machine learning approaches that use features extracted from protein sequences as input, testing various models to predict whether a protein is a transporter or not.
- Develop a deep learning (DL) approach with deep neural networks that uses the dataset containing features, thus trying to predict whether a protein is a transporter and comparing the results with the previous approach.
- Develop several recurrent neural networks using different hyperparameters and types of layers, which should be able to predict if a protein is a transporter, from its amino acid sequence, comparing the results with other methods.
- Validating these models with a large-scale dataset encompassing positive and negative cases, extracted from available public databases.

1.3. Document Organization

This document is organized in 7 chapters. The first one contemplates a brief introduction to the dissertation, where the principal areas are broached, and its motivation and objectives are announced.

Chapter 2 addresses a review of the state of the art with a more biological approach, where the concept of transport proteins and their importance are presented, and related to sequence analysis methods. Here, metabolic models are also explained and some general protein and transporter databases are introduced.

Chapter 3 can be divided into three subsections. It starts with an introduction to some machine learning concepts and models, where neural networks have a special attention. Similarly, the next subsection is where deep learning concepts and models are presented, along with some bioinformatics frameworks and tools. Finally, at the end of this chapter, a table is presented that summarizes several papers in the sequence analysis field using machine learning and deep learning techniques.

Chapter 4 presents the methods used throughout the thesis, introducing how the data were selected and treated, as well as the models and processes used in all approaches. This chapter also gives a brief explanation of the software developed and executed, and how the methods were applied, thus demonstrating all the development.

Finally, at the Chapter 5, the results are presented and discussed for all the approaches taken. And, lastly, the conclusions of this dissertation are presented as well as the future work.

2. Transport Proteins

2.1. Transport Proteins

Transport proteins, a major class of solute-translocation system, are usually located at membranes and have a very important role in the human body, and all living species. Thus, besides playing an important role in membrane stability and architecture, transport proteins are also important for the regulation of metabolic functions, being responsible for roles in transport, absorption, and signal and energy transduction [10].

Regarding energy-coupling mechanisms, there are two classes of membrane transporters, abundant in all known species of eukarya, archaea and bacteria. The primary transport systems convert light or chemical energy into electrochemical energy, for instance solute concentration gradients across membranes. Secondary transport systems use the free energy difference stored in the electrochemical gradients of protons, for instance, to drive translocation reactions [10].

The cell membrane is highly impermeable to foreign substances, thus, the search for new transport proteins that allow the intercellular delivery of drugs, genes or proteins therapeutics has been an active and growing research area [11]. The molecular and functional characterization of transport proteins in animals and man is increasing significantly. These play an important role in attenuating the absorption, distribution and excretion (ADE) of drugs that have been shown to be substrates or inhibitors [12]. Therefore, carrier-mediated processes govern drug disposition and response, influencing the efficacy of drug therapy, playing critical roles in the overall disposition of drugs in clinical use, such as xenobiotic transport [12]. A key determinant in the extent of drug ADE is the lipophilicity of the substance, its affinity and ability to be dissolved in lipids. Due to the contribution of transport proteins, an increased lipophilicity is not always accompanied of an increased permeability [12].

There are two major classes of drug transporters: uptake or influx transporters, that facilitate the translocation of drugs into cells, and efflux transporters that export drugs from the intracellular to the extracellular milieu, usually against high concentration gradients. However, there are transport proteins that can exhibit both properties, influx and efflux [12].

Transport proteins have several functions in different parts of the human body (Figure 1), specially controlling the permeability across tissue membranes, such as the distribution of drugs in the body. For instance, the active biliary excretion is governed by transport proteins, mainly ATP-dependent transporters, such as P-glycoproteins and MRP2. Across the gastrointestinal tract, P-glycoprotein has a

clear role in absorption, limiting permeability. Similarly, at the blood-brain barrier (BBB), although increasing lipophilicity leads to increased BBB permeability, mediated efflux by P-glycoprotein limits brain penetration of a range of drugs, specially poorly lipophilic compounds, having possible consequences in therapeutic effectiveness of Central Nervous System (CNS) agents. Uptake transporters present in the sinusoidal membrane have also an important role in the liver, being a rate-limiting step in hepatic clearance and extraction of many drugs from the portal blood into hepatocytes. The kidney has an important role in the elimination of many drugs. Thus, carrier-mediated processes are involved in active renal secretion of drugs, mainly hydrophilic cations and hydrophilic anions [12].

Although transport proteins have already been considered as having a very important role in drug ADE, there are some factors that need to be considered with respect to variability in drug disposition and response, such as the fact that they are saturable, inducible, can be inhibited and display some degree of polymorphism [12].

In short, the extent of drug absorption and access to target tissue compartments depends on physicochemical properties of a drug, such as lipophilicity, pKa, size, as well as transport proteins affinity. Those proteins can be involved not only in the active absorptive efflux of drugs and xenobiotics, but also in the active absorptive influx of compounds, endogenous substrates, such as sugars, amino acids, oligopeptides, monosaccharides, bile acids, several water-soluble vitamins and hormones, that are extremely important and critical for maintenance of normal homeostasis [12] [12].

The contribution of transport proteins on drug ADE needs to be carefully considered in drug discovery and development, requiring appropriate *in vivo* models to evaluate the role of an individual transport protein in the disposition and elimination of drugs [12]. For instance, analyzing if transport proteins are implicated in a particular therapeutic target area, or whether these modulate the disposition and safety of co-medication [13]. P-glycoprotein in BBB already proved to have a decisive effect on the clinical usage of certain drugs. These models are also important to explore inhibitors of drug transporters, through mutant strains and specific genetic knockout models. As stated above, P-glycoprotein plays a key role on limiting the brain penetration. Thus, researchers suggested the possibility of reducing AIDS-induced dementia complex by using selective and potent P-glycoprotein inhibitors, to improve brain penetration of HIV protease inhibitors [12].

The degree of expression and functionality of transport proteins can be influenced by substrate affinity, genetic polymorphism or even drug-drug interactions, affecting the therapeutic effectiveness, safety and target specificity of substrates [12].

Because of the increasing preclinical and clinical evidence that supports a key determining factor of transport proteins in drug ADE, drug transporter science is now a rapidly emerging area in drug metabolism and pharmacokinetics, becoming a field that requires the identification of new transport proteins, as well as specific probe substrates and inhibitors for novel transporters [12]. This emerging field may lead to drug therapies that will further maximize target specificity, while minimizing unintended toxicities [12].

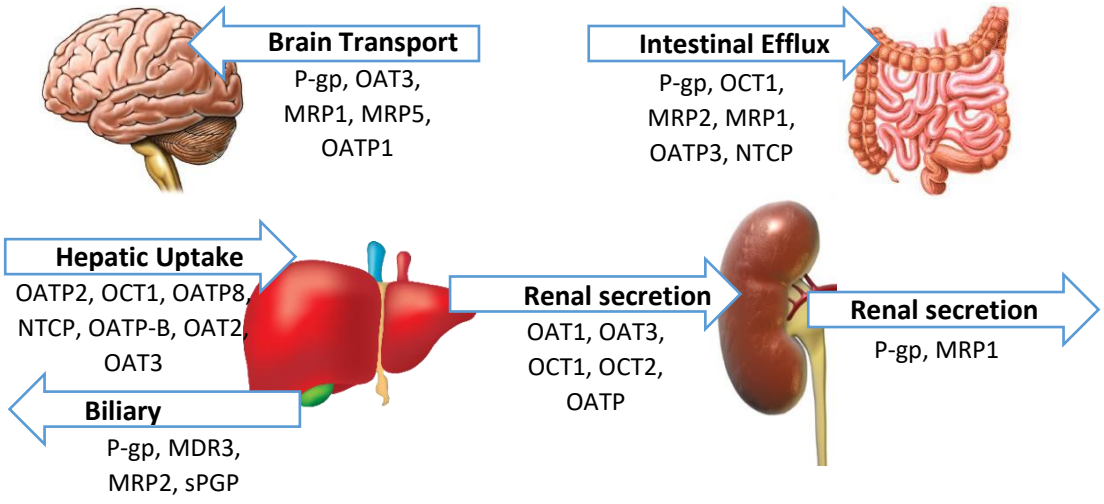

Figure 1. Transport proteins and their role in drug disposition. Adapted from [12].

2.1.1.Transport Classification system

The Transporter Classification (TC), a functional/phylogenetic system, was designed to classify transmembrane transport proteins [14]. In 2002, this nomenclature was adopted by the Transport Nomenclature Panel of the International Union of Biochemistry and Molecular Biology (IUBMB) as the internationally acclaimed system for transport proteins classification [14] [15]. The transporter classification system consists of a set of representative protein sequences, where the transporters are classified with a five-character designation indicating the transporter class (a single digit), subclass (a letter), family (a number), subfamily (a number), the protein (a number), in this specific order. This functional/phylogenetic system reflects the mechanism, mode of energy coupling, polarity and substrate specificity of a transporter, providing much information [16].

The TC system is divided in a total of 9 classes of transporter families and each of these classes are subdivided into subclasses, as shown in Table 1 [15]. The first 5 classes are well-defined classes.

Thus, when enough information is available a TC number can be assigned to a transporter [16]. The available classes are:

- Class 1 consists of channel-type facilitators;
- Class 2 includes electrochemical potential-driven transporters, also called secondary carrier-type facilitators;
- Class 3 consists on primary active transporters that use a primary source of energy to drive active transport of solutes against concentration gradients;
- Class 4 consists on the group of translocators, including transport systems that chemically alter the substrate during transport across a membrane;
- Class 5 is for transmembrane electron carriers, includes systems that catalyze electron flow across a biological membrane [14];
- Classes 6 and 7 are currently unused and are intended for new classes of transporters;
- Class 8 is reserved for accessory transport proteins, that is, proteins that function together with transport proteins;
- Finally, Class 9 is for incompletely characterized transport systems, transporters whose information available is not enough to characterize and classify the transporter [16].

Phylogenetic analyses may reveal structure/function relationships. Thus, computational approaches are useful to track phylogenetic relationships and the pathways by which proteins have evolved. In short, the TC system enables an extensive body of knowledge for transport systems, having an important role for any researcher [14].

Table 1. Hierarchical system for classifying families of transporters of defined function in the TC system.

Class	Subclass
1. Channels / Pores	1.A. α-Helical protein channels
	1.B. β-Barrel protein porins
	1.C. Toxin channels
	1.D. Non-ribosomally synthesized channels
	1.E. Holins

2. Electrochemical Potential-driven Transporters	2.A. Protein porters
	2.B. Non-ribosomally synthesized porters
	2.C. Ion gradient-driven energizers
3. Primary Active Transporters	3.A. P-P-bond hydrolysis-driven systems
	3.B. Decarboxylation-driven systems
	3.C. Methyltransfer-driven systems
	3.D. Oxidoreduction-driven systems
	3.E. Light absorption-driven systems
4. Group Translocators	4.A. Phosphotransfer-driven systems
5. Transmembrane Electron Carriers	5.A. Two-electron transfer carriers
	5.B. One-electron transfer carriers
8. Accessory Factors Involved in Transport	8.A. Auxiliary transport proteins
9. Incompletely Characterized Transport Systems	9.A. Transporters of unknown classification
	9.B. Putative uncharacterized transporters
	9.C. Functionally characterized transporters with unidentified sequences

2.2. Sequence Analysis

Protein sequence analysis is central to modern biological research and the need to understand the data is increasing. Sequence data can be useful to establish evolutionary relationships between proteins and to correlate gene structure, map active sites and domain boundaries [17] [18]. Sequence analysis allows to perform several tasks, such as to align two sequences to discover if these are related, phylogenetic comparison, motif detection, subcellular localization, domain detection, etc [18] [19]. Sequence analysis can also lead to identify mutations and translocations and can be helpful on cloning.

Most of the sequence analysis methods are based on probabilistic modelling, for instance on sequence alignments, where score matrices are used to determine its significance, or on phylogenetic trees, in which maximum likelihood approaches can be used [18].

One of the most common tasks of sequence analysis is homology searching, that is, sequence similarity that allows identifying homologous sequences in databases providing, if the match between the sequences is statistically significant, important information about new sequences [20]. There are several

tools that allow running sequence similarity searches. One of the most used is the Basic Local Alignment Search Tool (BLAST) which provides several programs, such as nucleotide blast, protein blast, blastx, tblastn and tblastx, being a tool that can be used with both DNA/RNA and proteins [21].

Protein sequence analysis allows predicting important information about the protein, such as its localization, function and structure. There are, however, problems in the analysis of proteins associated to its post-translation modifications (PTM), which may hinder the analysis of the proteome in comparison to the genome. After the translation of the mRNA, proteins usually pass through modifications that can lead to changes in physical or chemical properties, or even to changes of localization, activity or stability [22].

Several bioinformatics tools were developed to facilitate sequence analysis. One example is the Staden Package, developed at the Medical Research Council (MRC) Laboratory of Molecular Biology, a sequence analysis open-source software that performs most tasks of sequence analysis, such as gene finding, pattern searching, comparison, secondary structure prediction and other tasks [23]. Another example is the HMMER tool, that uses profile hidden Markov models (HMM) for detecting sequence similarity of proteins or nucleotides and allows sequence alignment [24].

There are also specific tools for protein characterization. For instance, there are several tools available for protein localization using different models, algorithms and datasets, to predict the subcellular localization, such as PSORTb [25], TargetP [26], SubLoc [27]. Tools like Phobius [28], BOMP [29], TMHMM [30] allow assessing the topology of membrane proteins, thus helping to identify transport proteins. The Transporter Substrate Specificity Prediction Server (TrSSP [31]) implements SVM models (see section 3.1.2.4) and allows predicting membrane transport proteins and their substrate category [32].

In short, sequence analysis is extremely powerful and plays an important role in genomics era, and, consequently, hypotheses derived by computational methods will be a successful step in the design of experiments [33].

2.3. Databases

With the rapid accumulation of genome sequences and, consequently, the increasing volume and variety of protein sequences and functional information, it is important to have available resources providing cornerstones for scientists. Areas like biological and biotechnological research, especially in the

field of proteomics, are information-dependent and databases of biological knowledge play an important role [34].

There are different types of databases, depending on its content and its biological interest, for instance, there are databases for nucleotides such as Genbank [35], DDBJ [36], and protein-based databases, such as SWISS-PROT [37], International Protein Index (IPI) [38] and the NCBI non-redundant database [39].

One of the most used databases with protein sequences and functional information is the Universal Protein Resource (UniProt) [34]. This database was formed by uniting 3 different protein database activities which previously coexisted, the Swiss-Prot and TrEMBL [40] groups at the Swiss Institute of Bioinformatics (SIB) and European Bioinformatics Institute (EBI), and the Protein Information Resource (PIR) [41] group at Georgetown University Medical Center an National Biomedical Research Foundation. Thus, UniProt is a non-redundant database that provides protein sequences with annotations and functional information, being a single, centralized, authoritative resource [42]. The main aim of this consortium is to freely support biological and biotechnological research facilitating knowledge discovery by maintaining a high-quality, comprehensive, fully classified, richly and accurately annotated protein sequence knowledgebase [44].

The power of computational methods in transporter analysis and prediction is growing exponentially due to transporter proteins critical role in ADE of drugs, and in life science industries. Therefore, the large amount available data regarding transport proteins, and the need to facilitate its analysis, led to databases specialized on transport proteins such as the Transporter Classification Database (TCDB) [44] and TransportDB [43].

TCDB is a freely accessible web resource, containing curated data of published information that allows to analyze the unique characteristics of transport proteins, serving as a genome transporter annotation resource. This database offers, for each transport protein, information, such as:

- the transporter classification (TC) number;
- protein sequence;
- structural, functional and evolutionary information;
- 3D macromolecular structure;
- and, protein domains.

These data provides access to hierarchical classification and several other resources for analyzing transmembrane proteins. In short, TCDB is originated from multiple sources such as Swiss-Prot and PubMed literature and is organized into transport families based on the TC system, being a centralized resource for transport protein data and their analysis [44].

TransportDB is a relational database that allows the comprehensive and comparative study of cytoplasmic membrane transport systems and outer membrane channels in different organisms whose complete genome sequences are available. This is a user-friendly web-interface with easy access, which provides tools like BLAST search and comparison between transporter and outer membrane channel contents, 3D structures and phylogenetic trees to these systems and channels. However, detailed information such as, as functional annotation, protein/DNA sequences, supporting bioinformatics evidences, publications and cross-referenced resource links are available for individual proteins. TransportDB, besides presenting the comprehensive transporter profiles, provides tools to view, search, compare and download the transporter data [44].

2.4. Metabolic Models

Following the developments in genomics and since the first genome-scale reconstruction was published, there has been a huge focus on the behavior of complete biological systems and, consequently, the availability and usefulness of genome-scale metabolic models reconstructions have exploded [45] [46].

Genome-scale metabolic models (GSMM's) are developed through the integration of biochemical metabolic pathways information with genomic and genetic data [49]. The process of developing these models is very complex, and the complete reconstruction of a GSMM can take up to one year. This process, presented on Figure 2, follows a protocol that can be summarized in six stages. The first consists on collecting data such as the Enzyme Commission (EC) and TC numbers, associated genes and gene products from different data sources. In the first step of the reconstruction process, the genome annotation or a reannotation, if the information available in public databases is not complete, is performed. The identification of the metabolic reactions associated with the organism, containing reactions catalyzed by the EC numbers previously identified and spontaneous reactions associated to the organism is performed in the second stage. Then, information concerning the stoichiometry of the organism is verified in online databases. The reactions localization, equation representing biomass formation, growth-associated energy requirements and other constraints, such as the reversibility of the

reactions and values of uptake boundary, are also required to complete the reconstruction of the GSMM's are added afterwards. The next stage consists on assembling a reaction set, followed by the use of experimental data to finally validate the GSMM. The final step is an iterative loop, allowing to improve the GSMM [2].

The GSMM of an organism allows performing *in silico* simulations and, predict responses to different genetic or environmental stressors, providing a cheaper and faster alternative to wet lab experiments. Therefore, it is possible to gain better understanding of the observable phenotypes and coordinated functions of the cell from genome sequence and biochemical information of an organism [49] [2]. These models are used with the purpose of contextualizing high-throughput data, guide metabolic engineering and hypothesis-driven discovery, determine multi-species relationships and network property discovery [46].

GSMM's allows predicting high-throughput experimental growth and gene deletion phenotypes. These models have to be robust and reliable, thus it is mandatory to annotate transport proteins, providing associations between genes and reactions. However, transport reactions are only included in models when evidences supporting the inclusion are available in experimental data or literature, though without gene-protein-reaction rules [49] [2].

Several tools that allow reconstructing GSMM's are now available. For instance, the Metabolic Models Reconstruction Using Genome-Scale Information (*merlin*), a user-friendly Java application that can perform the reconstruction process for every organism whose genome is sequenced [48], and has been developed in the host group. *Merlin* includes tools for the identification and annotation of genes encoding transport proteins, can generate transport reactions for transport proteins and it also has tools for the compartmentalization of the model, being capable to predict the localization of the proteins encoded in the genome, thus allowing to localize the metabolites involved in the reactions induced by these proteins [47].

A well-curated metabolic reconstruction is very useful for biological discovery and engineering applications, by identifying specific areas where knowledge is lacking and provide a framework for the integration of high-throughput data [49]. Thus, it is important to identify transport proteins so transport reactions can be included in GSMM's, providing association between genes and reactions and allowing the reconstruction of models more robust and reliable [2].

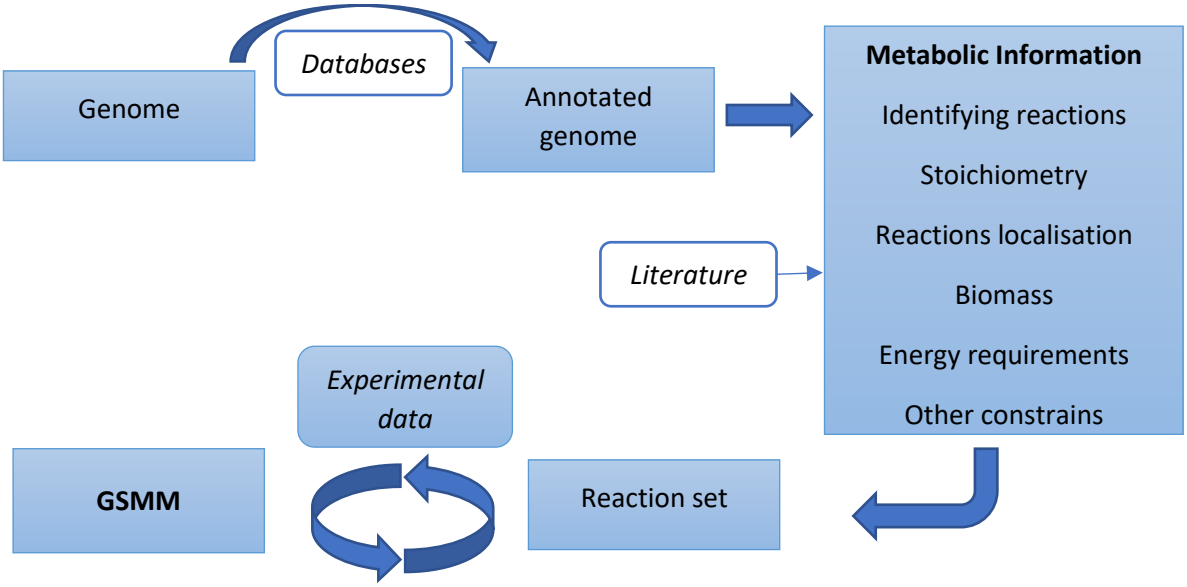

Figure 2. Genome-scale Metabolic Model reconstruction process. Adapted from [48].

3. Machine Learning and Deep Learning

3.1. Machine Learning (ML) concepts

Currently, there is a continuous generation of new data, of large dimensions, hence the designation of big data, and, consequently, the search of new methods / theories to process these data turning them into knowledge is also growing [49]. According to Kevin Murphy, machine learning can be defined "as a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty (such as planning how to collect more data!)" [50]. Machine Learning is part of artificial intelligence, computer science and statistics, and has been growing fast being applied on many biological problems, essentially on six domains: genomics, proteomics, transcriptomics, systems biology, evolution and text mining [51].

There are two main types of machine learning: supervised learning and unsupervised learning. In supervised learning, also called predictive learning, the dataset used contains labelled instances. That is, given a labelled set of input-output pairs (the training set), supervised learning intends to learn a mapping from inputs, x, to outputs, y. Here, the dataset contains the inputs, which represent the features, attributes, each usually as a one-dimensional vector of numbers. The output, or response variable, is usually a categorical or numerical variable, representing, for instance, the class of each item. According to the type of output, the problem can be a classification or regression problem, if it is a categorical or numerical variable, respectively. Classification problems intend to map the input values to a set of discrete classes (the output). Usually, given a labelled training set, a function approximation is estimated in order to predict the output from novel inputs. Regression problems are similar. The major difference to classification problems is that the response variable is continuous. Problems like predicting temperatures or ages are examples of real-world regression problems [50].

At unsupervised learning, also called descriptive learning, the instances are unlabelled. In this type of machine learning, the goal is to find interesting patterns in the data, only having the inputs [50]. These types of algorithms are usually used to discover unknown but useful classes of items [52].

The process of developing a machine learning algorithm involves several essential steps. The first one consists on collecting data, that is, selecting from everything available, the features that are most informative and can be helpful in the problem resolution. The second step involves data processing, where missing feature values and noise are treated, for example. Then, feature selection or other methods can be applied, to transform the data, leading to a more comprehensive, non-redundant, relevant and reduced

data. Finally, after pre-processing and preparing the data, a training set and testing set are defined. Then, the training step follows, where the algorithm is trained with the training set, extracting knowledge from the data and creating a representative model of the information gained. This model is subsequently used to test the algorithm, with the inputs of the test set, the respective output attributes are predicted, being possible to evaluate the effectiveness of the model. After this evaluation of the model performance, it is possible to improve the model, using a different dataset or different methods. Finally, the last step involves the application of the validated model to new and unlabelled data, predicting unknown outputs [52].

3.1.1.Error Estimation

Error estimation is used to evaluate the quality of the model for a given task and can help to choose the better learning method or model. There are different error metrics that can be used, according to the type of the problem [81].

On classification problems, during the testing are produced counts of the correct and incorrect classifications from each class. These counts are displayed into a confusion matrix, that shows the differences between the true (desired) values and the predicted ones, according to each class. The confusion matrix shows all the information about the classifier's performance, presented on table 2 for a binary classification problem. From the confusion matrix, it is possible to observe the values that were correctly predicted, that is, the desired value and the predicted have the same class: True Positive values (TP), when the true value is positive, or the True Negative (TN), when the true value is negative. The values that were not correctly predicted are also possible to observe through the confusion matrix: False Negative values (FN), when the true value is positive but is predicted as negative, or the False Positive (FP), when the true value is negative but is predicted as positive [81].

Table 2. Consufion Matrix

Confusion Matrix		Predicted Values	
		Negative	Positive
Desired Values	Negative	True Negative (TN)	False Positive (FP)
	Positive	False Negative (FN)	True Positive (TP)

With these values, it is possible to extract more meaningful measures. The accuracy can be calculated dividing all the true values (TP and TN) by the sum of all the values (TN, TP, FN, FP), as seen on Equation 1. For multiclass classification, when there are more than two class labels, the accuracy is calculated as the sum of the diagonal (all correct predictions) divided by the sum of all the values. The precision of the algorithm, also known as positive predicted value, is calculated by the TP divided by the sum of all the positive predictions (TP, FP), as seen in Equation 2. The negative predicted value is the opposite of precision, being calculated by the TN divided by the sum of all the negative predictions (TN, FN), as seen on Equation 3. The specificity, type error I, can be calculated by the division between the TN and the sum of TN and FP, as seen in Equation 4. The recall, also known as sensitivity, can be calculated dividing the TP by the sum of TP and FN, as seen on Equation 5. The F-measure combines the values of precision (Pr) and recall (Re), being calculated by the multiplication of 2 with the multiplication of Pr by Re, divided by the sum of Pr and Re, as seen on Equation 6 [82].

Equation 1

$$Accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

Equation 2

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

Equation 3

$$Negative\ Predicted\ Value = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FN}}$$

Equation 4

$$Specificity = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}$$

Equation 5

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Equation 6

$$F - measure = 2 \times \frac{\mathrm{Pr} \times \mathrm{Re}}{\mathrm{Pr} + \mathrm{Re}}$$

The higher the Recall, Specificity, and accuracy values, the better the model will be, however an increase in the Recall can lead to a decrease of the Specificity [70]. In order to evaluate the model, and the relationship between Recall and Specificity, ROC (Receiver Operating Characteristics) curves can be applied. This curve is useful to establish decision of threshold and to select a suitable operating point. The area under the ROC curve (AUC) is used as a measure of a classifier's performance, since it represents the probability that a randomly chosen positive example is correctly rated. AUC is basically the probability of correct ranking and translates the ability of the model to discriminate between the two classes, so, the closer to 1 is the AUC, better the model can discriminate [81].

For regression problems, the difference between the predicted value and the real value corresponds to the error, and, based on the error of each example, error metrics can be calculated. The sum of squared errors (SSE) is one of these error metrics, calculated by the summation of the squared difference between the real value (y_i) and the predicted value ($\hat{y}_i$), as seen in Equation 7. Another error metric is the square root of the mean of squared errors (RMSE), as seen in Equation 8, and the mean of absolute deviation (MAD), as seen in Equation 9 [83].

The lower these error metrics values are, more precise the model is.

Equation 7

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Equation 8

$$RMSE = \sqrt{\frac{SSE}{N}}$$

Equation 9

$$MAD = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{N}$$

There are several error estimation methods whose purpose is to improve the evaluation, based, for example, on resampling the original data. One technique is the Holdout, where the data is divided once into a learning set (also called training set) and a test set. This partition is based on a predetermined

p that represents the distribution between both sets, for instance, if p=1/4, the learning set will have three-quarters of the data and the test set a quarter [104].

Cross-validation (CV) is another effective method for estimating prediction error, that provides a more accurate evaluation, being one of the most used methods [84]. In K-fold cross-validation, the examples are randomly divided into mutually exclusive k-parts (k-folds) of equal sizes. Thus, k-1 folds are used as a training set for the algorithm, being used to generate the model, and the remaining fold is used as test set, to verify the performance of the model. In this way, the algorithm is tested k times, always considering a different fold for test, where the final error corresponds to the mean of the errors calculated for each of the folds [84].

Leave-one-out (LOO) is another cross-validation method, where, given a sample of size N, the training sample will correspond to N-1 examples and the sample left out will be the test set. Thus, we have N different training and test sets, and the process is repeated N times. This method can be equivalent to K-fold cross-validation, if the parameter k is set to N. The final value of the error corresponds to the sum of the errors of each test divided by N [89].

3.1.2.Overfitting

One major problem associated to machine learning algorithms is overfitting. A machine learning algorithm is trained with training data to create a model that is capable of making predictions on new data. However, sometimes, the model is too restricted to the training data and represents them too well, memorizing their noise and characteristics, not being a good model to new data predictions. This is called overfitting, when a model is good to represent the training data and cannot find a general predictive model, not being able to make correct predictions on new data [53].

There are several ways to prevent overfitting, such as the use of regularization methods, dropout and early stopping, which will all be explained further [54]. Very large training sets and pre-training can also reduce overfitting. Reducing the complexity of models, for instance by applying feature selection, prevents overfitting. Methods like Variance Threshold select the features responsible for the variability of the data, excluding those that do present low variance, not given much data information, avoiding overfitting [105].

3.1.3.Main machine learning models

Linear and logistic regression

A linear regression method intends to represent a relation between an independent variable and a set of dependent variables, allowing to predict the outcome variable for new values of the predictor variables [56]. Linear Regression is a method that tries to model this relationship with a linear equation that fits the data, where the outcome variable is assumed to be continuous [57].

Logistic Regression (LR) is a type of multivariable analysis with the ability to model binary or dichotomous outcomes. With a logistic function, that represents the best fitting model describing the relationship between a set of independent variables and a dependent variable, it is possible to estimate the probabilities of a given class. The probability function on LR is a sigmoid function that uses a set of weights (model parameters) and the input feature vector to calculate the probability of the output be equal to the input, equal to 1 (Equation 10). The error function is minimized using the gradient descent method, explained on section 3.2, Equation 11, allowing to estimate the parameters, θ.

Equation 10

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Equation 21

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + \left(1 - y^{(i)}\right) log(1 - h_\theta(x^{(i)}))]$$

This method is used with increasing frequency in the health sciences, since it is appropriate for models involving disease state and decision making. When applied to more than two classes, it is necessary to train a model for each class [56].

Regularization Techniques

Regularization techniques allow to reduce overfitting without being necessary to change the model size or the training data. One of the most commonly used techniques is the L2 regularization, also known as weight decay, where an extra term, called regularization term, is added to the cost function. This term

is basically the sum of the squares of all the weights in the network multiplied by a parameter. This technique penalizes large weights and tend to make the model prefer small weights [55].

Another regularization technique is the L1 regularization, where the unregularized cost function is modified by adding the sum of the absolute values of the weights, being similar to weight decay, penalizing large weights, however with a different behaviour [55].

Support Vector Machines (SVMs)

Support vector machines (SVMs) are ML models introduced by Vapnik, suitable for working with high dimensional data, being capable of classifying samples and helping in the identification of those which have been wrongly classified [62]. Based on a set of labelled training data, support vectors are created and then they are used to calculate a hyperplane that separates the training data, a linear decision function with maximal margin between the vectors of the two classes. SVMs are capable to generate input-output mapping functions like a classification or a regression function. A classification function consists on categorizing the input data, and, when is not possible to divide the data into the two classes by a linear hyperplane, are often used nonlinear kernel functions (like Gaussian and spline kernels) that automatically realize a non-linear mapping to a new feature space. The technique of 'kernels' allows the input data to become more separable compared to the original input space, by transforming it to a high-dimensional feature space [62] [63].

SVMs have mathematical basis, in statistical learning theory, and are also based on the Structural Risk Minimization principle that intends to find a hypothesis for which it is possible to guarantee the lowest true error [63] [64].

Problems like pattern recognition and function estimation have already been solved applying SVMs [65]. Several other real-world learning tasks used SVMs, such as handwritten recognition, object recognition, and text classification [66]. Similarly, areas of biological analysis like microarray expression data, detection of remote protein homologies, recognition of translation initiation sites showed good results when SVM was applied [62].

Decision Trees

A decision tree is based on a tree structure, containing nodes, where each node corresponds to an input attribute to be tested, and leafs, where each leaf corresponds to an output attribute's value. Each

node contains a test that best splits the space of data to be classified, leading to other nodes, depending on the attribute presented, until finally a leaf is reached, where the prediction is obtained [60].

Ensemble Methods

Ensemble methods are learning algorithms that train multiple learners to solve the same problem, combining a set of learners to improve the performance of the overall system. One of the largest areas of supervised learning research is precisely the search for new methods for the construction of good ensembles [78].

There are several methods for constructing ensembles, some general techniques used are [78] [79]:

- Resampling methods: this method consists in manipulating the training examples to generate different hypotheses. Different subsets of the training examples are generated and the algorithm is run several times, always with a different subset. There are several ways of manipulating the training set, the most common are Bagging, Boosting and cross-validation.
- Feature selection methods: the input features available to the learning algorithm are manipulated, reducing the number of input features of the base learners, in order to generate multiple classifiers. This method only improves the accuracy when the input features are highly redundant.
- Randomized ensemble methods: this method consists on inject randomness into the learning algorithm. For example, in the backpropagation algorithm the initial weights of the network are set randomly.

Boosting is an ensemble learning method very used on classification trees, where successive trees are taken into account on points incorrectly predicted by earlier predictors, influencing the final prediction. This method is about producing a very accurate prediction rule by combining rough and moderately inaccurate rules [80]. In boosting methods, at each iteration of the algorithm, a different distribution or weighting over the training examples is used and the training set used is chosen based on the previous performance. In this technique, the examples most often misclassified by the previous base learner are placed with the highest weight and, then, a weighted majority vote is taken into account to combine the base rules [78].

Another ensemble learning method, also used on classification trees, is bagging. In bagging, the ensemble is formed by making bootstrap replicates of the training sets, using them to generate multiple

hypotheses and finally to get an aggregated predictor. Here, the samples are drawn randomly and with replacement. Unlike the previous method, bagging is effective with noisy data and successive trees do not depend on earlier trees and each one is constructed based on a bootstrap sample of the data [78] [80].

K- Nearest Neighbours

K-Nearest Neighbours, KNNs, is an instance based learning method that, instead of generating a model function, it stores the training data using it when a prediction is made. KNN is based on the fact that instances with similar properties are close, so a label can be determined by observing the class of its nearest neighbours. Thus, for classification problems, a label is predicted by locating its k-nearest neighbours and by obtaining the most frequent class label on those k training examples. For regression problems, instead of choosing the most frequent class of the k-nearest neighbours, the final result corresponds to the mean of its values. The proximity between the instances is measured by a distance metric, calculating the nearest neighbours. This distance metric intends to minimize the distance between two similarly classified instances and to maximize the distance between instances with different classes. Different distance metrics can be used (for instance Euclidean, Manhattan and Chebychev), depending on the type of the features and on the user [52].

Naive Bayes

Naïve Bayes classifier is a very simple Bayesian network, a graphical model for probability relationships among a set of variables which structure is a directed acyclic graph (DAG) and the nodes are in one-to-one correspondence with the features. The arcs of this network represents casual influences among the features. The process of learning a Bayesian networks starts by learning the directed acyclic graph structure of the network and determine its parameters. Naïve Bayes networks are composed of director acyclic graphs with one unobserved node and several observed nodes. Naïve Bayes networks assumes that these several observed nodes are independent of each other, however, this assumption is often wrong, leading to a less accurate classification than algorithms like ANNs (further presented in section 3.1.4. Artificial Neural Networks). There has been several attempts to improve NB performance trying to avoid the independence assumption. Naïve Bayes uses a single probability distribution to summarize the data and discriminate classes, being robust to missing values that will not impact the final

decision [52]. In short, NB is the simplest Bayesian classifier that is built with the assumption of conditional independence of the predicted values.

Random Forests

Random Forests (RF) are powerful machine learning classifiers, based on an ensemble learning algorithm developed by Leo Breiman, being an effective tool in prediction [58]. RF are based on the bagging method and intend to create a strong and stable classification, using a combination of several decision trees [59].

A RF combines the classifications made by each decision tree, being necessary to the user to set the number of trees to be built and also the number of random split variables [58]. The random component of the algorithm is present on two steps during the training: selecting the node tests and on growing the tree, when a random subset of the variables is used, although each tree is already built with a bootstrap sample of the data [60]. RFs have several advantages: their non-parametric nature; high classification accuracy; efficient on large datasets; can handle thousands of input variables without variable deletion; robust to outliers and noise; give estimates of what variables are important in the classification; return a measure of error based on the out-of-bag cases for each fitted tree; do not overfit easily [58][61].

3.1.4.Artificial Neural Networks (ANNs)

Artificial neural networks (ANNs) are computation systems inspired by biological neural networks designed to solve problems like pattern classification and recognition, prediction, optimization, associative memory, control, clustering and function approximation [67].

Basically, ANNs are modelled based on an analogy with the structure and behaviour of neurons in the human brain consisting of many simple, densely connected processing units. This basis on the human brain is due to its desirable characteristics like massive parallelism, learning ability, generalization ability, adaptivity, inherent contextual information processing, low energy consumption, and many others [67].

A neuron is a biological cell that processes information, receives signals, called impulses, from other neurons and transmits signals, by synapses, to other neurons [67][68][69]. In ANNs, artificial

neurons, called nodes of the network, are organized into layers and are connected to several other nodes, each with a different output value. The first layer is the set of input units, and only nodes from consecutive layers are connected. Each node has an output value, calculated by an activation function applied over the activation value. This activation is based on all its input values (the output of the nodes connected to it) times the respective connection weight.

These nodes can transform, through an activation function, the input information to another form of output. There are four main possible activation functions usually used on ANNs: sigmoid, hyperbolic tangent (TANH), rectified linear unit (ReLU) or softmax functions [71]. The Rectified Linear unit, ReLU, is a non-linear activation function very popular that learns faster than other activation functions in networks with many layers. This function only activates a node if the input is positive or 0, giving as output the input itself, otherwise, the output is 0. ReLU is the most used activation function nowadays for neural networks with more than one layer [72]. The sigmoid function, also called logistic function, is differentiable, monotonic and the curve looks like an S-shape, taking values between 0 and 1. Both graphics, ReLU and sigmoid, and each activation function are illustrated on Figure 3.

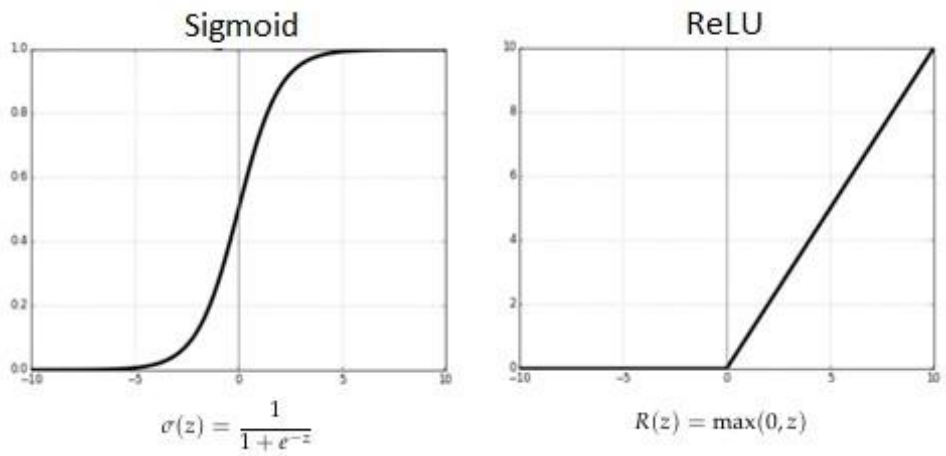

Figure 3. Sigmoid and ReLU activation functions. Adapted from [71].

The synapses correspond to the edges of the network, and the knowledge is presented on the connection between nodes as weights, where each connection has a weight associated [73]. Learning is about finding weights that make the network perform as desired [69].

According to how nodes are interconnected, there are different network architectures that can be grouped into two categories: feed-forward networks and recurrent networks, where both can be represented by a graph. In the feed-forward networks graphs have no loops and, in the most common

family of this category, called multilayer perceptron, the neurons are organized into layers that have unidirectional connections between them. This category consists on networks that produce only one set of output values from a given input, that is, static networks. Another feature of feed-forward networks is the fact that the previous networks state does not influence their response to an input, being memory-less networks.

Contrary to feedforward networks, in recurrent networks, or feedback networks, loops occur because of feedback connection, that is, information flows in both forward and backward directions. Feedback networks are dynamic systems since, when an input pattern is presented, the neuron outputs are computed, and the inputs to each neuron are modified leading the network to enter a new state [67].

Several distinguishing features of ANNs make them valuable and attractive, for instance: ANNs are data-driven self-adaptive methods, they learn from examples and capture subtle functional relationships among the data; ANNs can generalize, being able to correctly infer the unseen part of a population not being sensitive to noise; ANNs are also universal functional approximators, where the network is able to approximate any continuous function to any desired accuracy; ANNs are nonlinear, useful on real world systems that are often nonlinear; ANNs are flexible, with a big applicability domain and; ANNs do a massively parallel processing allowing complex tasks to be performed in a short time [70].

ANNs already proved to be useful in the health area, such as on diagnosing myocardial infarcts and arrhythmias from electrocardiograms and interpreting radiographs and magnetic resonance images, being appropriate for situations with large data sets, problems with nonlinear structure and multivariate time series forecasting problems [68] [70].

Gradient Descent

The gradient descent is an optimization method that intends to optimize the objective function. This algorithm basically “moves” between nodes, measuring the change in the error caused by weight changes, choosing the one that produces the lowest error. Then, the overall loss across all the training examples is calculated. Then, the gradient is calculated and the parameter vector is updated, that is, occurs a change of the weight in the direction of less error. This process is repeated until the weight arrives to a point where the error cannot go lower. The gradient is produced by the derivative of the loss function and gives the direction for the next step in the optimization algorithm [71].

The Stochastic Gradient Descent (SGD) calculates the gradient and updates the parameter vector after each training sample. This algorithm, for an input vector with a few examples, computes the outputs, errors and the average gradient for those examples and, then, adjusts the weights. This process is repeated for each small set of examples (batch), until the average of the objective function stabilizes, achieving the greatest accuracy. The stochastic part is due to each small set of examples provides a noisy estimate of the average gradient over all examples [72].

Backpropagation

The backpropagation algorithm is one of the most used training algorithms that minimize the cost (error) function for neural networks. It is an efficient gradient descent method applied to ANNs that can solve the problem of calculating the partial gradient for each of the weights in the network [73]. This method uses heuristic approaches, usually involving small modifications to the system parameters to improve system performance [74]. The term backpropagation is due to the fact that the error presented on the output is propagated backwards from the output layer to the hidden layer and then to the input layer. At each iteration occurs a forward activation to produce a solution and a backward propagation to modify the weights [75].

3.1.5.Data Processing

Feature Selection

Feature selection is a central problem in machine learning, consisting on identifying a representative sets of features for model construction. For instance, with feature selection it is possible to remove features of the data that are irrelevant to the task to be learned, leading to an easier and less time consuming execution of the learning algorithm. Thus, feature selection leads to a reduced execution time, a reduction in the amount of data needed to achieve learning, an improved predictive accuracy and a learned knowledge more compact and easily understood [76].

There are essentially three feature selection methods: wrapper, filter and embedded methods. The first method consists on applying the feature selection process coupled with the learning algorithm that is to ultimately be applied to the data. Filter methods use heuristics based on general characteristics of the data, ranking them, and then making a feature selection independent of the classifier. Although, in terms of the final predictive accuracy, wrappers often give better results than filters, they are

computationally more expensive, can be intractable for large databases containing many features and are less general [76]. The last method, embedded, performs variable selection in the process of training being specific to some classifiers [77].

3.2. Deep Learning concepts

3.2.1. Deep Neural Networks (DNNs)

Deep learning is an approach to machine learning that enables computer systems to improve with experience and data, creating a hierarchical representation of the concepts. These hierarchies involve several layers deep, and through this hierarchy of concepts it is possible to learn complicated concepts by building them out of simpler ones [85].

The application of deep learning to neural networks results on Deep Neural Networks (DNNs), a neural network with a deep architecture that has the capacity to learn more complex models [91][86]. Deep neural networks are inspired in the human brain, since it also has a deep architecture, and because humans organize their ideas hierarchically. However, although models with many hidden layers and hidden units are very flexible and allow more degrees of freedom, this leads to a major problem, overfitting [87].

A deep neural network is based on the feed-forward neural network, but with more than one hidden layer, where all the nodes of one layer are connected to those of the next layer, that is, each layer is fully connected to the adjacent layer [88]. This architecture forms an acyclic graph. Each layer of the network can have a different number of neurons, and the neurons activation function is nonlinear. Analogously to ANNs, DNNs can be trained by the backpropagation algorithm using stochastic gradient descent [71] [88]. In order to prevent equal gradients in hidden units in a layer, allowing the backpropagation algorithm to run as supposed, the initial weights are set randomly with small values [87].

Deep Neural Networks are extremely powerful models and have already proven to be very effective on problems like speech recognition and visual object recognition [89].

Overfitting can also occur on deep learning algorithms, and there are already some methods to solve this problem. One is Dropout, a technique that consists on dropping out units and their connections, removing them temporarily from the network, during the training, preventing units to co-adapt too much.

The dropped unit is chosen randomly and is retained with a fixed probability independent of other units. The fixed probability is usually 0.5, however it can be chosen by a validation set [54].

Early stopping is also a method that can reduce overfitting in deep neural networks. This method consists on stopping the gradient descent after only a few iterations, before the model begins to overfit. The number of iterations is chosen, and, after that number of iterations the error on the test set is calculated. The model that produced the lowest error, that is, achieved better performance, will be the final model [88].

3.2.2.Convolutional Neural Networks (CNNs)

Convolutional Neural Networks were designed to process data that are in the form of multiple arrays, like signals (1D), images (2D) and videos (3D) [72]. At convolutional neural networks, the fully connected architecture of DNNs is altered by pruning many of the connections or by connecting patches, to achieve better performance [88].

In CNNs, each layer is 3-dimensional and the architecture of this network involves two types of layers: convolutional and pooling layers. The convolutional layer is organized in feature maps were each node is connected, through a filter bank, to local patches in the feature maps of the previous layer, for instance, each neuron in the first hidden layer will be connected to a small region of the input neurons. The filter bank is composed by a set of weights and all units in a feature map share the same filter bank, while different feature maps in the same layer use different filter bank. This organization allows to easily detect local motifs, thus, the convolutional layer role is to detect local conjunctions of features from the previous layer [55] [72]. Immediately after the convolutional layer, a pooling layer is used taking each feature map output and preparing a condensed feature map. Thus, the pooling layer is used to simplify the information in the output from the convolutional layer. The convolutional layer usually involves more than one feature map, therefore, to each feature map is applied pooling. The repetition of convolution layers followed by pooling layers will create the output model of the CNN, as showed on Figure 4 [55].

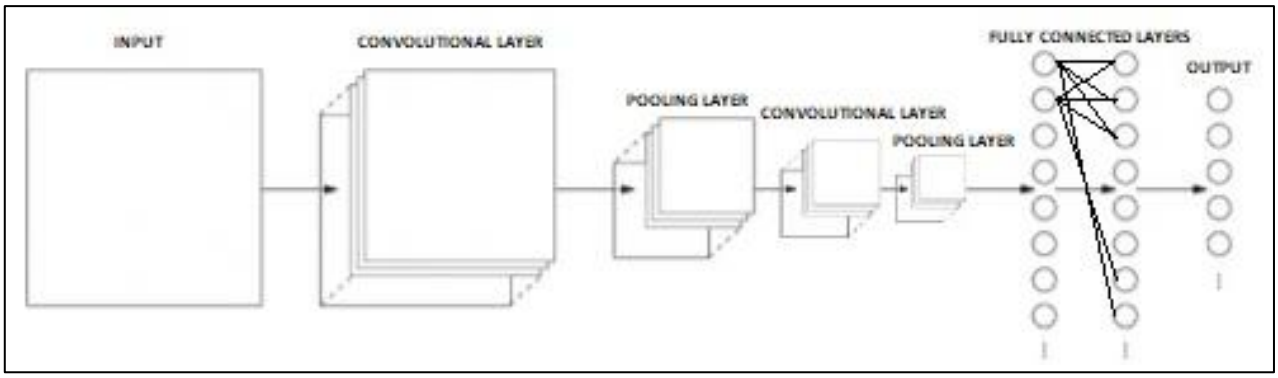

Figure 4. Convolutional Neural Networks. Adapted from [88].

3.2.3.Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a popular and powerful model for sequence tasks using data like text sentences, time-series and biological sequences. The RNN is characterized by its rich dynamics, once the hidden units use a nonlinear activation function, enabling to remember and process past information. At this architecture, the weights are shared across time and the hidden units are able to integrate information over many timesteps, using it to make accurate predictions. Therefore, the output depends on previous computations [90].

The gradients of the RNN, although easy to compute, tend to vanish or exponentially blow-up when backpropagated through time. That said, RNNs are difficult to train with backpropagation algorithm, not learning long-range temporal dependencies. One way to deal with this problem is to include "memory" units that store information over long time periods, using this information on a new activation function. Thus, the gradient at each output will be calculated based on the current time step and on previous ones [91].

One approach that uses memory cells to store information, using this information in a new and more sophisticated activation function, being good at finding and exploiting long range context, is the Long Short Term Memory (LSTM) architecture [3]. This architecture uses special hidden units, that remember inputs, and memory cells that act as gated neurons that copy its own real-valued state and accumulate the external signal. The content of the memory can be cleared by another unit that is multiplicatively gated to the unit. The principal goal of the LSTM is to estimate the conditional probability [91]. Long Short Term Memory model are designed to handle sequences and were already used to predict, with high accuracy, the subcellular location of proteins using only the protein sequence information [3].

The recurrent unit usually used is the traditional tanh. Another type of recurrent unit was recently proposed, referred as a gated recurrent unit (GRU). Both LSTM and GRU, applied to RNN leads to better performance on tasks with long-term dependencies [91].

As mentioned, RNNs are good at modelling sequence data for predictions, including biological sequences, especially LSTM recurrent neural networks because of its memory capacity and its ability to learn temporal dependencies. However, it is impossible to feed characters into a neural network. Therefore, it is necessary to map the characters of each sequence into a numerical representation, into numeric tensors. This process is called encoding and can be done in two different ways. One way is one-hot encoding, where the sequence is transformed into vectors of 0s and 1s, that is, sequences are encoded as binary vectors.

Another form of encoding involves turning all sequences into equal lengths and then turning them into real tensors. Subsequently, the network will have to start with a first layer called Embedding layer. Within the embedding layer, it is possible to be an learn word embedding, which will learn from data, learning word vectors and weights of the neural network. Another possibility is to use pretrained word embeddings, used when there is not enough data available to learn features on its own, being possible to load into the model a precomputed word embedding. This embedding layer is, basically, like a dictionary, that maps integer indices into dense vectors by checking these integers in an internal dictionary and returning the associated vectors that reflects the relationship between words. The embedding layer takes as input a 2D tensor of integers, where all sequences must have the same length, truncating longer sequences and padding with zeros shorter sequences. Then, a 3D floating-point tensor is returned, which can be processed by an RNN or a 1D Convolutional layer. At first, the embedding layer weights, i.e. its internal token vector dictionary, are initialized randomly, and these word vectors are then adjusted by backpropagation during training. However, when using a pretrained embedding layer, this layer's weight matrix is predefined and should not be updated throughout training, so it is necessary to freeze the embedding layer, not allowing it to be trainable [95].

3.2.4.Other networks

There are several other networks that were influential for several years. One of them is the Deep Belief Network (DBN), a generative model, that is, where can be generated for the input activations, by specifying the values of some of the feature neurons and then running the network backward. Thus, this model can learn to reconstruct its input. DBNs can do unsupervised learning, being one of reasons of

interest in this model, that, for example, has already been used as a pre-training layer of DNN, increasing its performance. Despite these attractive features, feedforward and recurrent nets have achieved better results on several tasks like image and speech recognition [55]. One key component of DBNs is the restricted Boltzmann machine (RBM), a two-layer undirected graphical model with a set of binary visible units (first layer) and hidden units (second layer). These two layers are symmetric connected, being this connection represented by a weight matrix. At RBM, there is only one hidden layer and, these models are composed to form DBNs [92].

3.3. Applications to sequence analysis

Over the last years, several studies in proteins and sequence analysis fields were published recurring to machine learning and deep learning methods. Some of these papers are chronologically summarized on Table 3.

Gene expression differences can be very useful in diagnosing diseases. Thus, DNA microarrays with gene expression measurements, with information from tissue and cell samples, can be used to classify tissue samples. The first paper, "Support Vector Machine classification and Validation of Cancer Tissue Samples Using Microarray Expression data" shows a new method that uses an SVM to analyse this kind of data. The data used, an unpublished dataset, consists of expression experiment results from samples from ovarian cancer tissues, normal ovarian tissues and other normal tissues, with 97 802 DNAs clones for each tissue, with a total of 31 tissue samples. However not with high confidence, perfect classification is achieved, and a sample is discovered and confirmed to be wrongly labelled. In order to demonstrate the generality of the method, experiments were also performed for two published datasets, one containing examples of patients with human acute leukemia and the other containing human tumour tissues and normal colon tissues [62].

The second paper in Table 3 presents the application of random forest with microarray data to gene selection for sample classification. Nine microarray datasets were used and the performance of the random forest model proved to be comparable to other classification methods. This approach also proved to select smaller sets of genes, while the gene selection procedure, producing likewise accurate predictions [61].

The third paper in Table 3 presents a transmembrane protein topology predictor, including signal peptide and re-entrant helix prediction, based on a support vector machine. The dataset used for SVM

training contains 131 transmembrane protein sequences with known crystal structures and 416 globular proteins. The method uses evolutionary information and a total of five SVMs, where the outputs were combined in order to return a list of predicted topologies that incorporates signal peptide and re-entrant helix prediction and discrimination between TM and globular proteins. The method achieves a high topology prediction accuracy and was able to distinguish between transmembrane and globular proteins [103].

"MultiLoc2: integrating phylogeny and Gene Ontology terms improves subcellular protein localization prediction" shows an extended version of MultiLoc predictor that incorporates phylogenetic profiles and Gene Ontology terms that improves subcellular protein localization prediction. The model was trained with two different datasets, resulting in two different versions, one specialized for globular proteins, predicting up to five localizations, and the second version can predict all eleven main eukaryotic subcellular localizations. On both versions, MultiLoc2 shows to outperform the previous MultiLoc, specially when trained with the second dataset, allowing to predict eleven main eukaryotic subcellular localizations [102].

At "DNdisorder: Predicting protein disorder using boosting and deep networks", a new sequence based approach, using boosted ensembles of deep networks for the prediction of protein disorder, is presented. On a dataset of 723 proteins, where a 10 fold cross validation was applied, this method proved to be successful, achieving an accuracy of 0.82. The deep neural network was trained using restricted Boltzmann machines and then a backpropagation procedure was used. The features used as input into the disorder predictor were values from a position specific scoring matrix (PSSM), predicted solvent accessibility and secondary structure, and a few statistical characterizations. Each one of these features needed to be encoded as a binary feature [99].

The paper "Prediction of Membrane Transport Proteins and Their Substrate Specificities Using Primary Sequence Information" introduces the use of an SVM model, using sequence information like amino acid composition, dipeptide composition, physico-chemical composition, biochemical composition and position-specific scoring matrices (PSSM), able to predict the substrate specificity of several transporter classes. This paper also presents a model capable of distinguishing transporters from non-transporters [31].

K-mers features, Oligomers of length k, are used for modelling functions and properties of DNA and protein sequences, however, when k becomes large, statistical learning approaches that use these features are very susceptible to noise data. The third article in Table 3 presents a solution to this problem,

introducing alternative feature sets using gapped k-mers, creating a new classifier, gkm-SVM. This new approach demonstrates more accurate predictions of functional genomic regulatory elements and tissue-specific enhancers [97].

The work "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling" compares different recurrent units in recurrent neural networks, LSTM unit and gated recurrent unit (GRU), evaluating them on the task of sequence modelling. More specifically, these were evaluated in polyphonic music modelling, using three polyphonic music datasets, from four different repositories, that contains sequences where each symbol is a dimensional binary vector (93-, 96-, 105- or 108-dimensional), and in speech signal modelling, using two internal datasets provided by Ubisoft where each sequence is an one-dimensional raw audio signal. The LSTM and GRU demonstrate to outperform the traditional tanh unit, specially on speech signal modelling [91].

The ninth paper present on the Table 3 demonstrates that the long short term memory (LSTM) model predict with high accuracy (around 90%) the location of proteins, only based on the protein sequence. The dataset used contains 5959 proteins, and each sequence was truncated to length 1000, to reduce the computational time. A LSTM model where convolutional filters were introduced and that was experimented with an attention mechanism is also presented in this paper, improving the performance, focusing on specific parts of the protein. All models were trained with gradient descent [3].

"DeepCNF-D: Predicting Protein Order/Disorder Regions by Weighted Deep Convolutional Neural Fields" introduces a new method, Deep Convolutional Neural Fields, that combines conditional neural fields (CNF) and deep convolutional neural networks, capable to improve the accuracy of predicting protein order/disorder. This method explores the long-range sequential information and assign different weights for each label during training and prediction. This method was trained and validated using Disorder723 dataset, the one used on the previous paper, and achieved a higher accuracy [100].

The paper "Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning" introduced a new approach, DeepBind, for predicting sequence specificities, based on deep convolutional neural networks. The convolutional network allows detecting motifs that can indicate RNA- and DNA-binding properties, such as variable-width gaps, position interdependence, and secondary motifs, or even suggest potential co-factors. This approach uses a set of sequences and determines a binding score for each sequence [96].

The work "Protein Secondary Structure Prediction Using Deep Convolutional Neural Fields" presents a new model, DeepCNF (Deep Convolutional Neural Fields), that is an extension of CNF, which is an integration of Conditional Random Field (CRF) and shallow neural networks. This new model can predict protein secondary structure, being capable to model complex sequence-structure relationship and interdependency between adjacent secondary structure labels. DeepCNF outperformed currently popular predictors [101].

The eighth paper in Table 3 proposes an approach for predicting binding affinities, once accurately predicting protein-ligand binding affinities can accelerate drug discovery, using 3D-convolutional neural networks. This approach is also compared to other methods and several diverse datasets are used. This new method proves to be easy to use, fast and with good performance [98].

Finally, the last paper in Table 3 presents a novel function based approach to protein annotation and discovery called Deep Semantic Protein Annotation Classification and Exploration, D-SPACE. D-SPACE is a multi-task and multi-label deep neural network that was trained over 70 million proteins and is capable of encoding proteins in high-dimensional representations (embeddings) enabling fast searches for functionally related proteins [106].

Table 3. Published papers of ML and DL over proteins and sequence analysis

Article Title	Year	Method	Ref
Support Vector Machine classification and Validation of Cancer Tissue Samples Using Microarray Expression data	2000	SVM	[62]
Gene Selection and Classification of Microarray Data Using Random Forest	2006	Random Forest	[61]
Transmembrane protein topology prediction using support vector machines	May 2009	SVM	[103]
MultiLoc2: integrating phylogeny and Gene Ontology terms improves	Sep 2009	SVM	[102]

subcellular protein localization prediction			
DNdisorder: Predicting protein disorder using boosting and deep networks	2013	DN, RBM	[99]
Prediction of Membrane Transport Proteins and Their Substrate Specificities Using Primary Sequence Information	June 2014	SVM	[31]
Enhanced Regulatory Sequence Prediction Using Gapped k-mer Features	July 2014	SVM	[97]
Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling	Dec 2014	RNN	[91]
Convolutional LSTM Networks for Subcellular Localization of Proteins	Mar 2015	Convolutional LSTM Networks	[3]
DeepCNF-D: Predicting Protein Order/Disorder Regions by Weighted Deep Convolutional Neural Fields	May 2015	CNN	[100]
Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning	July 2015	CNN	[96]
Protein Secondary Structure Prediction Using Deep Convolutional Neural Fields	2016	CNN	[101]
KDEEP: Protein-Ligand Absolute Binding Affinity Prediction via 3D-Convolutional Neural Networks	2018	CNN	[98]

Deep Semantic Protein Representation for Annotation, Discovery, and Engineering	2018	DNN	[106]

As it is possible to observe, in recent years, several papers developed in the field of bioinformatics applied to proteins use a deep learning approach, due to their great growth and success in recent years.

4. Methods and Software Development

A previous work, developed by Daniel Varzim in his master dissertation in bioinformatics, at the University of Minho, used machine learning techniques to distinguish transport proteins from non-transport proteins, based on features extracted using their amino acid sequences.

In this work, in order to compare both approaches, machine learning and deep learning, the process of dataset creation was similar to Daniel Varzim's work, and machine learning models were also created and tested. The baseline process developed is summarized in the workflow presented in Figure 5.

Daniel Varzim dissertation can be accessed in https://repositorium.sdum.uminho.pt/bitstream/1822/47386/1/Daniel%20Torres%20Varzim%20Faria.pdf and the code developed in his work is available in the URL: https://github.com/DanielVarzim/Master-s-Thesis-.

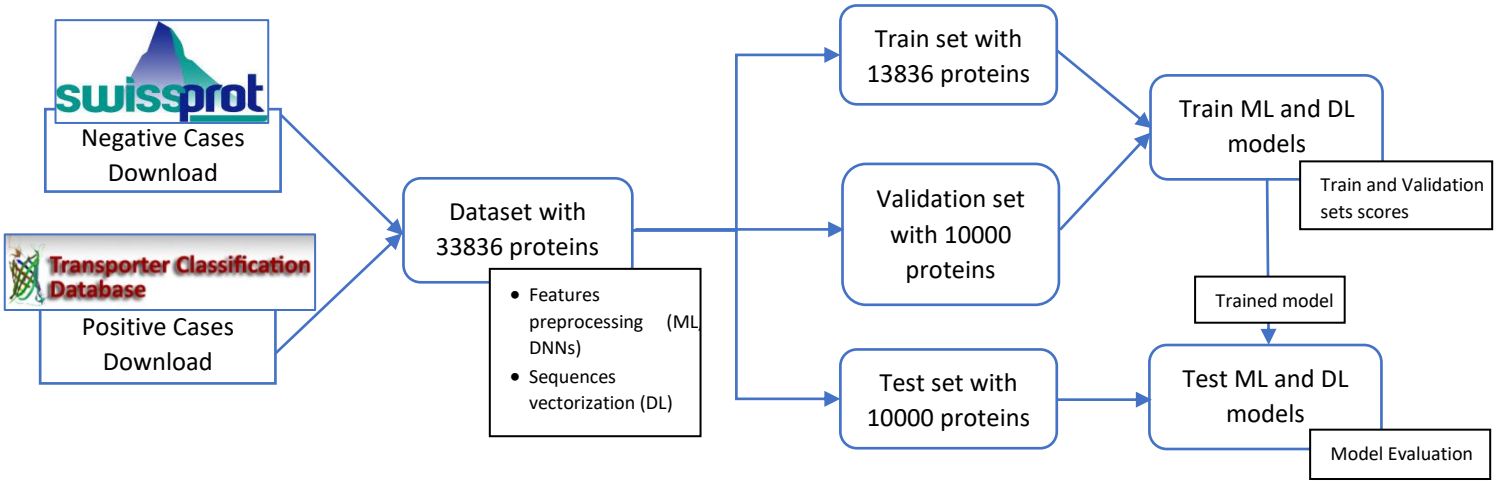

Figure 5. Workflow of the code developed.

4.1. Data

The data for both approaches contains an equal percentage of positive and negative cases, where the positive cases (transport proteins) were obtained by downloading the FASTA file from the TCDB database, containing all TCDB's proteins, a total of 18004 transporter proteins. These 18004 transporter proteins were filtered, selecting only proteins whose sequence contains more than 20 amino acids and less than 1000, thus removing some outliers that could harm the quality of the dataset and its analysis,

turning into 16918 positive cases. A brief overview of the functions developed to obtain the positive cases is given in Table 4.

The negative cases (non-transport protein) were obtained through the Swiss-Prot database, which contains well annotated and reviewed proteins, containing a total of 560,118 proteins. In order to only obtain non-transport proteins, the Swiss-Prot database was filtered with the query <NOT “transporter protein”>, <NOT “transmembrane”> and <NOT “transporter activity”>, resulting in a total of 443,040 non-transport proteins. Finally, 16918 negative cases were randomly chosen from this dataset, to have the same number of negative cases and positive cases. Although they are randomly chosen, it is important to ensure that the non-transport proteins chosen are not longer than 1000 amino acids or shorter than 20 amino acids, the same size range of the transporter proteins. In this sense, it was necessary to verify the length of the randomly chosen proteins to select new proteins if the sequence length was not between 20 and 1000.

To obtain the negative cases, a code was developed to select data, being the main functions briefly introduced in Table 5. To do this, this code starts by reading the dataset with 443,040 non-transport proteins and then randomly selects 16918 numbers from 0 to 443040, so that the number of negative cases selected is the same as the number of positive cases. Then, using the FASTA file, the sequences are obtained and it is verified if their size is within the supposed range, 20 and 1000. If some protein does not satisfy this property, a new random number is chosen to replace that protein, and this process is constantly done until all proteins have sizes between 20 and 1000. Finally, using the FASTA file containing all the negative cases, the proteins information from the lines corresponding to the numbers randomly chosen that satisfied the length property are selected and written in a new fasta file containing now the SwissProt information from the 16918 negative cases.

Table 4. Functions developed to select the positive cases.

	Functions	**Description**
	Get fasta id	Returns the fasta id of the protein

Positive Cases Code	Get sequence	Receives an index and gets the sequence of the correspondent protein from the dataset with all non-transport proteins.
	Select Positive Cases	Reads the file containing all transport proteins and filters them by their sequence length, returning the fasta id from the ones that pass the length property.

Table 5. Functions developed to select the negative cases and create it csv file.

	Functions	**Description**
Negative Cases Code	Chose rand	Returns a list of 16918 randomly chosen negative cases from the 473,916 total negative cases, making sure the sequence sizes are between 20 and 1000
	Get sequence	Receives an index and gets the sequence of the correspondent protein from the dataset with all non-transport proteins.
	Save Fasta	Reads a file containing all the negative cases and writes an fasta with the randomly chosen cases that pass the length property

Since there are already two FASTA files containing the positive cases and the negative cases, it is necessary to join them into a single dataset. To do that, a data creation code was developed where the Fasta ID, Uniprot Accession ID, Sequence, Is Transporter, TCDB ID and Taxonomy Domain information for each protein are stored into a *DataDoms* Comma Separated Values (CSV) file, i.e., a plain text file that contains a list of data separated by commas. To begin with, an array with the referred columns is started and the FASTA file containing the negative data is read. For each protein present, the information needed to fill the dataset is obtained from the one present in the file, except for the domain taxonomy that is obtained by accessing Uniprot. This way, using the functionalities of the BioPython library, using the *SeqIO* parser of FASTA files that creates a record where information can be extracted, it is possible to obtain the Fasta ID and Sequence of each protein. Through Fasta ID, dividing it, it was also possible to get the Accession number and the TCDB ID. Finally, through Biopython's *Entrez.efetch* it is possible to access

Uniprot using the protein's Accession number and, using *SeqIO* again, it is possible to obtain the protein's taxonomy domain. However, if the protein information cannot be accessed with the *Entrez.efetch* module, Uniprot is accessed remotely to obtain the Taxonomy domain using the *Requests* library. This process is similarly done for the positive proteins, joining all the proteins in the same array and finally creating a CSV file that contains all the positive and negative cases. A brief explanation of the functions developed to perform this process can be found in the Table 6.

Table 6. Functions developed to create the base dataset containing the positive and negative cases together.

	Functions	**Description**
Data Creation Code	Reverse	Returns the text backwards.
	Get Fasta ID	Returns the Id of the fasta record
	Get Uniprot Accession	Returns the Uniprot Acession number of the record
	Get sequence	Returns the record´s sequence
	Check if Transporter	Verifies if the record is a transporter protein (returns 1) or a negative case (returns 0), by checking if the first three characters of record id match "gnl" (transporter protein)
	Get TCDB ID	Returns the TCDB ID by reading the record id backwards until the character "|" is found and using the "reverse" function
	Get Domain	Returns the Taxonomy Domain by accessing the Uniprot database.
	Negative Cases	Reads the FASTA file containing the selected negative cases and, for each protein, obtains the information from the previous functions, adding this information to an array
	Positive Cases	Reads the text file containing all TCDB proteins and obtains the information from the previous functions only for the proteins that are in the list returned from the Positive Cases Code, adding this information to the same array as negative cases.
	Create Joined Data	Convert the array containing all cases into a DataFrame and finally saves this information into a csv file called *DataDoms*, containing all positive and negative cases.

4.2. Features and Protein sequences

The process of creating machine learning models involves a good set of features (input attributes). After joining the positive and negatives cases into a single dataset, for each protein, 8 features were generated using their amino acid sequence as basis:

1. **Amino acid composition:** This feature counts the number of times that each amino acid is present in the protein sequence, corresponding to an array with 20 columns, and divide this count by the length of the sequence;
2. **Amino acid physicochemical composition:** Some amino acids have specific physicochemical properties. This feature is an array of 11 columns were each column contains the sum of the times each amino acid referred to one physicochemical property occurs in the protein sequence dividing it by the length of the sequence. The 11 groups of properties taken into account were: charged amino acids (D, E, K, H, and R), 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).
3. **Dipeptide composition:** contains the number of times a specific dipeptide occurs divided by the total number of dipeptides, that is, the number of amino acids on the sequence minus one. It is an array with a total of 400 columns.
4. **Alpha helices:** contains the number of alpha helices estimated by the Phobius tool, having only one column.
5. **Signal peptides:** this array with only one column contains two possible values, “1” for proteins with an estimated signal peptide, or “0” for proteins with no estimated signal peptide. This estimated signal peptide for each protein is obtained through the Phobius tool.
6. **Beta barrel:** The BOMP tool predicts a beta formation or not, and this presence is represented in the array with “1” if a beta formation is predicted and with “0” if not.
7. **Subcellular location:** The LocTree3 tool predict the subcellular location of each protein, that is represented as a number between 0 and 24, represented in binary on an array of 25 columns. Each number represents a different subcellular location: 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".

8. **Number of transporter related Pfam domains:** contains the number of transporter related Pfam domains for each sequence.

In order to obtain those features for each protein, a features creation code was developed, whose functions are briefly presented in Table 7. Here, for each feature, a CSV file is created and only after all files are created, corresponding to all features, a final file is compiled with all features. First, a directory where all the CSV files will be stored is created.

Several features can be obtained just through the amino acid sequence of the proteins. The "Amino acid composition" is obtained by counting the presence of each amino acid in the sequence and dividing this count by the size of the sequence. For the "Amino acid physicochemical composition" feature, amino acids representing some physicochemical properties present in the sequence are counted and summed and them this sum is divided by the sequence length, for each sequence. The physicochemical properties observed are if it is Charged (amino acids D,E,K,H and R), Aliphatic (I, L, V), Aromatic (F,H, W, Y), Polar (D, E, R, K, Q, N), Neutral (A, G, H, P, S, T, Y), Hydrophobic (C, F, I, L, M, V, W), Positive Charged (K, R, H), Negative Charged (D, E), Tiny (A, C, D, G, S, T), Small (E, H, I, L, K, M, N, P, Q, V), Large (F, R, W, Y). Lastly, the "Dipeptide Composition" feature is also obtained by the amino acid sequence, where a counting of the presence of all possible dipeptides in each sequence is made, further dividing this count by the total of possible dipeptides in the sequence.

The "Number of alpha helices" and "Signal peptide" features are both achieved using a Phobius REST API where, using the sequence of each protein it is possible to obtain the response URL information from the Phobius web site, containing the results for each protein. Thus, filtering this response, it is possible to get the number of alpha helices and if there is a signal peptide (1) or not (0). During this process a text file is also created with the Phobius IDs of all dataset proteins. The REST API, Representational State Transfer, allows to define a set of functions that can perform requests and receive

responses via HTTP from some web application, such as Phobius, obtaining the necessary information from a web site via any programming language.

The “Beta barrel” feature is achieved using a BOMP REST API, where, similarly to the previous feature, using the protein sequence it is possible to access the URL response from the BOMP website, filtering this response to check if the protein has a beta barrel conformation. As in the previous feature, during this process a txt file is created with the BOMP IDs of all dataset proteins.

The feature “Subcellular location” is obtained using a LocTree3 REST API. With the sequence and the domain of the protein it is possible to get the LocTree3JOB ID, using the LocTree3 REST API. Then, with these IDs the response URL from the LocTree3 web site is obtained and filtered, being possible to get the protein subcellular location. This subcellular location is rated within 1 and 24, where each number is a different possibility of subcellular location. This feature will be one-hot encoded using a binary representation, i.e., an array of length 25, with 24 zeros and 1 one that corresponds to the subcellular location of that protein.

The “Number of transporter related Pfam domains” uses the NCBI and the Uniprot to obtain all Pfam domains for each protein. Then, comparing this Pfams to a list that contains all possible transporter related Pfams domains, it is possible to count the number of transporter related Pfam domains that each protein has.

Table 7. Functions developed to obtain the features for all proteins and to create an csv file with all features.

	Functions	**Description**
	Count Aminoacid	Returns the number of a given aminoacid in a sequence
	Create Aminoacid Composition	Returns an array with the aminoacid composition of each example in the dataset
	Create aminoacid physico chemical composition	Returns an array with the aminoacid composition based on the physico-chemical properties of each example in the dataset
	Count dipeptide	Returns the number of a given aminoacid in a sequence

Features Creation Code	Dipeptide Composition	Counts each possible dipeptide in the sequence, further dividing that count by the number of possible dipeptides in the sequence.
	Create Dipeptide Composition	Returns an array with the dipeptide composition of each example in the dataset
	Create PhobiusJobID	Creates a text file with all Phobius Job IDs for all proteins
	Create num alphahelices signalpeptide	Returns an array with the number of alpha helices for each protein and if it has a signal peptide (1) or not (0)
	Create BOMP Job IDs	Creates a text file with all BOMP Job IDs for all proteins
	Create BetaBarrels	Returns an array with the number of beta-barrels
	Create LocTree3JobID	Creates a text file with the LocTree3 ID and another txt file with the ReqID
	Create location prediction	Returns an array with the Protein Subcellular Location, in binary form.
	Create transporter related Pfam domains	Creates a text file with all Transport Pfam domains that are found in the proteins of the dataset. Returns an array with the number of Transport Pfam Domains that each protein contains.
	Pfam domains	Returns a list with only the names of all Pfam Domains related to transport proteins
	Create Features	Executes the function for each feature and creates an csv file for each feature
	Create file with all Features	Read all CSV files corresponding to all features and join them information into one array, saving this array in one only CSV file that now contains all features for proteins, all together.

To execute deep learning models, as convolutional neural networks or recurrent neural networks, the input attribute was changed. In this deep learning approach, protein sequences were directly used as

input attributes. The process for obtaining the protein sequence dataset will be explained in the section ***4.3. Dataset creation and pre-processing***.

The models created based on the features previously showed intend to separate transport proteins from non-transport proteins. Thus, the output attribute, called "Is Transporter" will consist on one column array with two possible values, "1" assigned to transport proteins and "0" assigned to non-transport proteins. Based on the column "Is Transporter" from the dataset containing all cases, the code to create the output attribute creates an array with 1 or 0´s depending if the protein is a transporter (1) or a negative case (0) and saves a CSV file with this information in an new *Out_Attributes* folder.

This is classified as a binary classification problem. The output attribute created is used in both approaches, machine learning and deep learning approaches, while the input attribute changes.

4.3. Dataset creation and pre-processing

Two different datasets were created, one composed by the features created as shown in the previous section and the output attribute, and the other composed by the protein aminoacid sequences and the output attribute.

The first dataset, used on machine learning models and DNNs, is a matrix of 461 columns (features and output attribute) and 33836 rows, half of which extracted from the TCDB database (positive cases) and the other half are negative cases randomly chosen from a dataset of non-transport proteins extracted from Swiss-Prot database.

The second dataset contains the aminoacid sequences from the exact same proteins present on the first dataset and also the output attribute. It should be emphasized that the order of the proteins on both datasets is the same, so that the future division on train set, validation set and test set in every model contains the same proteins.

Before creating both datasets, the base dataset that contains the basic information extracted from TCDB and Swiss-Prot databases was randomly shuffled to mix the positive and negative cases. Only after this reorganization of the base dataset, the two different datasets used on the models were created.

To create the dataset and then shuffle it, a mixed dataset creation code was developed, with the functions briefly presented on Table 8. First, this code merges the two base datasets into one, creating a dataset with all features and its output from all proteins. Then, this dataset will be mixed, randomly

reordering the proteins (rows) so that the positive cases are not all together, as the negative cases, for further division of the dataset into training, validation and test data. This mixed dataset is saved as a CSV file *Dataset_mixed*. Finally, the new mixed dataset is split in two, separating the features from the output and new CSV files are created for each part. Now, there are two CSV files, one containing the Features and one containing the Output, both with mixed data, in the same order.

Table 8. Functions developed to create a Dataset with mix data.

	Functions	**Description**
Create Mix Dataset Code	Insert features Outattribute	Merges the features and the output into one dataset, creating its csv file.
	Mix data	Randomly mix the proteins from the dataset containing both the features and the outputs, saving it in a new csv file.
	Split features Outattributes	Using the mixed data, splits the features from the outputs and saves both information, each in a csv file (*Features_Dataset_mixed* and *Out_Attributes_Dataset_mixed*).

After creating the mixed dataset, a new code was developed to create the mixed dataset with the sequences to use it in the RNNs models. The code starts by reading the mixed dataset containing the features, used on machine learning code and DNNs, and selecting all proteins ids in the exact same order that they are on the *Features_Dataset_mixed* csv file. After collecting the ids, the original dataset, *DataDoms* csv file, is read and the protein sequences are stored. Finally, a new CSV file is created, *Data_mixed_seqs*, containing all protein sequences in the same order as the dataset containing the features, *Features_Dataset_mixed* csv file. It is important that both datasets contain the proteins in the exact same order so that in the future the proteins used as train set, validation set and test set will be the same in all models.

The dataset containing the features needed to be pre-processed and transformed, once it may contain missing values and not be standardized or scaled. This preprocessing process was performed after the creation of the mixed dataset and before the use of data in machine learning models, executed at the beginning of the code developed to create the models when reading the data. This whole process

was performed using the Scikit-learn package, a Python toolkit with simple and efficient tools for data mining and data analysis. Scikit-learn is a well-designed machine learning package with tools to help classification problems, regression problems, clustering, dimensionality reduction, model selection and preprocessing, and also designed to interact with other python packages such as NumPy and SciPy. This package will be further used to created and train the machine learning models.

Thus, by using Scikit-learn's *Imputer* preprocessing feature it is possible to replace all missing values ("Nan") with the column mean and, using the preprocessing feature *StandardScaler* it is possible to standardize the dataset by removing the mean and scale to unit variance. The standardization process was done only for non-binary features, i.e. the feature with the number of alpha helices and the feature with the number of transport related Pfam domains. Once just two features needed to be standardized, this process to standardize the dataset was applied when obtaining these two features. A variance threshold filter was also used, to remove features with zero variance, i.e., redundant features that had the same value for every protein that do not contribute to a better discrimination of the classes. This *Variance Threshold* filter was also applied with the scikit-learn package using its feature selection model.

The dataset used on the deep learning approach contains protein sequences. In order to use them as input, it is first necessary to convert them into sequence of integers that can be interpreted by the models, using the class *Tokenizer* from the library Keras, using the backend TensorFlow (that will be further introduced), that enables to vectorize the text data. Then, it is still necessary to work around the variable sequence lengths problem by filling the sequences with the required number of zeros to have a common length in all sequences (padding), where the length of the longest dataset sequence was used.

Keras is a high-level neural networks API, written in Python and capable of running on top of several backends as TensorFlow, Theano and Microsoft Cognitive Toolkit (CNTK) [95] and of running seamlessly on CPU and GPU. Keras is a Python library that enables the construction and training of several deep learning models, having models like neural layers, cost functions, optimizers, initialization schemes, activation functions and regularization schemes.

As mentioned, Keras relies on a specialized, well-optimized tensor library, serving as its backend engine. The Keras backend used in this development was the TensorFlow. TensorFlow is a deep learning open-source software, built in C++, that operates at large scale using dataflow graphs to represent all computation, shared state and the individual mathematical operations, the parameters and their update rules, and the input pre-processing. This system supports advanced machine learning algorithms that contain conditional and iterative control flow, and contains tools that enable the visualization of networks

being possible to follow the training progress [93]. TensorFlow uses a declarative programming paradigm, that is, users can focus on the symbolic definition of what needs to be computed [94].

4.4. Models

As mentioned earlier, both machine learning and deep learning approaches were taken.

In the first approach, the dataset containing the features is used in seven different machine learning models, all applied using the Scikit-learn package: **Naive Bayes (NB)** model using the *GaussianNB* function as base estimator; **Decision tree** model using *ExtraTreesClassifiers* (ET) function; **Support vector machine (SVM)** model using the Support Vector Classification *SVC* function; **K-nearest neighbours (KNN)** model using *KNeighborsClassifiers* function; **Logistic Regression (LR)** model using *linear_model.LogisticRegression* function; **Random Forest (RF)** using *RandomForestClassifier* function; **Gradient Boosting (GB)** using *GradientBoostingClassifier* function. All these models were firstly trained and tested with their default hyperparametes, presented on Table 9. The data was separated into a train set of 13836 proteins (the first 13836 proteins of the dataset) to train the models, a validation set of 10000 proteins and a test of 10000 to test the models and obtain their metrics scores.

Table 9. Information about the ML models trained.

Model	Scikit-learn Module	Section	Hyperparameters
Naive Bayes (NB)	sklearn.naive_bayes	3.1.3	priors: None; var_smoothing: 1e-9
Decision Tree	sklearn.ensemble	3.1.3	n_estimators: 10; criterion: 'gini'; max_deph: None; min_samples_split: 2; min_samples_leaf: 1
K-nearest neighbours (KNN)	sklearn.neighbors	3.1.3	n_neighbours: 5; weights: 'uniform'; algorithm: 'auto'; leaf_size = 30
Support Vector Machine (SVM)	sklearn.svm	3.1.3	C: 1.0; kernel: 'rbf'; degree: 3; gamma: 'auto'; coef: 0.0
Logistic Regression (LR)	sklearn.linear_model	3.1.3	penalty: 'l2'; dual: False; tot: 0.0001; C: 1.0
Random Forest(RF)	sklearn.ensemble	3.1.3	n_estimators: 10; criterion: 'gini'; max_deph: None; min_samples_split: 2; min_samples_leaf: 1
Gradient Boosting (GB)	sklearn.ensemble	3.1.3	loss: 'deviance'; learning_rate: 0.1; n_estimatiors: 100; subsample: 1.0; criterion: 'friedman_mse'

In order to improve the model's performance, some techniques of ensemble methods were applied. Some resampling methods were implemented along the project, such as Bagging classifier, used in each machine learning model using the Sklearn package and its ensemble module.

The machine learning models developed were also implemented with two voting classifiers, using *VotingClassifier* function from Sklearn package. This Voting Classifier prediction method can use a majority voting or a weighted majority voting. In the majority voting, Hard Vote Classifier, the final

prediction corresponds to the class predicted by most of the models. Weighted voting classifiers considers that each model has a weight (given by the user) and the models with bigger weight have more influence in the final result.

In order to train and test all these models, a model creation and evaluation code was developed. This code starts by creating a directory where machine learning models will be stored, "ML Models" and by loading the input dataset (*Features_Dataset_mixed* csv file) and the output dataset (*Out_Attributes_Dataset_mixed* csv file) as the variables "Input" and "Output", respectively.

After loading the variables containing the data, the Input variable will be treated and pre-processed, as referred in section ***4.3***, using the Scikit-learn package and its preprocessing and *feature_selection* modules.

Finally, using the Sklearn package, several machine learning models were trained and tested with the train set (the first 13836 proteins of the dataset and test set and the last 10000 proteins of the dataset), obtaining different metrics scores and its confusion matrix: Gaussian Naïve Bayer model, Decision Tree, K-Nearest Neighbors (KNN), Logistic Regression, Random Forest Classifier, Gradient Boosting Classifier and SVM. The first five algorithms were executed applying the Bagging Classifier ensemble method. All these models are saved on the "ML Models" folder.

Finally, several voting classifiers are trained and tested and accuracy, ROC-AUC and F1 scores are obtained for hard vote classifiers. This process is repeated six times, creating different Weighted or non-weighted Hard votes and Soft votes classifiers. The first is a hard vote classifier without weights, where its final prediction corresponds to the class predicted by most of the models. The second and the third are weighted hard votes, with the weights [1,2,3,2,3,3,1] and [4,6,10,6,10,7,1] respectively, considering that each model has a weight where models with bigger weight have more influence in the final result. The fourth is just a Hard Vote without weights and only for the three models that demonstrated the best performance in the first part where they were trained with their default parameters (GB, RF and KNN). The fifth is actually a Soft Vote with weights [1,2,3,2,3,3,1] and the last one is a Soft Vote without weights. The final result in a Soft Vote is calculated by averaging out the probabilities calculated by each model, and, when is a weighted soft vote, models with a higher weight have more influence in the final result. All the weights are according the order of the trained models, presented on the next chapter on section **5.1**, in Table 10. Their weights are set according to their performance in that same part, where the models that achieved the best performance were assigned higher weights, those that achieved the worst performance lower weights and the models with an average performance were assigned median

weights. In the Weighted Hard Vote two different weights were tested, trying to see if by defining a larger difference between the weights of the best and worst models and by using a ranking with more than three ratings the final result would be influenced.

Since the models are first trained and tested with their default hyperparameters, as explained in the beginning of this section, a model optimization is made only to the model that in the first part demonstrates the best performance. This model optimization allows to find the best combination of hyperparameters for this model and to see if there is any change of improvement in the model performance. There are two common model optimization processes: Randomized Search and Grid Search. The first approach is used when, for example, there is a limited computational power and it is not possible to test all possible combinations of hyperparameters. So, instead of testing all possible combinations, this approach randomly selects a set of combinations and tests them, being this number of iterations defined by the user. The Grid Search approach runs all possible combinations of hyperparameters to find the best model, however, the runtime of this process can be massive.

Due to limited computational power, the Randomized Search model optimization approach was the only one applied, using the validation set to optimize the model. This process is achieved using the *model_selection* module from Sklearn package.

Finally, using the Keras library, the same featured dataset was used to train a **Deep Neural Network (DNN)** model. The code developed starts by reading the dataset containing the features of all proteins and use it to train deep neural network models (DNNs). First the data was separated into train set (size of 13836 proteins), validation set (10000 proteins) and test set (10000 proteins) and then several DNNs with different numbers of hidden layers and hidden units were trained. In the hidden layers of these models and in the last layer the activation function used was the *relu* function and the sigmoid function, respectively. Dropout layers of 0.25, 0.5 or no dropout layer were applied, and different optimizers were tested, *rmsprop, adam* and *sgd_momentum*. The models were tested with 50 epochs and a batch size of 512.

In the deep learning approach, using the Keras library and the TensorFlow backend, **Recurrent Neural Networks (RNN)** and **Convolutional Neural Networks (CNN)** models were developed. Several models were trained, using a different number of hidden layers or hidden units, or even a different type of layer, such as LSTM or GRU. Models combining RNNs with convolutional layers were also

developed. All developed models will be presented in section 5 along with their results. The code developed to train and test the RNN starts by reading and treating the data. After reading the dataset containing the protein sequences the length of the longest sequence is obtained and stored in a variable. Then, through the TensorFlow and Keras libraries using the preprocessing module and the text *Tokenizer* function, the sequences are vectorized, where each amino acid is turned into an integer, and transformed into the size of the previously stored variable, where zeros are added so that all sequences are equal in length to the largest sequence.

After data preprocessing, the various sets needed to train, validate and test the models are created, being the same used in the previous approach.

Finally, using TensorFlow and Keras library, several deep learning models were trained and tested, combining different numbers of hidden layers or of hidden units and also different type of layers and neural networks.

4.5. Performance Evaluation

To evaluate models' performance, after training the models, it is possible to use the metrics module from Scikit-learn that includes several score functions, performance metrics and pairwise metrics and distance computation. Thus, different score functions were computed such as accuracy, F1, ROC-AUC and recall by comparing the true test set outputs with the predicted test set outputs, which were predicted using the predict function that receives the test set inputs. Also, using the metrics module, it was possible to compute the confusion matrix for each model to observe where the models are misclassifying the data and then calculate the specificity score.

The performance evaluation for the deep neural networks was made by using the evaluate function which receives the test set inputs and predicts its outputs, comparing them to the true test set outputs and calculating the accuracy of the model.

For RNNs, the evaluation of the model's performance is similar to the DNN's evaluation, made using the history function which returns the loss and accuracy values of both training and validation data. Then using the evaluate function it is possible to test the model obtaining these same metrics, loss and accuracy, for the test data.

The code of each algorithm developed in this work is freely available in GitHub at "https://github.com/AndreaFMSilva/Thesis".

5. Results and Discussion

5.1. Machine Learning Models

As explained in the previous chapter, several machine learning models were tested and trained with their default hyperparameters using a featured dataset. These models were tested with the combination of the test and validation set. The different score metrics for each model are presented in Table 10.

Table 10. Machine Learning Models performance evaluation for the test set (test + validation sets)

Model	Accuracy	F1	ROC-AUC	Specificity	Recall
Naive Bayes	0.8672	0.8513	0.8672	0.9746	0.7598
ExtraTreeClassifier	0.8765	0.8712	0.8765	0.9178	0.8352
K nearest neighbours	0.8855	0.8802	0.8855	0.9302	0.8408
Logistic Regression	0.879	0.8674	0.879	0.9666	0.7914
GradientBoosting	0.8916	0.8849	0.8916	0.9492	0.8340
RandomForest	0.8817	0.8743	0.8817	0.9404	0.823
SVM	0.8563	0.8339	0.8563	0.9912	0.7215

In this first evaluation, the SVM model showed lowest performance in all scores. The gradient Boosting model was the only one achieving scores slightly above 0.89, on accuracy and ROC-AUC scores, achieving the best performance. The Random Forest and K nearest neighbours models presented very similar performance, close to the Gradient Boosting model, reaching score values of 0.88 in the accuracy and ROC-AUC scores.

As mentioned, the Gradient Boosting model achieves the best performance, however, this result was achieved with the hyperparameters defined by default. So, a Randomized Search model optimization was made for this model, where the number of combinations was limited to 50 and the possible hyperparameters tested are presented on Table 11.

Table 11. Hyperparameters used on model optimization process

Hyperparameter	Values
N° Estimators	[50, 100, 500]
Max Depth	[3, 5, 8]
Learning Rate	[0.05, 0.1, 0.2]
Min Samples Split	[2, 10, 100, 300, 500]
Min Samples Leaf	[1, 10, 50]

The best configuration of hyperparameters found in the Randomized Search process and the respective accuracy score are presented on Table 12. The results obtained in the model optimization process demonstrate an improvement over those obtained previously, presented in Table 10, slightly increasing the accuracy score to 0.899. This indicates that the hyperparameters set by default are not the best configuration for the highest score, being possible to improve its result, testing with even more possible hyperparameters.

Table 12. Best configuration of hyperparameters found in Randomized Search process.

HYPERPARAMETERS	RANDOMIZED SEARCH
N° Estimators	500
Max Depth	8
Learning Rate	0.2
Min Samples Split	100
Min Samples Leaf	1
TEST SET ACCURACY	**0.8991**

After evaluating each model, an ensemble method was also implemented, Hard Voting classifiers, Soft Voting classifiers and Weighted voting classifiers, ensembling the models to combine their prediction capabilities into a better suited model. As mentioned in section 4.4, six different ensemble voting classifiers were implemented with the same train and test set as before. The scores from these ensemble voting classifiers, shown in Table 13, show no significant improvement over the scores presented in Table

10. However, within all these processes, the Hard Vote classifier with only the three best performing models demonstrated the best performance with an accuracy score of 0.894, slightly improving the result from the best performance model obtained with the Gradient Boosting model. The two weighted hard votes implemented showed no major differences in their results, indicating that defining a larger difference between the weights for the best and worst model has no major influence on the result compared to the weight used with a difference of only one unit between the three possible classifications.

Table 13. Accuracy and F1 scores for ensemble voting classifiers.

Ensemble	Models Used	Accuracy	ROC-AUC	F1	Recall
Hard Vote	All	0.8874	0.8881	0.8785	0.8075
Weighted Hard Vote [1,2,3,2,3,3,1]	All	0.8918	0.8923	0.8855	0.8299
Weighted Hard Vote [4,6,10,6,10,7,1]	All	0.8921	0.8927	0.8856	0.8281
Hard Vote	GB, RF, KNN	0.8939	0.8944	0.8823	0.8360
Soft Vote [1,2,1,3,3,2,3]	All	0.8904	0.8904	0.8811	0.8118
Soft Vote	All	0.8821	0.8821	0.8706	0.7930

To understand and evaluate where the data are wrongly predicted, confusion matrices were created for each ML model trained in the beginning. The results from the confusion matrix to all models are presented on the table below (Table 14).

Table 14. Confusion Matrices of the first fold from all models.

NB Model				**ExtraTree model**			
Predicted / Actual	**NO**	**YES**	**All**	**Predicted / Actual**	**NO**	**YES**	**All**
NO	4872	127	4999	**NO**	4588	411	4999
YES	1201	3800	5001	**YES**	824	4177	5001

All	6073	3927	10000	**All**	5412	4588	10000

KNN model				**Logistic Regression model**			
Predicted **Actual**	**NO**	**YES**	**All**	**Predicted** **Actual**	**NO**	**YES**	**All**
NO	4650	349	4999	**NO**	4832	167	4999
YES	796	4205	5001	**YES**	1043	3958	5001
All	5446	4554	10000	**All**	5875	4125	10000

GradientBoosting model				**RandomForest model**			
Predicted **Actual**	**NO**	**YES**	**All**	**Predicted** **Actual**	**NO**	**YES**	**All**
NO	4745	254	4999	**NO**	4701	298	4999
YES	830	4171	5001	**YES**	885	4116	5001
All	5575	4425	10000	**All**	5586	4414	10000

SVM model			
Predicted **Actual**	**NO**	**YES**	**All**
NO	4955	44	4999
YES	1394	3607	5001
All	6349	3651	10000

Analyzing the confusion matrices, it is possible to conclude that all models have similar results, where the number of FNs is higher than the number of FPs, misclassifying transport proteins as non-transport proteins. The SVM model has the higher number of TN, however, is also the model that presents the higher number os FNs, reflecting on its specificity score being the highest of all (0.99), and its recall score being the lowest of all (0.72). On the contrary, the KNN model contains the higher number of TP but it also has the third higher number of FPs, reflecting on its recall score, being the

highest of all (0.84). Lastly, the ExtraTree model contains the higher number of FP, being reflected on its specificity score being the lowest of all (0.91), however being a high value compared to the worst recall score, showing that the models are effectively misclassifying transport proteins as non-transport proteins, not the other way around.

The presence of transport proteins in the negative cases, due to the queries used to create the negative cases dataset, may influence this misclassification, being one of the possible reasons for this misclassification of transport proteins as non-transport proteins.

5.2. Deep Learning Models

5.2.1. Deep Neural Network

To start the deep learning approach, some deep neural networks were trained and tested using the dataset containing protein features, to test whether DNNs can be more effective than machine learning models, using the exact same dataset as input data on all these models. Different DNNs were trained and tested, combining different numbers of hidden layers, hidden units, dropout rates and optimizer. Thus, a model optimization was performed testing different combinations of number of hidden layers (1,2 or 4), numbers of hidden units (64, 128 and 256), dropout rate (0, 0.2, 0.5) and optimizer (*adam, rmrprop and sgd_momentum*), training with 50 epochs and a batch size of 512. The best configuration found, presented on Table 15, achieved the best accuracy score for the test set of 0.8918, showing no significant improvement over the machine learning models trained and tested.

Table 15. Best hyperparameters configuration on DNN model optimization.

Hyperparameters	Value
Number of hidden layers	4
Number of hidden units	128
Dropout rate	0.2
Optimizer	Adam
Epochs	50
Batchsize	512

This deep learning approach using protein features as input showed no improvement compared to machine learning models that use this same dataset as input. However, it has shown to resemble machine learning models, achieving a similar performance.

5.2.2. Recurrent Neural Networks

Several deep learning models, such as RNNs, were developed using the dataset containing the protein sequences, trying to distinguish transport proteins from non-transport proteins and improve the results obtained in the machine learning models which use features as input. Thus, to test and evaluate the models performance the dataset was, once again, divided as mentioned in the previous chapter into the same datasets.

Different types of neural networks were trained and different combinations of numbers of hidden layers and hidden units were tested. All trained models and their characteristics and scores are described in Table 16. All models start with an Embedding layer, and end with a Dense layer with sigmoid function as the activation function, returning 1 output. The models are then compiled with the *rmsprop* optimizer, the *binary_crossentropy* loss function and the metric defined is accuracy. Lastly, the model is trained and validated with the train set and validation set, respectively, with 10 epochs and a batch size of 128, and finally the model is tested with the test set.

Table 16. Deep Learning Models characteristics and scores.

Neural Network	Nº of Hidden Layers	Nº of Hidden units	Train score	Validation score	Test score
CuDNNLSTM	1	32	0. 8317	0.8058	0.8048
CuDNNGRU	1	32	0.8214	0.8202	0.8240
LSTM	1	32	0.7905	0.8252	0.8263
GRU	1	32	0.8236	0.8313	0.8337
Bidirection CuDNNGRU	1	16	0.8083	0.8271	0.8268
1D Convnet	1	32	0.8963	0.8740	0.8795

1D Convnet	2	32	0.8976	0.8834	0.8872
1D Convnet + CuDNNGRU	2	32	0.8972	0.8852	0.8857
1D Convnet + GRU	2	32	0.8932	0.8810	0.8852
1D Convnet + CuDNNLSTM	2	32	0.8669	0.8434	0.8477
1D Convnet + LSTM	2	32	0.8781	0.8735	0.8841
CuDNNLSTM	3	16	0.8296	0.8369	0.8431
1D Convnet + 2 CuDNNGRU + Dropout (0.5)	3	32	0.8932	0.8871	**0.8928**
1D Convnet + CuDNNGRU + Dropout (0.5)	2	128	0.9415	0.8395	0.8482
1D Convnet + CuDNNLSTM + Dropout (0.5)	2	128	0.8894	0.8756	0.8825
2 1D Convnet + 2 CuDNNLSTM + Dropout (0.5)	4	128	0.8894	0.8750	0.8853
1D Convnet + GRU	2	128	0.9407	0.7969	0.8007
1D Convnet + 2 GRU + Dropout (0.5)	3	Convnet – 64 GRU – 32	0.9108	0.8885	0.8925
1D Convnet + 2 CuDNNGRU + Dropout (0.5)	3	128	0.9768	0.8699	0.8758

Almost all models with 1D Convnet overfit the data, some more than others, being better in the train and validation set than in the test set. One way to avoid the overfitting is to add dropout layers.

However, not always the addition of a dropout layer improves the test set score. For example, the last model presented on Table 16 has a very high train score value, however the test score of this model is much lower.

Although overfitting occurs in the vast majority of networks tested, one of them has achieved a slightly higher test score than Gradient Boosting model, with an accuracy of 0.8928, however it cannot be considered a significant improvement. This score was obtained in the model composed by a 1D Convnet layer followed by 2 GRU layers with a dropout layer between them, with a rate of 0.5. One possible reason for no significant improvement is the embedding layer used in every network, which, being a freshly trained network, may not represent the data well. A possible approach is to use a pre-trained embedding layer, preferably trained with protein sequences.

6. Conclusions and Further Work

This work consisted on the application of machine learning and deep learning models to predict, through protein sequences, whether a given protein is a transporter or not.

Seven different machine learning models were trained and tested. Some ensembles of the models, such as Hard Voting classifiers and Weighted voting classifiers, were also implemented. In this machine learning approach, the best score obtained was of 0.8916 in both accuracy and ROC-AUC scores, corresponding to the performance of the Gradient Boosting model. In order to understand where the models were misclassifying the data, their confusion matrices were created and analyzed. Thus, it was observed that the number of FNs was greater than the number of FPs in all models, indicating that transporter proteins predicted as non-transporter proteins is the most common error.

A hyperparameter optimization was also performed in the model that reached the best score, Gradient Boosting, thus slightly improving the accuracy to 0.899 through the Randomized Search process. Finally, an ensemble method was implemented by applying Hard Voting classifiers, Weighted voting classifiers and Soft Voting classifiers by testing different weights and testing only with the best 3 models (GB, RF, KNN). However, no significant improvement over previous scores was obtained, only achieving a similar accuracy score of 0.8939 with the Hard Vote tested with only the best 3 models.

To apply a deep learning based approach, some deep neural networks were trained and tested using the dataset containing the features, thus using the same input as the machine learning models. Here, the main objective was to see if DNNs can match/improve the results obtained with machine learning models when using the same dataset as input data. DNN hyperparameters were optimized, combining different numbers of hidden layers, hidden units, dropout rates and different optimization algorithms, obtaining a maximum accuracy of 0.8918 corresponding to a four layer model, with 128 hidden units, with a dropout rate of 0.2 and Adam optimizer. Thus, with this configuration, no improvement over the machine learning models was verified, matching the best score obtained with the Gradient Boosting model with its default hyperparameters and not far from the score obtained in its model optimization process.

Finally, the objective falls on the application of deep learning models to try to improve the results of the previous approaches, trying only to use the protein sequences as input. The main neural network used in sequences is the RNNs. Thus, several recurrent neural network models were trained and tested by combining different numbers of hidden units, hidden layers, epochs, but mainly by varying the type

of layers, using gated units, memory units, convolutional layers and even bidirectional layers. All models achieved accuracy test set scores greater than 0.80, demonstrating already a reasonable performance in the separation of transporter and non-transporter proteins using only their sequences as input data. The maximum accuracy obtained is of around 0.89, corresponding to the model consisting of a gated recurrent unit with two convolutional layers with dropout layers between them, and only 32 hidden units. Thus, the performance of the best RNN model resembles the performance of the best machine learning model, obtaining very close accuracies. Most models with a convolutional layer have demonstrated overfitting, and not even adding dropout to these networks has been shown to work in every case. Unfortunately, due to the limited computational power and the large dataset, it was only possible to test models with low numbers of hidden units and epochs.

Finally, although a deep learning model capable of identifying transporter proteins with an accuracy of 0.89 was already found, it did not outperform the best machine learning model, Gradient Boosting model, only matching it.

A possible change to improve the performance of the deep learning model is to change the embedding layer. All embedding layers used in the tested models are embeddings that are trained on the model itself with current data, which can lead to poor data classification. A possible improvement would be to use pretrained word embeddings, especially if it is a pretrained model with sequence proteins. It would also be interesting to change the filters used to obtain the negative cases through Swiss-Prot database, since there may be positive cases within the negative cases dataset. But, one more time, to train and test models with more layers and hidden units it would be interesting to do experiments with more computational power.

In short, the proposed objectives were achieved despite the results obtained do not meet the idealized, thus, as mentioned, there is always room for improvement and future work.

Bibliography

[1] Teusink, B., Wiersma, A., Molenaar, D., Francke, C., de Vos, W. M., Siezen, R. J., & Smid, E. J. (2006). Analysis of growth of Lactobacillus plantarum WCFS1 on a complex medium using a genome-scale metabolic model. Journal of Biological Chemistry.

[2] Dias, O., Rocha, M., Ferreira, E. C., & Rocha, I. (2015). Reconstructing genome-scale metabolic models with merlin. Nucleic acids research, 43(8), 3899-3910.

[3] Sønderby, S. K., Sønderby, C. K., Nielsen, H., & Winther, O. (2015, August). Convolutional LSTM networks for subcellular localization of proteins. In International Conference on Algorithms for Computational Biology (pp. 68-80). Springer, Cham.

[4] Ayrton A, Morgan P. Role of transport proteins in drug absorption, distribution and excretion. Xenobiotica, 2001, vol. 31, no. 8/9, 469-497.

[5] Feist A, Henry C, Reed J, Krummenacker M, Joyce A, Karp P, Broadbelt J, Hatzimanikatis V, Palsson B. A genome-scale metabolic reconstruction for Escherichia coli K-12 MG1655 that accounts for 1260 ORFs and thermodynamic information. Mol. Syst. Biol., 2007, vol. 3, no. 121, p. 121.

[6] Wainberg, M., Merico, D., Delong, A., & Frey, B. J. (2018). Deep learning in biomedicine. Nature biotechnology, 36(9), 829.

[7] Saier M, Jr, Tran C, Barabote R. TCDB: the Transporter Classification Database for membrane transport protein analyses and information. Nucleic Acids Research, 2006, Vol. 34, D181–D186.

[8] Awad M, Khanna R. Efficient Learning Machines. Berkeley, CA: Apress, 2015.

[9] Ching T, Himmelstein D, Beaulieu-Jones B, Alexandr A. Kalinin A, Do B, Way G, Ferrero E, Agapow P, Xie W, Rosen G, Lengerich B, Israeli J, Lanchantin J, Woloszynek S, Carpenter A, Shrikumar A, Xu J, Cofer E, Harris D, DeCaprio D, Qi Y, Kundaje A, Peng Y, Wiley L, Segler M, Gitter A, Greene C. Opportunities and obstacles for deep learning in biology and medicine. bioRxiv, 2017.

[10] Veenhoff, L. M., Heuberger, E. H., & Poolman, B. (2002). Quaternary structure and function of transport proteins. Trends in biochemical sciences, 27(5), 242-249.

[11] Kam, N. W. S., & Dai, H. (2005). Carbon nanotubes as intracellular protein transporters: generality and biological functionality. Journal of the American Chemical Society, 127(16), 6021-6026.

[12] Ho, R. H., & Kim, R. B. (2005). Transporters and drug therapy: implications for drug disposition and disease. Clinical Pharmacology & Therapeutics, 78(3), 260-277.

[13] Ayrton, A., & Morgan, P. (2008). Role of transport proteins in drug discovery and development: a pharmaceutical perspective. Xenobiotica, 38(7-8), 676-708.

[14] Busch, W., & Saier, M. H. (2002). The transporter classification (TC) system, 2002. Critical reviews in biochemistry and molecular biology, 37(5), 287-337.

[15] Saier, M. H. (2000). Families of transmembrane sugar transport proteins. Molecular microbiology, 35(4), 699-710.

[16] Saier Jr, M. H., Tran, C. V., & Barabote, R. D. (2006). TCDB: the Transporter Classification Database for membrane transport protein analyses and information. Nucleic acids research, 34(suppl_1), D181-D186.

[17] Aebersold, R. H., Leavitt, J., Saavedra, R. A., Hood, L. E., & Kent, S. B. (1987). Internal amino acid sequence analysis of proteins separated by one-or two-dimensional gel electrophoresis after in situ protease digestion on nitrocellulose. Proceedings of the National Academy of Sciences, 84(20), 6970-6974.

[18] Durbin, R., Eddy, S. R., Krogh, A., & Mitchison, G. (1998). Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge university press.

[19] Stackebrandt, E., & Goebel, B. M. (1994). Taxonomic note: a place for DNA-DNA reassociation and 16S rRNA sequence analysis in the present species definition in bacteriology. International Journal of Systematic and Evolutionary Microbiology, 44(4), 846-849.

[20] Pearson, W. R. (2013). An introduction to sequence similarity ("homology") searching. Current protocols in bioinformatics, 42(1), 3-1.

[21] Madden, T. (2013). The BLAST sequence analysis tool. National Center for Biotechnology Information (US).

[22] Walsh, C. (2006). Posttranslational modification of proteins: expanding nature's inventory. Roberts and Company Publishers.

[23] Staden, R. (1996). The Staden sequence analysis package. Molecular biotechnology, 5(3), 233.

[24] Yu, R. D., Clements, J., & Eddy, S. R. (2011). HMMER web server: interactive sequence similarity searching. Nucleic acids research, 39(suppl_2), W29-W37.

[25] Yu, N. Y., Wagner, J. R., Laird, M. R., Melli, G., Rey, S., Lo, R., ... & Brinkman, F. S. (2010). PSORTb 3.0: improved protein subcellular localization prediction with refined localization subcategories and predictive capabilities for all prokaryotes. Bioinformatics, 26(13), 1608-1615.

[26] Chen, S. A., Ou, Y. Y., Lee, T. Y., & Gromiha, M. M. (2011). Prediction of transporter targets using efficient RBF networks with PSSM profiles and biochemical properties. Bioinformatics, 27(15), 2062-2067.

[27] Chen, H. U., Huang, N. I., & Sun, Z. (2006). SubLoc: a server/client suite for protein subcellular location based on SOAP. Bioinformatics, 22(3), 376-377.

[28] Käll, L., Krogh, A., & Sonnhammer, E. L. (2007). Advantages of combined transmembrane topology and signal peptide prediction—the Phobius web server. Nucleic acids research, 35(suppl_2), W429-W432.

[29] Berven, F. S., Flikka, K., Jensen, H. B., & Eidhammer, I. (2004). BOMP: a program to predict integral β-barrel outer membrane proteins encoded within genomes of Gram-negative bacteria. Nucleic acids research, 32(suppl_2), W394-W399.

[30] Sonnhammer, E. L., Von Heijne, G., & Krogh, A. (1998, July). A hidden Markov model for predicting transmembrane helices in protein sequences. In Ismb (Vol. 6, pp. 175-182).

[31] Mishra, N. K., Chang, J., & Zhao, P. X. (2014). Prediction of membrane transport proteins and their substrate specificities using primary sequence information. PLoS One, 9(6), e100278.

[32] Alballa, M., Aplop, F., & Butler, G. (2018). TranCEP: Predicting transmembrane transport proteins using composition, evolutionary, and positional information. bioRxiv, 293159.

[33] Bork, P. (2000). Powers and pitfalls in sequence analysis: the 70% hurdle. Genome research, 10(4), 398-400.

[34] Apweiler, R., Bairoch, A., Wu, C. H., Barker, W. C., Boeckmann, B., Ferro, S., ... & Martin, M. J. (2004). UniProt: the universal protein knowledgebase. Nucleic acids research, 32(suppl_1), D115-D119.

[35] Benson, D. A., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., & Wheeler, D. L. (2008). GenBank. Nucleic acids research, 36(Database issue), D25.

[36] Miyazaki, S., Sugawara, H., Ikeo, K., Gojobori, T., & Tateno, Y. (2004). DDBJ in the stream of various biological data. Nucleic acids research, 32(suppl_1), D31-D34.

[37] Bairoch, A., & Boeckmann, B. (1991). The SWISS-PROT protein sequence data bank. Nucleic acids research, 19(Suppl), 2247.

[38] Kersey, P. J., Duarte, J., Williams, A., Karavidopoulou, Y., Birney, E., & Apweiler, R. (2004). The International Protein Index: an integrated database for proteomics experiments. Proteomics, 4(7), 1985-1988.

[39] Pruitt, K. D., Tatusova, T., & Maglott, D. R. (2006). NCBI reference sequences (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. Nucleic acids research, 35(suppl_1), D61-D65.

[40] Boeckmann, B., Bairoch, A., Apweiler, R., Blatter, M. C., Estreicher, A., Gasteiger, E., ... & Pilbout, S. (2003). The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. Nucleic acids research, 31(1), 365-370.

[41] Wu, C. H., Yeh, L. S. L., Huang, H., Arminski, L., Castro-Alvear, J., Chen, Y., ... & Vinayaka, C. R. (2003). The protein information resource. Nucleic acids research, 31(1), 345-347.

[42] Bairoch, A., Apweiler, R., Wu, C. H., Barker, W. C., Boeckmann, B., Ferro, S., ... & Martin, M. J. (2005). The universal protein resource (UniProt). Nucleic acids research, 33(suppl_1), D154-D159.

[43] Ren, Q., Kang, K. H., & Paulsen, I. T. (2004). TransportDB: a relational database of cellular membrane transport systems. Nucleic acids research, 32(suppl_1), D284-D288.

[44] Ren, Q., Chen, K., & Paulsen, I. T. (2006). TransportDB: a comprehensive database resource for cytoplasmic membrane transport systems and outer membrane channels. Nucleic acids research, 35(suppl_1), D274-D279.

[45] Åkesson, M., Förster, J., & Nielsen, J. (2004). Integration of gene expression data into genome-scale metabolic models. Metabolic engineering, 6(4), 285-293.

[46] Oberhardt, M. A., Palsson, B. Ø., & Papin, J. A. (2009). Applications of genome-scale metabolic reconstructions. Molecular systems biology, 5(1), 320.

[47] Dias, O., Rocha, M., Ferreira, E. C., & Rocha, I. (2015, May). Metabolic models reconstruction using genome scale information (merlin): assessment and validation. In Copenhagen Biosciences Conferences-7th Conference: Cell Factories and Biosustainability (pp. 99-100).

[48] Dias, O., Rocha, M., Ferreira, E. C., & Rocha, I. (2018). Reconstructing High-Quality Large-Scale Metabolic Models with merlin. In Metabolic Network Reconstruction and Modeling (pp. 1-36). Humana Press, New York, NY.

[49] Alpaydin, E. (2009). Introduction to machine learning. MIT press.

[50] Murphy, K. P., Bach, F. (2012), Machine Learning: A Probabilistic Perspective. MIT Press

[51] Larranaga, P., Calvo, B., Santana, R., Bielza, C., Galdiano, J., Inza, I., ... & Robles, V. (2006). Machine learning in bioinformatics. Briefings in bioinformatics, 7(1), 86-112.

[52] Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. Emerging artificial intelligence applications in computer engineering, 160, 3-24.

[53] Dietterich, T. (1995). Overfitting and undercomputing in machine learning. ACM computing surveys (CSUR), 27(3), 326-327.

[54] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

[55] Nielsen, M. A. (2015). Neural networks and deep learning(Vol. 25). USA: Determination press.

[56] Bagley, S. C., White, H., & Golomb, B. A. (2001). Logistic regression in the medical literature:: Standards for use and reporting, with particular attention to one medical domain. Journal of clinical epidemiology, 54(10), 979-985.

[57] Hosmer Jr, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). Applied logistic regression (Vol. 398). John Wiley & Sons.

[58] Rodriguez-Galiano, V. F., Ghimire, B., Rogan, J., Chica-Olmo, M., & Rigol-Sanchez, J. P. (2012). An assessment of the effectiveness of a random forest classifier for land-cover classification. ISPRS Journal of Photogrammetry and Remote Sensing, 67, 93-104.

[59] Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.

[60] Bosch, A., Zisserman, A., & Munoz, X. (2007, October). Image classification using random forests and ferns. In Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on (pp. 1-8). IEEE.

[61] Díaz-Uriarte, R., & De Andres, S. A. (2006). Gene selection and classification of microarray data using random forest. BMC bioinformatics, 7(1), 3.

[62] Furey, T. S., Cristianini, N., Duffy, N., Bednarski, D. W., Schummer, M., & Haussler, D. (2000). Support vector machine classification and validation of cancer tissue samples using microarray expression data. Bioinformatics, 16(10), 906-914.

[63] Wang, L. (Ed.). (2005). Support vector machines: theory and applications (Vol. 177). Springer Science & Business Media.

[64] Joachims, T. (1998, April). Text categorization with support vector machines: Learning with many relevant features. In European conference on machine learning (pp. 137-142). Springer, Berlin, Heidelberg.

[65] Suykens, J. A., & Vandewalle, J. (1999). Least squares support vector machine classifiers. Neural processing letters, 9(3), 293-300.

[66] Tong, S., & Koller, D. (2001). Support vector machine active learning with applications to text classification. Journal of machine learning research, 2(Nov), 45-66.

[67] Jain, A. K., Mao, J., & Mohiuddin, K. M. (1996). Artificial neural networks: A tutorial. Computer, 29(3), 31-44.

[68] Khan, J., Wei, J. S., Ringner, M., Saal, L. H., Ladanyi, M., Westermann, F., ... & Meltzer, P. S. (2001). Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. Nature medicine, 7(6), 673.

[69] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural networks, 61, 85-117.

[70] Zhang, G., Patuwo, B. E., & Hu, M. Y. (1998). Forecasting with artificial neural networks:: The state of the art. International journal of forecasting, 14(1), 35-62.

[71] Patterson, J., & Gibson, A. (2017). Deep Learning: A Practitioner's Approach. " O'Reilly Media, Inc.".

[72] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436.

[73] Chu, C. T., Kim, S. K., Lin, Y. A., Yu, Y., Bradski, G., Olukotun, K., & Ng, A. Y. (2007). Map-reduce for machine learning on multicore. In Advances in neural information processing systems (pp. 281-288).

[74] Specht, D. F. (1990). Probabilistic neural networks. Neural networks, 3(1), 109-118.

[75] Basheer, I. A., & Hajmeer, M. (2000). Artificial neural networks: fundamentals, computing, design, and application. Journal of microbiological methods, 43(1), 3-31.

[76] Hall, M. A. (1999). Correlation-based feature selection for machine learning.

[77] Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. Journal of machine learning research, 3(Mar), 1157-1182.

[78] Valentini, G., & Masulli, F. (2002, May). Ensembles of learning machines. In Italian Workshop on Neural Nets (pp. 3-20). Springer, Berlin, Heidelberg.

[79] Dietterich, T. G. (2000, June). Ensemble methods in machine learning. In International workshop on multiple classifier systems (pp. 1-15). Springer, Berlin, Heidelberg.

[80] Liaw, A., & Wiener, M. (2002). Classification and regression by randomForest. R news, 2(3), 18-22.

[81] Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern recognition, 30(7), 1145-1159.

[82] Forman, G., & Scholz, M. (2010). Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. ACM SIGKDD Explorations Newsletter, 12(1), 49-57.

[83] Phien, H. N., & Kha, N. D. A. (2003). Flood forecasting for the upper reach of the Red River Basin, North Vietnam. Water SA, 29(3), 267-272.

[84] Varma, S., & Simon, R. (2006). Bias in error estimation when using cross-validation for model selection. BMC bioinformatics, 7(1), 91.

[85] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1). Cambridge: MIT press

[86] Szegedy, C., Toshev, A., & Erhan, D. (2013). Deep neural networks for object detection. In Advances in neural information processing systems (pp. 2553-2561).

[87] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A. R., Jaitly, N., ... & Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal processing magazine, 29(6), 82-97.

[88] Aggarwal, C. C. (2018). Neural networks and deep learning(pp. 978-3319944623). Berlin, Germany:: Springer.

[89] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Advances in neural information processing systems (pp. 3104-3112).

[90] Sutskever, I., Martens, J., & Hinton, G. E. (2011). Generating text with recurrent neural networks. In Proceedings of the 28th International Conference on Machine Learning (ICML-11) (pp. 1017-1024).

[91] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

[92] Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2009, June). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In Proceedings of the 26th annual international conference on machine learning (pp. 609-616). ACM.

[93] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016, November). Tensorflow: a system for large-scale machine learning. In OSDI (Vol. 16, pp. 265-283).

[94] Rampasek, L., & Goldenberg, A. (2016). Tensorflow: Biology's gateway to deep learning?. Cell systems, 2(1), 12-14.

[95] Chollet, F. (2017). Deep learning with python. Manning Publications Co..

[96] Alipanahi, B., Delong, A., Weirauch, M. T., & Frey, B. J. (2015). Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning. Nature biotechnology, 33(8), 831.

[97] Ghandi, M., Lee, D., Mohammad-Noori, M., & Beer, M. A. (2014). Enhanced regulatory sequence prediction using gapped k-mer features. PLoS computational biology, 10(7), e1003711.

[98] Jiménez, J., Skalic, M., Martínez-Rosell, G., & De Fabritiis, G. (2018). K DEEP: Protein–Ligand Absolute Binding Affinity Prediction via 3D-Convolutional Neural Networks. Journal of chemical information and modeling, 58(2), 287-296.

[99] Eickholt, J., & Cheng, J. (2013). DNdisorder: predicting protein disorder using boosting and deep networks. BMC bioinformatics, 14(1), 88.

[100] Wang, S., Weng, S., Ma, J., & Tang, Q. (2015). DeepCNF-D: predicting protein order/disorder regions by weighted deep convolutional neural fields. International journal of molecular sciences, 16(8), 17315-17330.

[101] Wang, S., Peng, J., Ma, J., & Xu, J. (2016). Protein secondary structure prediction using deep convolutional neural fields. Scientific reports, 6, 18962.

[102] Blum, T., Briesemeister, S., & Kohlbacher, O. (2009). MultiLoc2: integrating phylogeny and Gene Ontology terms improves subcellular protein localization prediction. BMC bioinformatics, 10(1), 274.

[103] Nugent, T., & Jones, D. T. (2009). Transmembrane protein topology prediction using support vector machines. BMC bioinformatics,

[104] Molinaro, A. M., Simon, R., & Pfeiffer, R. M. (2005). Prediction error estimation: a comparison of resampling methods. Bioinformatics, 21(15), 3301-3307.

[105] Hawkins, D. M. (2004). The problem of overfitting. Journal of chemical information and computer sciences, 44(1), 1-12.

[106] Schwartz, A. S., Hannum, G. J., Dwiel, Z. R., Smoot, M. E., Grant, A. R., Knight, J. M., ... & Liu, Y. (2018). Deep semantic protein representation for annotation, discovery, and engineering. BioRxiv, 365965.