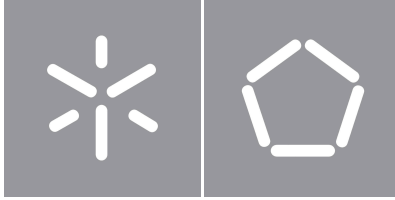


University of Minho
School of Engineering

Guilherme Lobo Sousa

Developing an automated machine and deep learning framework for protein classification



University of Minho
School of Engineering

Guilherme Lobo Sousa

Developing an automated machine and deep learning framework for protein classification

Masters' Dissertation
Master in Bioinformatics

Dissertation supervised by
Miguel Francisco Almeida Pereira da Rocha
Ricardo Nuno Correia Pereira

Copyright and Terms of Use for Third Party Work

This dissertation reports on academic work that can be used by third parties as long as the internationally accepted standards and good practices are respected concerning copyright and related rights.

This work can thereafter be used under the terms established in the license below.

Readers needing authorization conditions not provided for in the indicated licensing should contact the author through the RepositóriUM of the University of Minho.

License granted to users of this work:



CC BY-NC-ND

<https://creativecommons.org/licenses/by-nc-nd/4.0/>

Acknowledgements

Este trabalho não teria sido possível sem o apoio e orientação de várias pessoas, às quais expresso o meu sincero agradecimento.

Em primeiro lugar, gostaria de agradecer aos meus supervisores, Dr. Miguel Rocha e Dr. Ricardo Pereira, pela oportunidade e pelo valioso conhecimento transmitido ao longo deste percurso. Agradeço também ao Dr. Oscar Dias por me ter acolhido na OmniumAI, permitindo-me integrar a equipa e, na fase final do mestrado, reconhecer o meu esforço com a bolsa do IEFP.

Um agradecimento especial ao João e à Marta, cujo apoio incansável foi fundamental para que este trabalho chegasse a bom termo. Também não posso deixar de agradecer à restante equipa da OmniumAI, pela partilha de conhecimento e por terem tornado esta experiência extremamente gratificante, onde o ambiente de trabalho, muitas vezes, foi uma fonte de diversão.

Aos meus colegas de turma do mestrado em Bioinformática, agradeço pelos bons momentos partilhados ao longo destes dois anos.

Por fim, mas certamente não menos importante, agradeço à minha família e à minha namorada, que me apoiaram incondicionalmente ao longo deste segundo mestrado, sempre dando o máximo suporte e encorajando-me a abraçar este desafio.

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.

University of Minho, Braga, october 2024

Assinado por: **Guilherme Lobo e Sousa**
Num. de Identificação: 14458243
Data: 2024.10.28 11:14:17+00'00'



Guilherme Lobo Sousa

Abstract

Researchers are increasingly concerned with the challenge of providing nutritious, high-quality food to meet the growing global population. In this context, applying machine learning (ML) and deep learning (DL) models to classify and identify key protein properties, such as antioxidants and allergens, holds significant potential. This approach can improve food formulation, safety, and sustainability, contributing to healthier products that meet nutritional needs.

This project, in collaboration with OmniumAI, a spin-off from the University of Minho, focused on advancing the development of *OmniA*, an automated machine learning (AutoML) platform. In the *Proteins* sub-package, new protein descriptors were added to the feature extraction module, and three encoding methods were updated (Non-linear fisher, Evolutionary scale modeling, and Protbert) to handle sequences of up to 600 amino acids through padding and truncation. In the *Generics* sub-package, four DL models were implemented: Autoencoder and 1D convolutional neural networks (CNN1D), for tabular data; recurrent neural networks (RNNs), and CNN1D-RNNs for matrix data. Cross-validation support was added and both sub-packages now include presets at three complexity levels, which progressively increase the computational resources utilized.

Benchmarking was conducted on specific antioxidant and allergen datasets to validate *OmniA*'s development. In the antioxidant study, *OmniA*'s top pipeline (based on pre-trained ESM2 model, with a tabular predictor) achieved balanced results (accuracy and F1-scores around 0.7). While not the top-performing approaches, it provided strong reliability through robust methods that maintained the integrity of the test set. For the allergenic study, *OmniA*'s pipeline (similar configuration) outperformed other approaches with excellent results (accuracy and F1-score of 0.94).

In conclusion, *OmniA* strikes a balance between predictive capacity and computational efficiency across both case studies. The platform is flexible, automatic, and user-friendly, even for individuals without deep expertise, while remaining competitive with traditional approaches.

Keywords Automated machine learning, *OmniA*, proteins

Resumo

Os investigadores estão preocupados com o desafio de fornecer alimentos nutritivos e de alta qualidade para satisfazer a crescente população mundial. Neste contexto, a aplicação de modelos de aprendizagem de máquina e de aprendizagem profunda para classificar propriedades proteicas, como antioxidante e alergénica, tem um potencial significativo. Esta abordagem pode melhorar a formulação, segurança e sustentabilidade dos alimentos, contribuindo para produtos que satisfaçam as necessidades nutricionais.

Este projeto, em colaboração com a OmniumAI, uma spin-off da Universidade do Minho, focou-se no desenvolvimento da *OmniA*, uma plataforma de aprendizagem máquina automática. No sub-pacote de proteínas, foram adicionados novos descritores proteicos ao módulo de extração de características, e três métodos de codificação foram actualizados para lidar com sequências de até 600 aminoácidos através de preenchimento e truncagem. No sub-pacote de genéricos, foram implementados quatro modelos de aprendizagem profunda (Autoencoder, CNN1D para dados tabulares, RNN e CNN1D-RNN para dados matriciais), juntamente com suporte para validação cruzada. Ambos os sub-pacotes incluem agora predefinições em três níveis de complexidade: leve, médio e pesado.

Para validar o desenvolvimento, foi efectuada uma avaliação comparativa com comparativa em conjuntos de dados específicos de proteínas antioxidantes e alergénicas para validar o desenvolvimento da *OmniA*. No estudo de antioxidantes, a melhor pipeline da *OmniA* (baseada no modelo ESM2 pré-treinado, com um modelo tabular predictor) obteve resultados equilibrados (accuracy e F1-score de 0,7). Embora não seja a abordagem com melhor desempenho, proporcionou uma forte fiabilidade através de métodos robustos que mantiveram a integridade do conjunto de testes. Para o estudo em proteínas alergénicas, a pipeline do *OmniA* (configuração semelhante) superou as outras abordagens com excelentes resultados (accuracy e F1-score de 0,94).

Em suma, a *OmniA* consegue um equilíbrio entre a capacidade preditiva e eficiência computacional. A plataforma é flexível e automática, mantendo-se competitiva em relação às abordagens tradicionais.

Palavras-chave Aprendizagem máquina automatizada, *OmniA*, proteínas

Contents

1	Introduction	1
1.1	Context and motivation	1
1.2	Objectives	2
1.3	Thesis Organization	3
2	Fundamentals and Applied Approaches	4
2.1	Food challenges	4
2.2	Proteins	6
2.2.1	Protein sources	6
2.2.2	Amino acids residues	7
2.2.3	Protein structure	8
2.2.4	Protein function	9
2.2.5	Case studies	9
2.3	Machine and deep learning	10
2.3.1	Data collection and preparation	11
2.3.2	Unsupervised learning	12
2.3.3	Supervised learning	13
2.3.4	Deep learning	18
2.3.5	Feature Selection	26
2.3.6	Hyperparameter tuning	26
2.3.7	Model evaluation	27
2.4	Automated machine learning	30
2.5	Machine and deep learning applied to proteins	32
2.5.1	Protein descriptors	32
2.5.2	Sequence encoding	33

2.5.3	Relevant work	35
2.5.4	Relevant packages and tools	37
3	Methods	40
3.1	Data Collection	40
3.1.1	Antioxidant dataset	40
3.1.2	Allergenic dataset	41
3.2	<i>OmniA</i>	42
3.2.1	<i>Proteins</i> sub-package	44
3.2.2	<i>Generics</i> sub-package	46
3.2.3	Global pipeline	48
3.3	Propythia	49
4	OmniA development	51
4.1	Feature extraction	51
4.1.1	Grouped amino acid composition	52
4.1.2	Composition of K-spaced amino acid pairs	52
4.1.3	Conjount triad	52
4.1.4	Dipeptide deviation from expected mean	53
4.2	Encoding	54
4.2.1	NLF encoder	54
4.2.2	ESM-2 encoder	54
4.2.3	ProtBert	55
4.3	Presets	55
4.4	Deep learning models	57
4.4.1	AutoEncoder	58
4.4.2	CNN1DModel	59
4.4.3	RNNModel	60
4.4.4	ConvRNNModel	61
4.5	Global Pipeline	62
5	Results and discussion	63
5.1	Data analysis	63
5.1.1	Sequence length	63

5.1.2	Labels	65
5.1.3	Distribution of sequence length by label	65
5.2	<i>OmniA</i>	66
5.2.1	Antioxidant case study	67
5.2.2	Allergenic case study	69
5.3	Propytha	70
5.3.1	Antioxidant case study	71
5.3.2	Allergenic case study	73
5.4	Comparison with different studies	76
5.4.1	Antioxidant case study	76
5.4.2	Allergenic case study	78
6	Conclusions and future work	81
6.1	Conclusions	81
6.2	Prospect for future work	82

List of Figures

1	Projection of food demand and food production, in millions of tonnes, from 2020 to 2050.	4
2	Exploring alternative protein sources: A comprehensive overview of the different options, highlighting some benefits and challenges.	6
3	Tetrahedral arrangement of an amino acid residue. A common structure to almost all α -amino acids.	7
4	The four levels of protein structure: Primary (amino acid sequence), Secondary (uniform folding), Tertiary (3D structure), and Quaternary (association of multiple polypeptides) . .	8
5	Artificial intelligence and its subfields.	11
6	Pipeline of unsupervised learning.	12
7	Pipeline of supervised learning.	14
8	Optimal hyperplane for support vector machines, showing the separation between two classes: red squares and blue circles.	16
9	Scheme of an artificial neuron showing the process from input data to output, involving weights, bias, and an activation functions.	19
10	Representation of a convolutional neural network applied to a zebra image, showing three convolutional layers, ReLU activation, flattening, and dense layers for classification. . . .	20
11	Recurrent neural networks model's architecture (A) and its unfolded version (B).	22
12	The structure of the long short-term memory neural network.	23
13	The structure of the gated recurrent unit neural network.	24
14	A confusion matrix in blue and commom metrics in green.	28
15	Behavior of machine learning models: Underfitting (poor data capture), good fit (efficient generalization), and overfitting (too adapted to training data, poor generalization).	29
16	Key stages of a machine learning pipeline, focusing on feature selection, model selection, and hyperparameter optimization, all streamlined by automated machine learning. . . .	31

17	<i>OmniA</i> architecture and respective methods implemented in each subpackage.	43
18	The Proteins sub-package in <i>OmniA</i> and its implement modules.	44
19	Sequence of steps in the automated machine learning pipeline implemented in <i>OmniA</i> , emphasizing the role of presets in the <i>Proteins</i> and <i>Generics</i> sub-packages.	48
20	Architecture of the AutoEncoderMLP model, illustrating the main components: encoder, decoder, and Multi-layer Perceptron	58
21	Architecture of the CNN1DModel, showing the 1D convolutional neural network and dense layers used for processing tabular data.	59
22	Architecture of the RNNModel, showcasing recurrent neural network layers with options for LSTM, GRU, or simple recurrent networks, including a bidirectional configuration. . .	60
23	Architecture of the ConvRNNModel, combining convolutional and recurrent layers to ex- tract spatial and temporal features from matrix data.	61
24	Representation of the <i>OmniA</i> global pipeline using the light configuration preset.	62
25	Sequence length distribution histograms for (A) the antioxidant dataset and (B) the aller- gen dataset.	64
26	lass distribution for the antioxidant dataset and the allergen dataset.	65
27	Sequence length distribution for the antioxidant dataset and the allergen dataset, cate- gorized by class label.	66
28	Bar chart comparing the performance metrics of various approaches in the antioxidant case study.	77
29	Bar chart comparing the performance metrics of various approaches in the allergenic case study.	79

List of Tables

1	Common metrics for evaluating regression models, where y is the array of real outputs and $\hat{y}$ are the predicted outputs.	28
2	Main types of descriptors, as well as an example of each and its description.	32
3	Recent studies applying machine learning and deep learning methodologies to predict the antioxidant (blue) and allergenic (green) properties of proteins.	35
4	Machine learning models used in ProPythia alongside their respective hyperparameters, which were tuned using randomized search optimization.	49
5	Descriptor presets that generate a set of descriptors from protein sequences.	55
6	Overview of preset configurations for protein sequence analysis, detailing the complexity levels and the respective choices available for standardization and feature extraction. . .	56
7	Default hyperparameters of the developed deep learning models	57
8	Validation metrics for the top 4 pipelines selected for the antioxidant case study.	68
9	Validation metrics for the top 4 pipelines selected for the allergenic case study.	70
10	Optimized hyperparameters and performance metrics for each model and type of data representation in the antioxidant case study.	71
11	Optimized hyperparameters and performance metrics for each model and type of data representation in the allergenic case study.	74

Acronyms

Adam Adaptive Moment Estimation

AE Autoencoder

AI Artificial Intelligence

ANN Artificial Neural Networks

ANOVA Analysis of Variance

API Application Programming Interface

ASPs Allergen-Specific Patterns

AutoML Automated machine learning

BLOSUM Block Substitution Matrix

CART Classification And Regression Trees

CKSAAP Composition of k-Spaced Amino Acid Pairs

CNN Convolutional Neural Networks

COMPARE Comprehensive Protein Allergen Resource

CT Conjoint Triad

DDE Dipeptide Deviation from Expected Mean

DL Deep Learning

DNA Deoxyribonucleic Acid

DNN Deep Neural Networks

ESM Evolutionary Scale Modeling

FAO Food and Agriculture Organization

GAAC Grouped Amino Acid Composition

GPU Graphics Processing Unit

GRU Gated Recurrent Unit

ID3 Iterative Dichotomiser 3

IUIS International Union of Immunological Societies

IoT Internet of Things

KNN K-Nearest Neighbor

LASSO Least Absolute Shrinkage and Selection Operator

LSTM Long Short-Term Memory

NB Naïve bayes

MCC Matthews Correlation Coefficient

MLM Masked Language Modelling

MAE Mean Absolute Error

MSE Mean Square Error

MRMR Maximum Relevance Minimum Redundancy

ML Machine Learning

MCC Matthews Correlation Coefficient

MLP Multi-Layer Perceptron

NLP Natural Language Processing

NN Neural Network

NNI Neural Network Intelligence

NLF Non-Linear Fisher

PCA Principal Component Analysis

PCs Principal Components

PDB Protein Data Bank

PR Precision Recall

PSI-BLAST Position-Specific Iterative Basic Local Alignment Search Tool

PSSM Position-Specific Scoring Matrices

PSFM Position-Specific Frequency Matrix

R^2 Coefficient of Determination

ReLU Rectified Linear Unit

ROC Receiver Operating Characteristic

RNA Ribonucleic Acid

scRNA-seq single-cell RNA sequence

SDAP Structural Database of Allergenic Proteins

RNA-seq RNA sequence

RMSE Root Mean Square Error

RMSprop Root Mean Square Propagation

RNN Recurrent Neural Networks

SGD Stochastic Gradient Descent

SNE Stochastic Neighbor Embedding

SVM Support Vector Machines

t-SNE t-Distributed Stochastic Neighbor Embedding

Tanh Tangent Hyperbolic

TPE Tree-structured Parzen Estimator

UniProt Universal Protein Resource

WHO World Health Organization

Chapter 1

Introduction

This chapter covers the context and motivation for carrying out this project, including a description of the main and intermediate objectives that will contribute to the completion of everything that has been suggested and discussed with the company OmniumAI.

1.1 Context and motivation

In modern society's complex challenges, Artificial Intelligence (AI) is emerging as a vital discipline within computer science. Within this vast field, Machine Learning (ML) stands out, a specialised domain whose purpose is to use models capable of offering accurate predictions for unseen data . The generalised ML pipeline, which ranges from pre-processing raw data to choosing suitable algorithms for training models and evaluating their performance, demonstrates its versatility and usefulness in various areas. A crucial subdivision within the ML field is Deep Learning (DL), which uses Neural Networks (NNs) as its backbone. Especially effective for tackling highly complex problems, intricate feature engineering requirements and large datasets, DL stands out as a powerful tool [1].

New omics technologies, especially in areas such as proteomics, generate large-scale datasets and, as such, ML and DL algorithms have emerged as indispensable tools [2]. The application of suitable algorithms and relevant descriptors makes it possible to efficiently process this type of data, providing robust results in a scenario where manual processing would be impractical and prone to inaccuracies [3]. However, despite the computational limitations, the real complexity in implementing these algorithms lies in carefully analysing each stage of the pipeline [1].

On the other hand, nowadays, a fundamental issue is causing growing concern among the world's leading organisations: the ability to provide nutritious and quality food to keep up with the increase in the global population [4]. In this sense, the application of ML and DL to classify and identify specific protein properties is extremely important, as it can influence food formulation, concerns related to food safety and

sustainability, and the creation of healthier food products suited to the nutritional needs of the population [5, 6]. In this context, the development of personalized foods and those based on alternative proteins present viable solutions to address the challenges posed by current demographic growth [5].

OmniA, developed by OmniumAI, is an advanced AI platform that uses precise ML and DL models to analyse a variety of data. One of *OmniA* main highlights is its ability to automate the creation of models using an Automated machine learning (AutoML) approach, making it a "hands-off" tool that can be used even by users with little prior knowledge. Thus, the continued development of this platform is highly justified, allowing it to be used to analyze and classify proteins with specific properties in the food industry, such as antioxidants and allergens. This expansion of *OmniA*'s capacity will contribute significantly to advances in food formulation, driving the creation of safer, high-quality products [7].

1.2 Objectives

The main objective of this work is to develop an automated ML and DL framework based on Python, covering back-end development to solve protein classification challenges based on their amino acid sequences, extending and validating existing software developed at OmniumAI. More specifically, the work will address the following scientific/technological objectives:

- A literature review on the relevant topics and exploring potential applications of ML and DL models in a related context;
- Exploring different datasets for protein properties, from which a few will be chosen to test the package's ability to deal with binary, multiclass and regression problems;
- Explore *OmniA*, and possibly other competitive frameworks, for protein sequence representation and classification, assessing their performance in different case studies, including antioxidant and allergenic classification;
- Improving *OmniA*'s back-end, namely in what pertains to the module that handles protein information, which will include code optimization, bug fixing, the automation of some pipeline steps, and the possible addition of new models, algorithms and general ML and DL features;
- Validate the previous application in different scenarios, with a focus on the ones provided by the case studies put forward above;
- Writing the thesis and, possibly, scientific publications with the main results of this work.

1.3 Thesis Organization

The thesis is organized in six chapters, a brief description of the chapters is given as follows:

- Chapter 1: Introduces the motivation and main goals of this work, providing a brief overview of the research context.
- Chapter 2: Provides a bibliographical review on topics essential to this work, addressing the challenges that motivate the study of proteins and their properties. It explores the biological context of protein and delves into the foundations of ML and DL. Additionally, discusses the application of these techniques in predicting specific protein properties, such as antioxidant and allergenic characteristics, along with relevant software tools used throughout the study.
- Chapter 3: Describes the methods used, with emphasis on the architecture of the *OmniA* platform and the Propythia framework.
- Chapter 4: Details the enhancements made to *OmniA*, particularly the development of modules within the Proteins and Generics sub-packages.
- Chapter 5: Presents the results and discussion, highlighting the characteristics of the datasets, the performance of the enhanced AutoML platform integrated into *OmniA*, and a comparative analysis with other methods reported in the literature.
- Chapter 6: Concludes the dissertation with key aspects and suggestions for future work.

Chapter 2

Fundamentals and Applied Approaches

This chapter provides a bibliographical review on the topics essential to the development of this work. The challenges that justify research into this topic will be addressed, with a focus on proteins, which represent the biological aspect of the investigation. ML and DL techniques, which form the computational basis of this study, will also be explored, as well as the applicability of these techniques in predicting protein properties of interest, namely antioxidant and allergenic properties.

2.1 Food challenges

By 2050, the world's population is expected to reach 9.7 billion people, representing an increase of around 2.3 billion compared to today [8]. This continued growth presents significant challenges for guaranteeing universal access to nutritious and healthy food. Population growth is directly associated with increasing demand, as shown in Figure 1, for products such as meat, fruits, vegetables, and cereals, putting considerable pressure on natural resources [9].

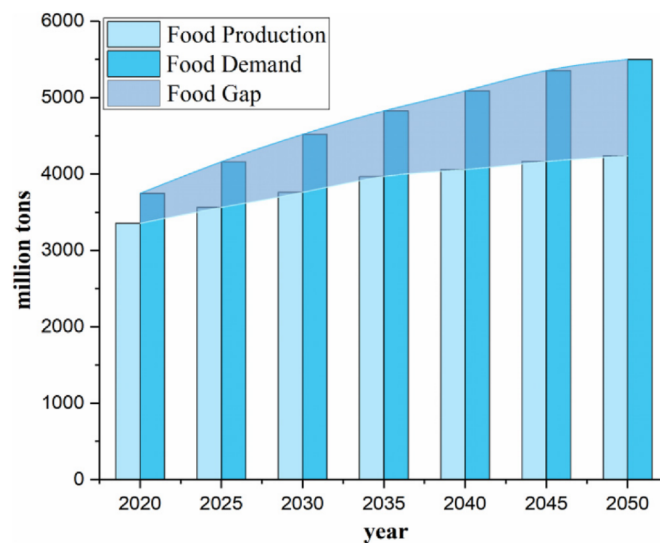


Figure 1: Projection of food demand and food production, in millions of tonnes, from 2020 to 2050 [9].

Faced with this challenge, there is a pressing need to adopt the 'sustainable food system' [10]. This new concept aims to guarantee everyone's food security and nutritional value, while respecting economic, and environmental standards [11]. The fundamental objective is to ensure that the well-being of future generations is not compromised by promoting a sustainable approach to food production, distribution, and consumption [10].

In response, the advent of Industry 4.0, which incorporates concepts such as Big Data, Cyber Security, Cloud Computing, Internet of Things (IoT), Advanced Robotics, AI, Digital Twins and Blockchain, stands out as a beacon of hope. It revolutionizes food production and safety processes sustainably and opens doors to practical applications in research into customized functional foods, providing unprecedented opportunities to create innovative products that benefit human health [12].

Recently, the research and application of functional foods have experienced a remarkable transformation, driven by the incessant search for new sources of bioactive molecules [13, 14]. Proteins have been the focus of special attention in this scenario, with efforts aimed not only at optimizing production processes, but also at personalizing the nutritional offer. Industry 4.0 plays a crucial role in this panorama, enabling not only improved efficiency in the extraction of these molecules from different sources, but also a dynamic response to the specific demands of developing functional foods [13].

Food traceability technology is a fundamental pillar of this innovation [15]. By integrating advanced systems, the industry ensures safety and transparency in protein production. Consumers can now trace the origin, production methods, and even quality tests associated with the proteins present in the functional foods they consume. This synergy between cutting-edge research and Industry 4.0 not only guarantees quality, but also strengthens consumer confidence in the origin and beneficial properties of the proteins they eat [13, 16]. By integrating Food Industry 4.0 innovations into protein research and application, we are not only driving efficiency, but also shaping the next generation of functional foods in an adaptable, connected, and transparent way [13, 15].

Although functional and fortified foods offer notable advantages, their application faces substantial challenges that transcend the sphere of nutritional composition. The degradation and loss of functionality associated with the instability of bioactive compounds, in particular the stability of proteins, not only compromise the sensory properties of food products, but also introduce a complicated dilemma in the search for effective and sustainable production of these innovative foods [13, 17]. It is therefore imperative to implement innovative strategies that also take into account the highest quality standards and market demands, giving rise to a more nutritionally, healthy and easily marketed customized food [18, 13, 14].

2.2 Proteins

Proteins, as essential macromolecules with high nutritional value, are present in all living organisms, playing an irreplaceable role in a variety of biochemical processes crucial to cellular structure and functioning [19]. Their intricate structure and functional diversity give them the status of 'molecular machines' in the context of the biological world, being the subject of research since the 19th century, and thus placing them at the heart of biological complexity [19, 20].

2.2.1 Protein sources

The growing demand for protein raises serious concerns, especially given the predominance of the current source, which is animal-based [21, 22, 23]. This type of source is associated with considerable amounts of resources and has significant impacts on environmental sustainability. Given this scenario, it becomes imperative to direct efforts toward the development of new protein sources capable of addressing the imminent shortage of this vital nutrient [21, 22]. Figure 2 provides a comprehensive overview of the main protein sources discussed in the literature, highlighting the positive aspects and the challenges associated with each category of protein source.

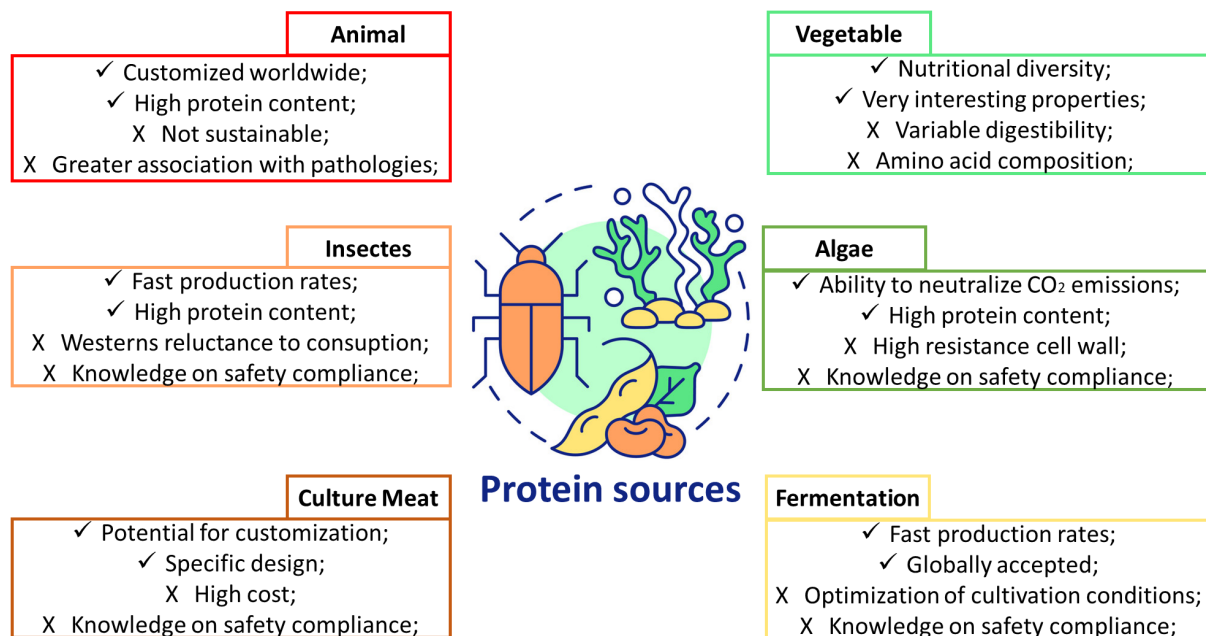


Figure 2: Exploring alternative protein sources: A comprehensive overview of the different options, highlighting some benefits and challenges. Adapted from [21, 22].

Analyzing recent data and considering the technological innovations underway, it is clear that the alternative protein market is constantly advancing [21, 22]. However, the sector faces several challenges that demand attention. Price parity, achieving competitive flavors and textures, consumer acceptance, lack of functional and technological properties (e.g., solubility, emulsion, gelation, and foaming), as well as access, security, and distribution issues, emerge as crucial constraints to overcome if the alternative protein sector is to achieve a more significant market share globally [21, 22]. In this way, proteins of alternative origin are becoming a real revolution in food, presenting clear benefits for industry and consumers [21].

2.2.2 Amino acids residues

Proteins come from a Deoxyribonucleic Acid (DNA) molecule, which is transcribed into an Ribonucleic Acid (RNA) molecule and then translated in ribosomes, resulting in a linear sequence of amino acids connected by covalent bonds, forming a polypeptide [19, 24]. These monomeric molecules have a general structure, illustrated in Figure 3, with a central alpha carbon covalently linked to a carboxyl group, an amino group, a hydrogen atom, and a side chain (R group) that varies for each amino acid [19].

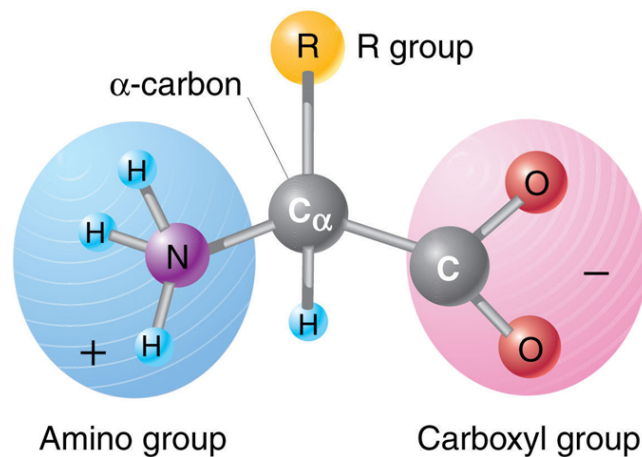


Figure 3: Tetrahedral arrangement of an amino acid residue. A common structure to almost all α -amino acids. Adapted from [19].

There are 20 common amino acids designated as standard amino acids [19, 25]. Diversity in the side chain (R group) gives each amino acid unique properties, such as size, electrical charge, polarity, molecular weight, and structure, playing a crucial role in the biochemical processes associated with these molecules [19, 20].

To simplify the identification of amino acids and reduce the size of files containing protein sequences, coding systems have been adopted [19]. These systems generally use abbreviations of the real names of each amino acid or just consider symbols.

2.2.3 Protein structure

Proteins are often described in four structural levels, each marked by increasing complexity: primary, secondary, tertiary, and quaternary [19, 20, 26]. This hierarchy reveals the architecture and functionality of proteins, offering an engaging analogy with the construction of a house, as shown in Figure 4 [20].

In the primary structure, the linear sequence of covalently linked amino acids, from the N to the C-terminal, acts as the building blocks, outlining the backbone of the protein [20, 24, 25]. The secondary structure adds layers of complexity, with the peptide chain forming stable patterns such as α -helices, β -sheets, and non-regular coils - comparable to the walls of a house [20, 24]. In tertiary structure, the protein folds three-dimensionally, creating a unique shape stabilized by more subtle interactions, such as Van der Waals bonds [19, 27]. This phase is equivalent to the architecture of the rooms and corridors [20]. In the quaternary structure, the final architecture of the house is completed, where two or more polypeptides organize themselves three-dimensionally, folding independently to form complexes called oligomers [19, 20]. These oligomers can be homooligomers, composed of identical subunits, or heterooligomers, with a mixture of different tertiary structures.

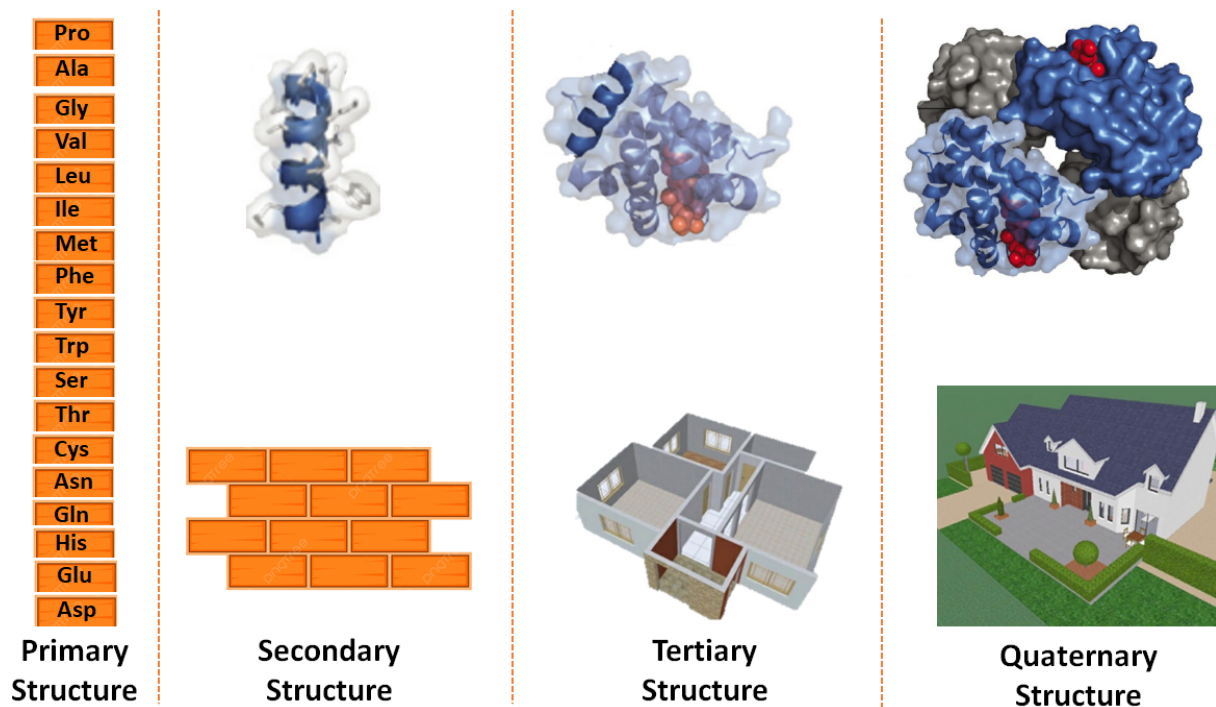


Figure 4: The four levels of protein structure. The building blocks, the amino acid sequence forms the primary structure; The assembly of the building blocks into a uniform formation constitutes the secondary structure; The assembly of the secondary structures into a further dimension forms the tertiary structure; The quaternary structure consists of the spatial association of all polypeptides. Adapted from [19, 20].

This analogy between protein architecture and the construction of a house not only simplifies understanding, but also highlights the beauty and functionality present at each structural level. Just as a house requires harmony between its parts, each structural level contributes to the functional integrity of proteins, revealing a true spectacle of molecular bioengineering [20].

2.2.4 Protein function

Proteins perform crucial functions, such as building and maintaining tissues, transporting key substances, supporting the immune system, hormonal regulation, acting as catalysts in biochemical reactions, cellular communication, storage, and controlling the cell cycle, among others [19, 28].

The diversity of protein functions opens the door to countless applications, including in the food industry [27, 28]. Understanding these functions and their relationship with specific protein properties makes this sector attractive, enabling the creation of products that sustainably promote health and well-being. Innovative approaches, such as computational biology, have been employed to explore the protein properties of diverse sources, providing valuable alternatives in a constantly growing food market [29, 30].

It is crucial to recognize that the chemical properties of proteins are shaped by their interactions with major food constituents, such as water, other proteins, lipids, and carbohydrates, as well as minor constituents such as salts, metal ions, and phenolic compounds [19, 25, 27]. Therefore, exploring protein properties alone is not enough. Key characteristics, such as surface properties of proteins, foaming capacity, and creating emulsions, play a crucial role and must be taken into account [27], especially in processed products, where texture, appearance, and stability are critical factors in satisfying consumer demands [25, 27].

2.2.5 Case studies

Antioxidant proteins

Antioxidant properties have come under increasing researches due to their crucial role in minimising and controlling the formation of free radicals [31]. These highly reactive compounds are associated with food spoilage and the development of pathologies such as diabetes, alzheimer's, cancer, among others [32].

Xu et. al, highlighted six amino acids (tryptophan, methionine, histidine, lysine, cysteine, arginine, and tyrosine) with remarkable antioxidant capacity [33]. However, it is important to emphasize that analyzing the protein structure may offer more comprehensive insights, suggesting that the linkage and structural arrangement between amino acids significantly influences the antioxidant activity of peptides [31].

In summary, it must be recognized that the relationship between the structure and antioxidant activity of peptides is not yet completely established. There are significant gaps in understanding the structure of antioxidant peptides from different protein sources, highlighting the critical need for bioinformatics tools to overcome this challenge [31, 32]. However, it is undeniable that foods enriched with antioxidant proteins would be of added value to both the food industry and the well-being of the population [32, 34].

Allergenics proteins

Allergenic properties have been the subject of study for several decades, since, according to the World Health Organization (WHO), food allergies are among the 5 most important public health concerns [35, 36]. A food allergy is an adverse immune response to a normally tolerated food protein, the clinical manifestations of which can range from minor digestive disturbances to life-threatening situations [36].

According to Food and Agriculture Organization (FAO), more than 90% of food allergens are proteins and are present in eight types of food: Fish, eggs, milk, crustaceans, soybeans, peanuts, nuts, and wheat [37]. However, it is necessary to analyze and identify a characteristic pattern for this type of protein property. Laboratory methods, mostly impregnated, although highly effective, are very expensive and time-consuming, and there is a need to apply bioinformatics methods to speed up the detection of allergenic proteins [38, 39]. Thus, bioinformatics approaches play an important role in identifying what the literature refers to as Allergen-Specific Patterns (ASPs) [39].

Ongoing research in the area of allergenic proteins aims, in parallel, to identify and promote the development of safe and effective substitute molecules for those associated with allergic reactions. This approach not only contributes to food safety, but also paves the way for alternatives that meet the needs of consumers with dietary restrictions, promoting a more inclusive approach in the food industry [38, 39].

2.3 Machine and deep learning

In the 1950s, AI emerged as a groundbreaking innovation, raising expectations for technological transformation. As a sub-field of AI, ML has significantly evolved, particularly since the 1990s, enabling systems to learn from data, identify patterns, and make decisions without explicit programming [40, 41]. This subfield also covers DL, which will be discussed in greater detail in subsection 2.3.4. The interrelationship between AI, ML and DL is illustrated in Figure 5 [42].

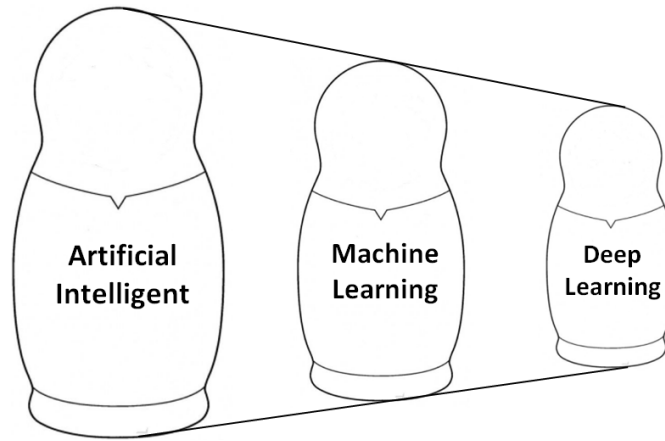


Figure 5: AI and its subfields. Adapted from [42].

ML methods can be categorized into four distinct types: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. In supervised learning, models learn from labeled data, while in unsupervised learning the data is unlabelled. Semi-supervised learning uses datasets that combine labeled samples with a larger number of unlabelled samples, overcoming the limitation of fully labeled datasets [43, 44, 45]. By incorporating unlabelled samples, which are more easily accessible, the approach aims to enrich the information about the problem, resulting in more robust models [45]. In reinforcement learning, an agent interacts with an environment, seeking to achieve a goal linked to the state of that environment [43, 46]. The agent, without prior knowledge of the best actions, learns a policy by executing actions, observing the rewards and subsequent states, to maximize the rewards over time.

In the context of ML, data quality is fundamental to building robust and effective models [1]. Before diving into the specifics of supervised and unsupervised learning, it is essential to first address a critical aspect of the ML process: data collection and preparation.

2.3.1 Data collection and preparation

The dataset usually takes the form of a matrix structure, composed of samples (rows) collected from observations and their extracted features (columns) [1]. These features cover a variety of information, from physical data to specific biochemical properties. Each characteristic can be classified as continuous (a subset of real numbers that can take on any value within a range), nominal categorical (samples grouped into unordered categories), or ordinal categorical (samples grouped into ordered categories) [47].

The dataset may include undesirable attributes, such as missing values and outliers, requiring pre-processing (data cleaning) [47, 48]. In addition to these challenges, data normalization may be necessary to ensure that features are on similar scales, especially when dealing with scale-sensitive algorithms [48].

2.3.2 Unsupervised learning

Unsupervised learning, whose pipeline is represented in Figure 6, consists of algorithms designed to analyze, detect, and provide valuable insights into data [1, 49]. The characteristic dataset for this type of learning has N inputs $X(i)$, $i = 1, \dots, N$, without any associated output (label), that is, no labeled data. Thus, this approach is considered an exploratory analysis technique that infers knowledge, mainly using clustering and/or dimensionality reduction methods on each independent variable [49].

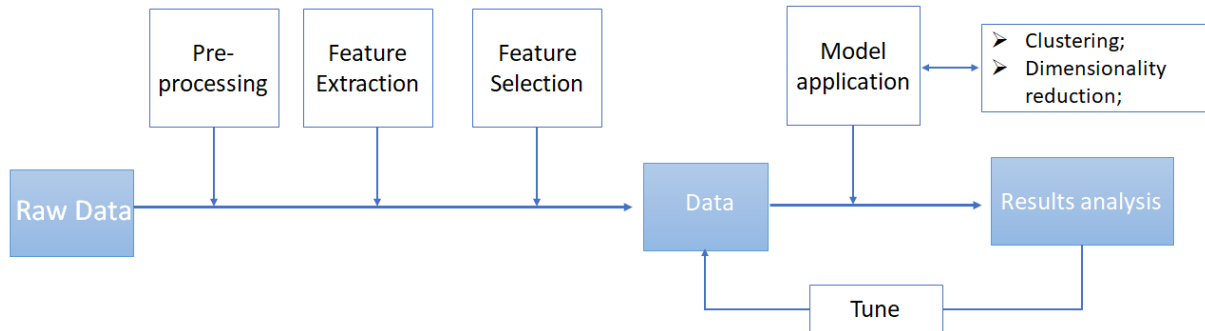


Figure 6: Pipeline of unsupervised learning. Adapted from [50].

Clustering

Clustering aims to identify subgroups (clusters), that best separate the data [51]. Essentially, it groups similar data in the same cluster, while distinct data is placed in separate clusters. From this partitioning, it is possible to associate a particular example with a specific group. Clustering methods, such as k-means and hierarchical, differ in their definition of similarity, as well as in the presentation of the results [51, 52].

A hierarchical cluster, visually represented in a dendrogram, can be constructed in two ways: agglomerative, where samples are initially assigned to individual clusters and then merged sequentially; or divisive, where all the data starts in a general cluster and it is separated consecutively [52]. Both methods use different approaches such as single, complete, or average linkage, as well as Euclidean or Manhattan distances to define the similarity between clusters.

The linkage method determines how the similarity or dissimilarity between two clusters is measured. Single linkage calculates the smallest distance between any pair of points from the two clusters, complete linkage considers the largest distance and average linkage computes the average of all pairwise distances. The choice of linkage method is data-dependent and significantly influences the structure of the resulting, thereby shaping the interpretation of clustering results [52].

In K-means, the parameter k represents the desired number of clusters. Initially, the algorithm randomly populates the space of observations with k centroids, which are the initial centers for each cluster. In each iteration, each observation is assigned to the nearest centroid, and then the centroid values are recalculated as the average of the assigned observations. This process is an optimization problem, seeking the minimum value for the objective function in each iteration. The model continues to iterate until it reaches a predetermined number of iterations or until there are no more significant improvements [51, 53].

Dimension reduction

Dimension reduction is a process that involves reducing the number of features, while preserving most of the information contained in the data [49]. This technique makes it possible to represent the dataset in a space with reduced dimensions, simplifying data visualization and requiring less computing power.

A notable example of this type of procedure is Principal Component Analysis (PCA). PCA starts by calculating new characteristics called Principal Components (PCs) [54, 55]. These PCs are orientated in a new coordinate space, following the decreasing order of the variance explained in the data. In this way, the data is projected onto a more compact and representative coordinate system of new latent variables.

Another approach is t-Distributed Stochastic Neighbor Embedding (t-SNE), a variation on an older technique called Stochastic Neighbor Embedding (SNE). Widely used for data visualization, t-SNE can also be applied to dimensionality reduction [56, 57]. This method focuses on preserving the local structure of the data, unlike PCA, which preserves the global structure and supports non-linear relationships between the new latent variables and the original ones, unlike PCA which is based on linear transformations [57].

2.3.3 Supervised learning

The main objective of a Supervised learning algorithm, whose general pipeline is represented in Figure 7, is to infer precise relationships between independent variables (x) and dependent variables (y) after a determined number of iterations [58].

However, before feeding data to the model, this type of learning requires an additional step known as data split [47]. The dataset is divided into a training and a test set. Training data is used to train the model, while test data is only used to evaluate the performance of the trained models [1, 49]. In many cases, a validation set may also be necessary to evaluate different model parameters [49].

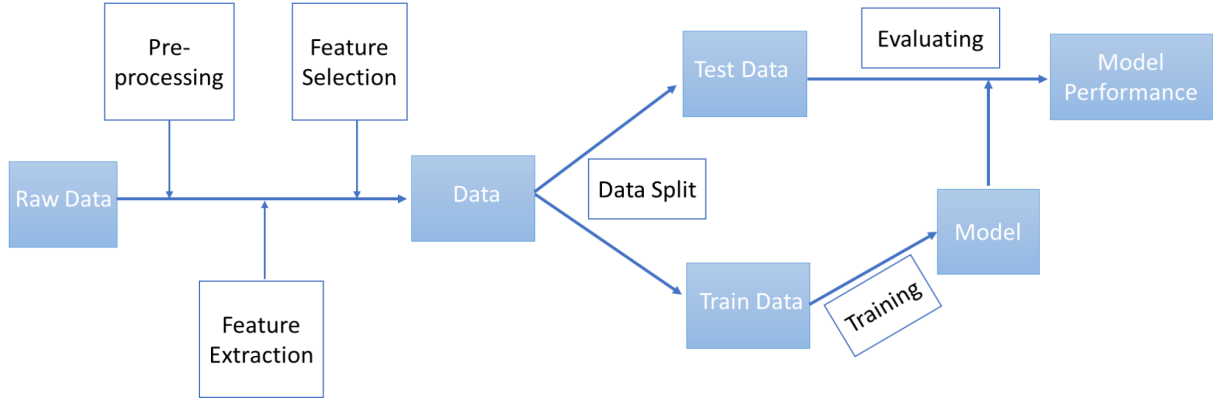


Figure 7: Pipeline of supervised learning. Adapted from [50].

This type of learning allows two types of variables to be predicted: continuous and/or discrete [1, 49]. When the output variable is continuous, it is a regression problem, where an output value is calculated (e.g. the biological activity of a protein). When it is discrete, it is called a classification problem (e.g. identification of protein subgroups), which can be binary or multiclass. Given this, supervised learning models share the aforementioned objective but adopt different approaches [1]. Below, some examples of these models are listed to better illustrate how they can be applied to achieve your specific purposes.

Linear models

Linear models can be used to predict continuous and categorical outcomes. Examples of linear models include Linear, Logistic, Ridge regression, and Least Absolute Shrinkage and Selection Operator (LASSO). These models differ in the way the parameters are calculated and how the complexity of the model is managed [49].

The Linear regression model seeks to establish a linear relationship between multiple independent features, presented as a vector of input values, and a numerical output variable [59]. Equation (1) represents the formula for a Linear regression model:

$$\hat{y} = \theta_0 + \sum_{j=1}^p x_j \theta_j \quad (1)$$

where $\hat{y}$ is the numerical output variable, obtained from an input vector X , p is the number of values in the input vector (number of features), θ is a vector with the values of the model parameters and θ_0 is called bias. In short, the key aspect of this Linear regression model is to find the optimum values of θ to minimize a given cost function (e.g. mean squared error) between the predicted and actual values.

Logistic regression is an algorithm widely used in classification problems [60]. This model is based on a sigmoidal function, which makes it possible to calculate values between 0 (indicating a negative

label) and 1 (representing a positive label) [58, 60]. In binary classification, the threshold is usually set at 0.5. The model parameters are optimized by estimating the maximum likelihood to obtain the maximum probability for each discrete value [60]. This process aims to adjust the function to maximize the probability of the labels observed in the data, giving the model improved predictive capacity.

Ridge regression and LASSO are regularization methods designed to address overfitting in Linear or Logistic regression models, differing in their approach to penalizing θ parameters [49]. In Ridge, penalization keeps θ close to zero, allowing each feature to have a minimal impact. On the contrary, LASSO, by permitting $\theta = 0$, distinguishes between dispensable and essential features. It is important to note that the difference arises from the type of penalization each imposes: LASSO penalizes parameters by their absolute values, while Ridge penalizes them by their squared values. The combination of these approaches results in Elastic Net, providing a balance between LASSO's selective capacity and Ridge's smoothing, offering flexibility in managing the model's complexity [61].

K-Nearest neighbor

The K-Nearest Neighbor (KNN) algorithm is a non-parametric approach used in classification and regression problems. Being non-parametric, it is particularly useful when there is little knowledge about the distribution of the data. This instance-based method does not create a model to learn the training data, but instead stores the training instances and uses them exclusively to classify new examples [49, 62].

In short, the essence of the KNN algorithm lies in calculating the distance between the sample to be predicted and the k nearest neighbors in the search space. These nearest neighbors, determined by the Euclidean distance (or other distance metrics), are crucial references for classifying the new sample [62].

Naïve bayes

The Naïve bayes (NB) algorithm is part of a set of probabilistic models that use statistical approaches to analyze data. This particular algorithm is based on Bayes' theorem, which assumes independence between the characteristics of each sample, considering all features to be equally important [49].

This algorithm calculates the probability of an example belonging to each output class, determining the conditional probability of each attribute in the sample for that class. The final classification is based on the highest probability value among the available classes. Each probability is calculated by multiplying the class representation (resulting from dividing the number of instances belonging to the class by the total number of samples in the dataset) by the product of the conditional probabilities of each attribute belonging to that class [49, 63].

Support vector machines

Support Vector Machines (SVM) models are versatile and can be used in both classification and regression problems, although their most common application is in binary classification [64]. In this context, the model aims to establish a decision boundary, known as a hyperplane, which maximizes the margin of separation between the two classes (Figure 8), that is, maximizes the distance between the support vectors [49, 64].

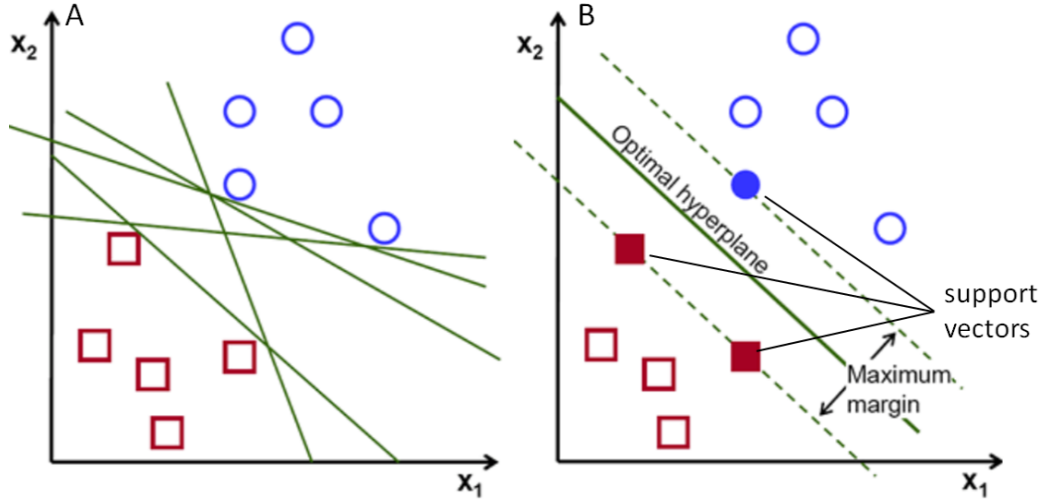


Figure 8: SVM ideal hyperplane. In situation A, it shows the process of optimizing an ideal hyperplane that maximizes the distance between support vectors. In situation B, this optimal hyperplane has already been found, guaranteeing the separation between two classes: red squares and blue circles. Adapted from [65].

Support vectors are the data points closest to the decision boundary and play a crucial role in determining the position and orientation of the hyperplane [64]. They are essential since changes in the position of data points that do not belong to the support vectors do not affect the location of the decision hyperplane, unlike changes in the support vectors themselves, which have a direct impact on the position and orientation of the hyperplane.

To create this hyperplane, it is necessary to choose a kernel that defines the essential mathematical equations for transforming the data [66]. If the data is linearly separable, a linear kernel is the appropriate choice. However, in most cases, where the separation is not linear, kernels such as sigmoid, polynomial, and radial basis function are useful for applying non-linear transformations to deal with complexity [49, 66].

It is important to note that each kernel has its hyperparameters that also influence the model performance [67]. Thus, this type of model, despite having a time-consuming training process, is widely described in the literature as highly accurate and capable of dealing with large volumes of data [64].

Decision trees

Decision trees are used to solve classification problems, following a model based on recursive division rules [68, 69]. The hierarchical structure of the tree is formed by nodes, where the parent nodes encompass the child nodes. Each node makes a decision based on a certain characteristic value, and the branches that originate from that node represent the different possible values (or ranges of values) for that characteristic.

The tree is traversed from top to bottom, reaching the terminal nodes (leaves), which determine the value of the output variable for the corresponding sample [68, 69, 70]. Node formation occurs by finding data division conditions that minimize variation within each separate group of samples [68]. To avoid overfitting, the addition of nodes is limited using specific measurements. This includes stopping group separation if there is no significant increase in subset purity, limiting the maximum tree depth, and removing leaf nodes to improve performance on test/validation data.

The widely adopted algorithm for building decision trees is Iterative Dichotomiser 3 (ID3), which follows a greedy top-down search approach without backtracking [69, 70]. It uses the concepts of entropy and information gain to guide the construction of the tree. Other algorithms, such as Classification And Regression Trees (CART), can also be applied, presenting variations in approach, splitting criteria, and impurity measurements [69].

Ensemble

Ensemble learning is an approach that integrates several models to form a new model that is more accurate than the individual models [49, 71]. This method is particularly effective when the models used are simple and have difficulty dealing with complex datasets. By combining the results of several models, ensemble learning can overcome individual limitations and offer more robust performance. There are several methods of ensemble learning, like voting, bagging, and boosting [71, 72].

There are two main types of methods used in voting models for class prediction: majority voting and weighted voting [71]. In majority voting, a pool is created with the classes predicted by each model, and the class chosen as the output from the pool is the one with the highest number of votes. In weighted voting, the procedure is similar, but each vote has a different "weight" in the voting process, depending on the model performance. Models with better predictive quality have a greater influence on the outcome of the vote compared to models with lower performance. For regression problems, two methods are generally adopted: simple average and weighted average. In the simple average method, the average of the results generated by the models is calculated. In the weighted average method, similar to weighted voting, different weights are applied to each model output value [49, 58, 71].

Bagging (Bootstrap Aggregating) is an ensemble learning technique [72]. In this method, several models are trained with different subsets of data obtained by sampling with replacement, ensuring that each model has equal weight in the final decision.

A notable example of bagging is Random Forest, which uses multiple decision trees [73]. Unlike traditional trees, Random Forest introduces more randomness by selecting different sets of features to train each tree, reducing the correlation between them. This gives the model robustness, as each tree is unique, capturing features with low correlation and improving overall accuracy.

Boosting is an algorithm designed to reduce bias and variance in ML models [71]. It achieves this by initially assigning equal weights to all data points. As the algorithm iterates, it dynamically adjusts these weights, placing more emphasis on samples that were misclassified. This iterative process encourages subsequent models to focus on capturing nuanced patterns that might have been overlooked initially. The final result is determined by aggregating the votes or averages of these iteratively refined models. Notable examples of boosting algorithms include AdaBoost, XGBoost, and gradient boosting [72].

2.3.4 Deep learning

The previously assessed models are categorized as forms of "shallow" learning, a term used to distinguish traditional machine learning models from deep learning algorithms. DL stands as a subfield within ML, showcasing remarkable achievements in recent decades. This approach emphasizes the sequential learning of layers, progressively evolving towards more insightful and meaningful representations [42].

DL represents a notable breakthrough by removing the necessity for manual feature design, a time-consuming and expertise-demanding task. This enables to directly engage with raw data, learning from it effectively, which is an advantageous edge over traditional ML methods. Such progress holds substantial implications, empowering DL to address more intricate challenges and extract valuable insights from the extensive information embedded in raw data [74].

Neural networks

A NN, also commonly referred to as Artificial Neural Networks (ANN), aims to simulate the functioning of the human brain, using neurons as interconnected building blocks to process the output based on the input data. In this process, the process of calculating the output (Z) of a neuron (summarised in Figure 9) results from multiplying the data in the neurons by the weights of the connections with other neurons, adding bias [75]. Subsequently, this result is subjected to the application of an activation function, chosen on the basis of different types of problems [42].

The most common activation functions include softmax, sigmoid, Tangent Hyperbolic (Tanh), and Rectified Linear Unit (ReLU). The softmax function is applied in multi-class classification problems, transforming real values into normalized probabilities, making it easier to interpret the output as the probability of belonging to each class. The sigmoid function is used in binary classification problems. It transforms values into an interval between 0 and 1, and is useful for modeling probabilities (given a certain threshold) of belonging to a positive or negative class. The Tanh function maps values to the interval between -1 and 1, making it suitable for hidden layers. The ReLU function is widely adopted in DL, improving computational efficiency. However, it is important to note that because the output is 0 for negative inputs, it can result in neurons entering a perpetually inactive state, known as "Dying ReLU" [42, 76].

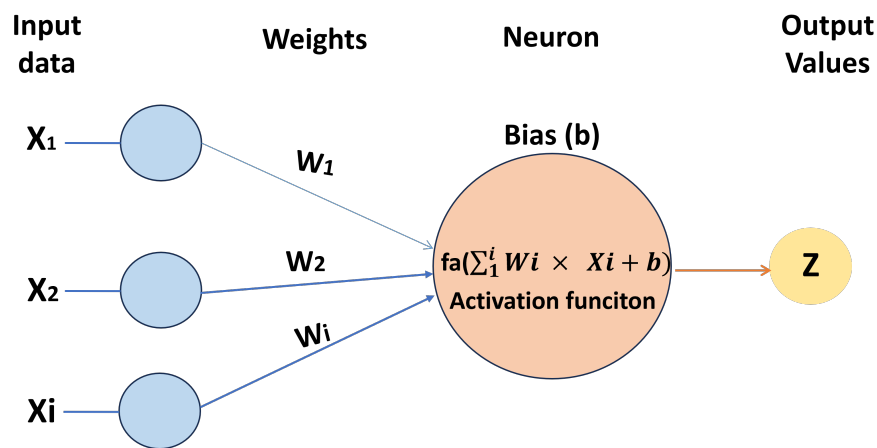


Figure 9: Scheme of an artificial neuron and the entire process of calculating the output from the input data. This process involves multiplying the input data in the neurons by the weights, followed by adding the bias. Subsequently, this result is subjected to the application of an activation function, giving rise to an output value, represented by "Z". Adapted from [75].

Training a NN involves continuously adjusting the learnable parameters (weights and bias), which are initially defined randomly. Successive comparisons between the model's predictions and the actual values are made using a loss function (also called an objective function). During forward propagation, the inputs are processed by the NN to generate an output, which is then compared to the actual values via the loss function. To improve the model's predictions, parameters are updated using backpropagation, which computes gradients of the loss function with respect to each parameter. This process enables the network to minimize loss and improve predictions over time. The choice of an optimizer, such as Adaptive Moment Estimation (Adam), Root Mean Square Propagation (RMSprop), or Stochastic Gradient Descent (SGD), is essential, as it adjusts the learning rate and update rules to efficiently navigate the parameter space and speed up convergence [42].

Optimizers are selected based on the specific characteristics of the dataset and the network architecture, aiming to balance between computational efficiency and model performance. This choice impacts training speed, stability, and the final quality of the trained NN model [42].

A Deep Neural Networks (DNN), also known as a dense neural network, has an additional level of complexity compared to a ANN. This complexity is associated with an increase in layers, resulting in an input layer, several hidden layers and an output layer. The input layer contains one neuron for each feature, while in the output layer the number of neurons is equal to the number of possible labels in a classification problem or to a single neuron in regression models [42].

Convolutional neural networks

Convolutional Neural Networks (CNN) are part of the feedforward NN group and are widely used in computer vision problems, image recognition and video [76, 77]. They generally consist of an input layer, followed by alternating convolutional and pooling layers. After this sequence, a flatten layer is used to transform the data structure into a one-dimensional vector. In the final stage, one or more dense layers are used to determine the output of the model, with all layers being connected, including the output layer. This architectural arrangement is illustrated in Figure 10 [78].

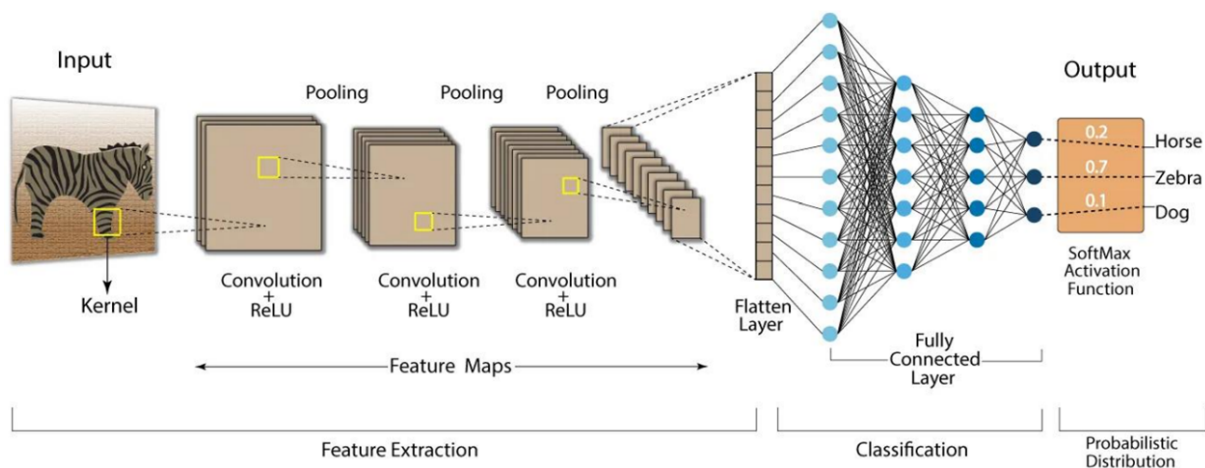


Figure 10: Representation of a CNN applied to an image of a zebra, illustrating the sequential flow through three convolutional layers with ReLU activation, followed by flattening, and concluding with dense layers for a classification problem [78].

Convolutional layers operate on feature maps with two spatial dimensions (height and width) and one depth dimension (colour channels in the input image or different types of features in the hidden layers) [42]. A new feature map is generated by multiplying the input feature map by a kernel/filter [42, 79]. In this process, convolution extracts patches from the feature map and applies the same transformation to all correspondences, resulting in a new feature map. The resulting feature map represents the presence of local patterns in the different blocks [42, 77].

Pooling layers are used to reduce the dimension of resource maps, reducing the number of coefficients and increasing efficiency [42, 79]. The most common strategies are average pooling, which calculates the average for the defined window, and max-pooling, which extracts the maximum value from the window. In comparison, max-pooling has been shown to lead to faster convergence, better generalization, and superior selection of invariant features [42].

In addition, other essential concepts influence the architecture and performance of CNN. Padding is a technique often used to preserve the spatial dimensions of the feature map during convolution by adding zeros around the input. On the other hand, stride refers to the number of positions the kernel moves through along the feature map. A larger stride reduces the spatial resolution of the feature map, speeding up the process, while a smaller stride captures more detailed information. The choice of appropriate values for padding and stride plays a critical role in the effectiveness of CNN, impacting the network's ability to learn and generalize patterns in complex data [42].

After several layers of convolution and pooling, one or more fully connected layers follow. At this point, the output is flattened to feed the dense layer as a 1D feature vector. This feature vector can be applied to classification tasks based on the extracted features, by fully connected layers, or used for further processing [42, 76].

Therefore, the main distinction between CNN and other DL architectures lies in the convolution and pooling operations. While dense layers identify global patterns, convolutional layers focus on local patterns, with subsequent layers learning more complex patterns based on previous detections [42].

Recurrent neural networks

In feedforward networks, communication between the layers is unidirectional, with each layer processing signals independently and forwarding the results sequentially. In Recurrent Neural Networks (RNN), this unidirectional communication is extended, allowing the network to maintain a memory of previous information when processing input sequences over time [76, 79].

In practice, a RNN can be understood as a type of NN equipped with an internal loop (Figure 11 A). This loop mechanism is crucial, as it gives the network the ability to process sequential data, allowing it to maintain a memory or state containing information from previous processing steps. In operational terms, the RNN takes a sequence of vectors as input and iterates over this sequence. The output production is the result of an activation function (f), which takes into account the current iteration (t) of the input with its respective weights (W_i), the current state of the model with its associated weights (W_s), and a bias (b). This output then becomes the new state at $t + 1$ (Figure 11 B) [42]. Consequently, each state, representing an output influenced by previous outputs, incorporates information related to all previous segments of the sequences, eliminating the need to keep records of all states [42, 79].

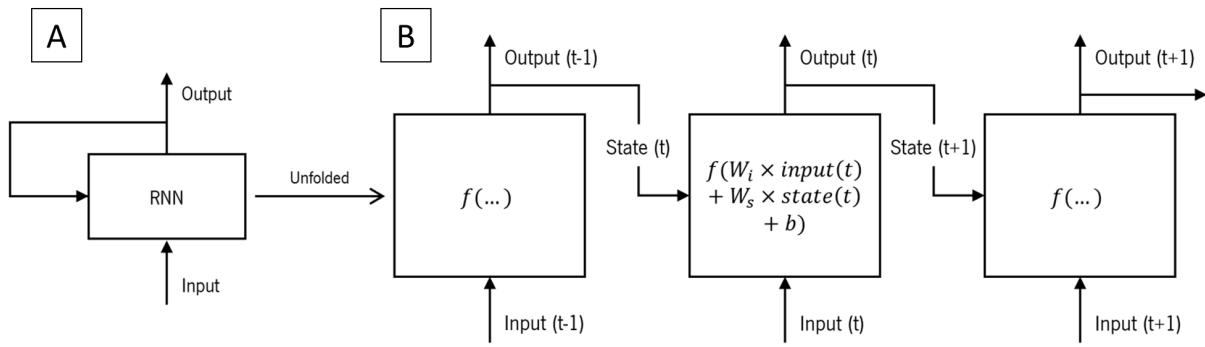


Figure 11: RNN model's architecture (A) and its unfolded version (B). Adapted from [42].

Hochreiter et al. point out the main limitation of RNN: the difficulty in effectively dealing with long-term sequences, known as the vanishing gradient problem. To overcome this limitation, improved variants such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) have emerged [80].

An LSTM is often made up of memory blocks consisting of a cell state and three gates: the input gate, the forget gate, and the output gate [79, 81]. The cell state, also known as the latent vector, is responsible for transferring information along the sequence chain. As the neural network processes the sequence, information is aggregated and manipulated by the cell state, allowing relevant information to be retained over time [81].

Figure 12 shows the structure of LSTM, where the memory block starts by adding the current input ($X(t)$) to the output of the previous memory block ($h(t - 1)$), both weighted by the corresponding weights. This value is then increased by a bias (b) and subjected to an activation function. The output of the Forget Gate is multiplied by the previous cell state ($C(t - 1)$). Two activation functions are applied to the Input Gate: Tanh and the sigmoid function. The result of the input gate is obtained by multiplying the resulting values of the activation functions. This result is added to the product of the multiplication between the previous cell state ($C(t - 1)$) and the output of the forget gate, thus generating the new cell

state ($C(t)$). With the new value, it is necessary to update the memory block that is still in $h(t-1)$. To do this, $C(t)$ is entered into the activation function Tanh and multiplied by the result of the sigmoid activation function, obtaining the value $h(t)$, which is the output of this unit LSTM, giving rise to the name output gate [79, 81, 82].

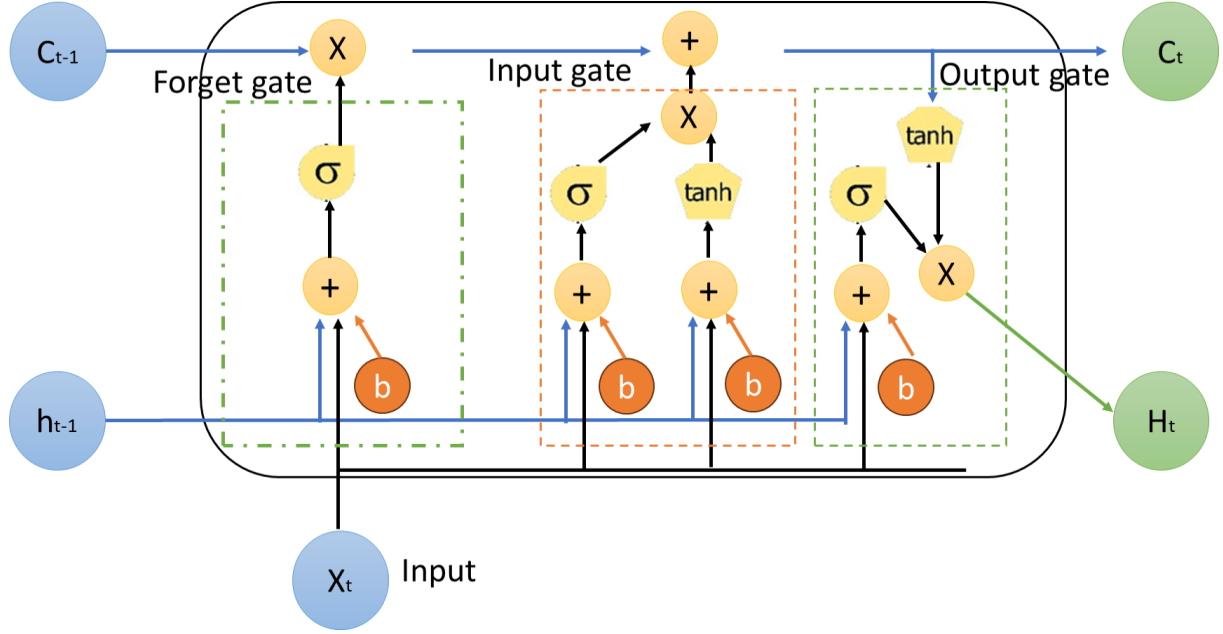


Figure 12: The structure of the LSTM neural network. Adapted from [83].

In short, the complexity of LSTM lies in the ability of the gates to make judicious decisions during sequential processing [81]. The input port decides what information from the input is relevant, the output port decides what information can be outputted based on the cell state, and the forget port can decide what information to discard from the cell state. LSTM to efficiently process data sequences and adapt to varied contexts, making it a powerful model [81, 82].

The GRU layers, first mentioned in 2014 by Cho et al. whose architecture is shown in Figure 17, have only two gates: the reset and update [84, 83, 85]. The reset gate determines the amount of information to be passed and performs the sigmoid function on the sum of the current input ($X(t)$) and the previous state ($h(t-1)$). The update gate performs a similar process, with the particularity of how the result is used. The output of this gate is used in the computation of both the current state ($h(t)$) and in the provisional state ($hp(t)$). The provisional state is calculated by applying the Tanh activation function on the sum of the current input ($X(t)$) and the result of the reset gate, multiplied by the result of the update gate. The current state is obtained by adding the provisional state to the product of the previous state ($h(t-1)$) and the inverted output ($1 - \text{output}$) of the update gate. The final output ($h(t)$) of this memory block will be used in the next memory block [85].

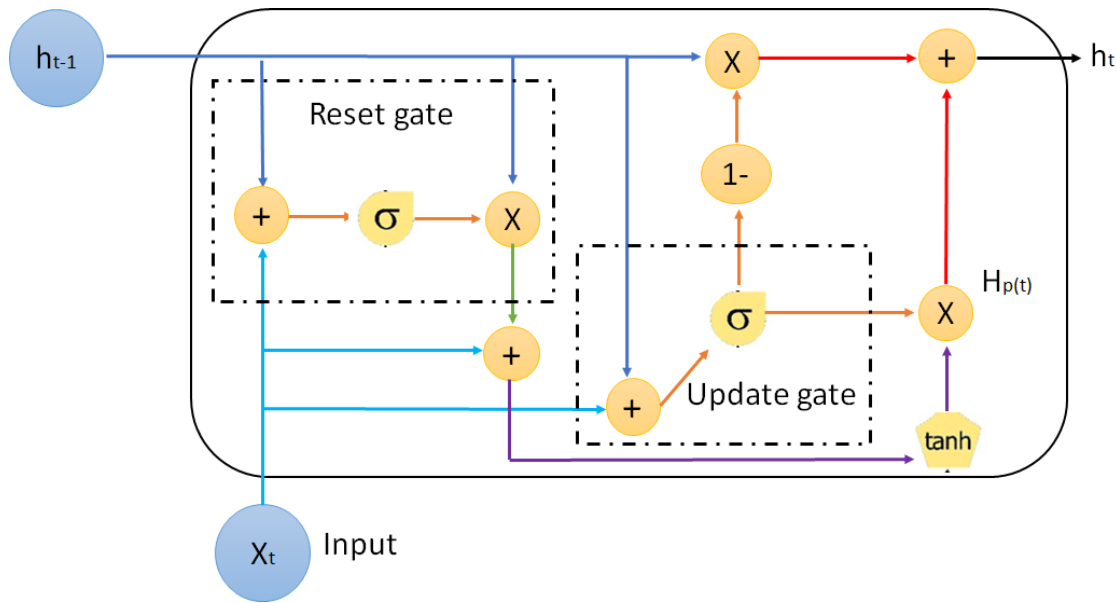


Figure 13: The structure of the GRU neural network. Adapted from [85].

To summarize, GRU offer a simplified and more computationally efficient alternative to LSTM. Despite their simpler structure, GRU are capable of capturing long-term dependencies in data sequences, making them suitable for many natural language processing and time series applications [85]. However, LSTM, with their more complex architecture, may still be preferable in scenarios where modeling very long-term dependencies and more intricate relationships within the data are crucial [82].

In addition to traditional RNN architectures, bidirectional RNNs have emerged as a powerful approach to analysing complex biological sequences such as proteins. By processing the sequence in both the forward and reverse directions, these models capture contextual information from both directions, providing a more detailed understanding of the interactions between amino acids along the chain. This bidirectional approach can significantly improve the modelling of long-range dependencies, which are typical in biological sequences [42].

Transformers

Recurrent architectures in Natural Language Processing (NLP), such as LSTM, were widely used for sequential data like text. However, these approaches faced challenges when dealing with long sequences of data [42, 79]. In 2017, the introduction of transformers by Vaswani et al., with their innovative attention-based approach, overcame these limitations, representing a significant advance in understanding more extensive contexts in language models [86].

The transformers are built with encoding and decoding blocks. The attention mechanism, essential to this architecture, processes data in parallel, eliminating sequential dependency [87]. In contrast to the LSTM approach, where each token in the sequence is represented by a hidden state, the encoder in a transformer architecture transforms all the information in the input sequence into a set of hidden states simultaneously, expressed as a numerical vector. The decoder has the function of reverting the vector representation provided by the encoder back to the original information. However, when dealing with long sequences, there is a risk of information loss. To get around this challenge, the attention mechanism comes into play, an essential feature of this architecture [87, 88].

The attention mechanism plays a key role between the encoder and decoder. While the increase in hidden states in the decoder can result in a voluminous input, attention allows it to select specific states of the encoder to focus on. This ability to give differentiated weight to certain states optimises processing and efficiency in generating the output sequence, thus improving the overall performance of the model [88].

Nowadays, there exist three main types of transformer models. Encoder-only models, exemplified by BERT [89], aim to craft robust representations of input sequences. These models undergo training using Masked Language Modeling, where specific parts of the sequences are masked for subsequent reconstruction. On the other hand, decoder-only models, including GPT [90], are designed to predict the next token in a sequential context. Finally, encoder-decoder models, such as BART [91], adeptly map complex sequences from one form to another, being particularly suited for sequence-to-sequence tasks like translation, text summarization, and question-answering [88].

Alternative architectures

The previously mentioned architectures are the most popular among the DL community. However, other architectures have also proven to be highly useful, such as Autoencoder (AE) [92].

AE can be employed for tasks like dimensionality reduction and the creation of generative models. The key element is the "bottleneck" layer, where the number of units is typically reduced compared to other layers [92]. This layer forces the model to learn a compressed representation of the input data, capturing the most relevant features. AE can be built using various sub-architectures, including DNN, CNN, and RNN, each bringing unique strengths to different types of data and tasks [92, 74, 79].

A classical AE creates a latent vector space capable of reconstructing the original data. The typical architecture includes an encoder and a decoder. The encoder is responsible for transforming the data into the latent space, while the decoder performs the reverse operation, reconstructing the original data

from this latent representation [92]. This latent space can then be utilized in various ML and DL models for tasks such as classification and regression. The advantages of using the latent space include reduced dimensionality, which leads to faster training times and improved performance, as well as the extraction of the most relevant features, enhancing the model's ability to generalize from the data [92, 42, 44].

2.3.5 Feature Selection

In some cases, especially in biological problems, it is advantageous to reduce the number of input attributes to simplify the analysis of the data and the models created. This can be achieved by extracting features or selecting the most important ones [47].

There are three main classes of algorithms for feature selection: filter, wrapper, and embedded methods. Filter methods select features independently of the model, using statistical measures. Wrapper methods create different subsets of attributes, which are evaluated by training a model and estimating its error. Examples of heuristic wrapper algorithms include forward and backward selection. The forward selection method starts with a few attributes and adds one more to the model each time until acceptable results are obtained, while the backward selection method starts with all attributes and removes one at each iteration. Finally, embedded methods perform feature selection along with model creation and evaluation simultaneously [47, 93].

2.3.6 Hyperparameter tuning

When training ML and DL models, it is essential to consider two types of parameters: model parameters and hyperparameters. Model parameters are learned during training, while hyperparameters define the model configuration and are set before training begins [1].

The proper selection of hyperparameters aims to minimize errors and can be done manually or automatically. Manual hyperparameter tuning, while sometimes effective, involves adjusting hyperparameters by hand, making it a labor-intensive and time-consuming process, even for experts. To overcome these challenges, automatic hyperparameter tuning methods have been developed [49].

The grid search method efficiently explores all possible combinations of hyperparameter values specified by the user, though it can become computationally expensive due to the exponential growth in combinations as additional values are included. On the other hand, random search mitigates this issue by selecting a fixed number of combinations randomly within defined ranges, offering a more efficient approach to discovering high-performing models. Beyond these methods, other advanced techniques like Bayesian optimization, successive halving, and hyperband further enhance the optimization process [49, 1].

2.3.7 Model evaluation

A thorough evaluation of predictive models is crucial for gauging their reliability and effectiveness. This process entails more than mere performance testing; it necessitates a systematic evaluation of their ability to generalize to new data and achieve intended task objectives [49].

Different error estimation methods, such as holdout, cross-validation, and leave-one-out, are used to evaluate the model's performance [49]. In holdout, the data is split into training and test sets, usually using 2/3 for training and 1/3 for testing. Cross-validation, the most commonly used method, divides the data set into k folds, using one fold for testing and the remaining $k-1$ for training, and iterating this over the k folds. The Leave-one-out method, less common due to the computational cost, makes N (the number of samples) error estimates, using each sample as a test set in each iteration [49, 58].

After this stage, it is crucial to analyze the metrics in detail to assess the predictive capacity of the model used. The metrics chosen vary depending on the nature of the problem, be it classification or regression [94].

The metrics for evaluating binary classification problems are grounded in the confusion matrix, which compares predicted labels with real labels, organizing the outcomes into four categories: true positives, false negatives, false positives, and true negatives (Figure 14 in blue) [1, 94]. From these categories, essential metrics emerge, including accuracy, precision, and recall, among others, as illustrated in Figure 14 in green.

However, it is crucial to strike a balance between certain metrics, as an increase in sensitivity may lead to a decrease in specificity. In this context, Receiver Operating Characteristic (ROC) curves play a pivotal role by correlating sensitivity with the false positive rate (specificity) [1, 49, 58]. Furthermore, the area under the ROC curve quantifies the model's performance. In situations of class imbalance, Precision Recall (PR) curves prove more suitable than ROC curves for evaluating these models [49].

It is also essential to emphasise the change in approach when tackling multi-class classification problems. Unlike binary classification, where the metrics are derived directly from the confusion matrix, in multi-class problems it is essential to calculate these metrics individually for each class. This approach provides a more thorough evaluation of the model's performance in different categories, allowing for a more detailed understanding of how the model performs in each specific class [1].

Predict \ REAL	Positive	Negative	
Positive	True Positive (TP)	False Negative (FN)	Recall = $\frac{TP}{TP+FN}$
Negative	False Positive (FP)	True Negative (TN)	Specificity = $\frac{TN}{FP+TN}$
	Precision = $\frac{TP}{TP+FP}$	False Negative Rate = $\frac{FN}{TN+FN}$	Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$
Other important metrics			
F1-score = $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$		MCC = $\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$	

Figure 14: A confusion matrix in blue and commom metrics in green. Adapted from [1].

For regression problems, other types of metrics are used, the most popular being Mean Square Error (MSE), Root Mean Square Error (RMSE), Mean Absolute Error (MAE) and Coefficient of Determination (R^2), shown in table 1 [49, 58]. These metrics provide a comprehensive view of the model's accuracy in terms of absolute and relative errors and are essential for effective evaluation in regression problems [49].

Table 1: Common metrics for evaluating regression models, where y are the real outputs and $\hat{y}$ the predicted outputs [49, 58].

Metric	Formula	Description
Mean Square Error (MSE)	$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$	Mean squared error between the real and predicted values.
Root Mean Square Error (RMSE)	$RMSE = \sqrt{MSE}$	Root mean square error between the real and predicted values.
Mean Absolute Error (MAE)	$MAE = \frac{1}{n} \sum_{i=1}^n y_i - \hat{y}_i $	Average absolute error between the real and predicted values.
Coefficient of Determination (R^2)	$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$	Measure of prediction quality for unseen samples, calculating the proportion of variance explained.

The evaluation of ML models is crucial to determine whether they are underfitted, overfitted, or properly fitted (Figure 15) [1]. Underfitting, characterized by high bias, results in poor adaptation to the training data, while overfitting, with high variance, leads to a lack of generalization to new data.

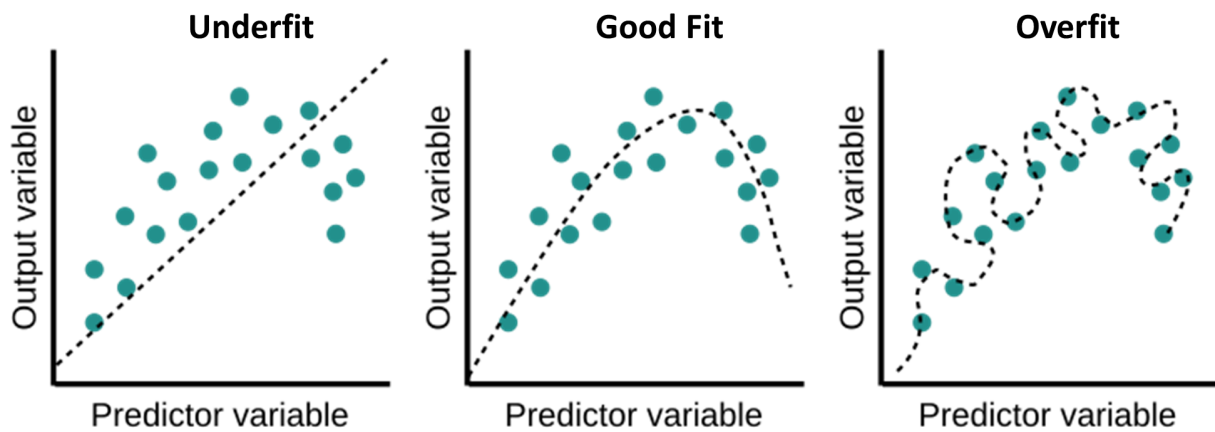


Figure 15: Behaviour of ML models: Underfitting when the model fails to capture the complexity of the data, resulting in poor performance; Good fit which is where the model fits the training data efficiently and can generalize well to unseen data; Overfitting when the model adapts too much to the training data, losing generalization to new data. Adapted from [1].

Underfitting occurs when a model is too simplistic to capture the underlying patterns in the training data, leading to poor performance on both the training and validation/test datasets [49]. Various techniques are employed to improve the model's complexity and ability to fit the data better, including:

- **Increasing Model Complexity:** Using more complex models with additional layers or neurons can help capture more intricate patterns in the data [49].
- **Feature Engineering:** Creating new features or modifying existing ones can provide the model with more relevant information, improving its ability to learn from the data [49].
- **Reducing Regularisation:** While regularisation helps prevent overfitting, too much regularisation can lead to underfitting. Adjusting regularisation parameters can help find a balance [49].
- **More Training Data:** Increasing the amount of training data can provide the model with more examples, helping it learn better and generalize more effectively [49].
- **Hyperparameter Tuning:** Adjusting hyperparameters such as learning rate, batch size, and number of epochs can improve the training process and model performance [49].

Overfitting arises due to the ability to capture an extensive variety of patterns in the training data [42]. Various techniques are employed to improve the generalisation capacity of these models, including:

- Regularisation: Adds terms to the loss function during training, penalising weights that are too large and avoiding excessive adjustments to the training data [49].
- Dropout: Randomly deactivates neurons during training, preventing the network from relying too heavily on specific parts and promoting better generalisation [49, 42].
- Early Stopping: Monitors performance on a validation set, stopping training when performance stops increasing for a specified number of iterations (epochs) [49, 42].
- Transfer Learning: Uses pre-trained models in related tasks to initialise weights, taking advantage of prior knowledge, especially useful when training data is limited [42].

Therefore, accurate predictions in new data require an equilibrium between overfitting and underfitting. Achieving this balance ensures that the model is complex enough to capture the underlying patterns in the data, yet general enough to perform well on unseen datasets [49].

2.4 Automated machine learning

The idea of automation in ML, known as AutoML, is promising. Over time, ML algorithms and strategies have evolved, and the complexity associated with these methods has increased significantly. The automation provided by AutoML can provide substantial benefits, making the application of ML techniques more accessible and efficient [95].

By simplifying the development of ML pipelines, AutoML democratizes access to these technologies, allowing even users without in-depth knowledge of the area to enjoy the benefits of ML. This accessibility is particularly valuable in complex scenarios, where aspects such as selecting and engineering features, choosing algorithms, and adjusting hyperparameters can be challenging [96].

Figure 18 shows the general pipeline used in ML, highlighting the stages where the application of AutoML takes place, delimited by dashes. AutoML plays an active role in optimizing the selection of relevant features, as well as in the automatic generation of new features, simplifying the feature engineering process by finding patterns and relationships in the data. In addition, it automates the choice of the model and its hyperparameters, training several models to find the best combination, and eliminating the need for extensive manual adjustments. In short, AutoML focuses mainly on areas that can be optimized automatically, while some responsibilities, such as initial data preparation and final model validation, remain important and often user-driven tasks [95, 96].

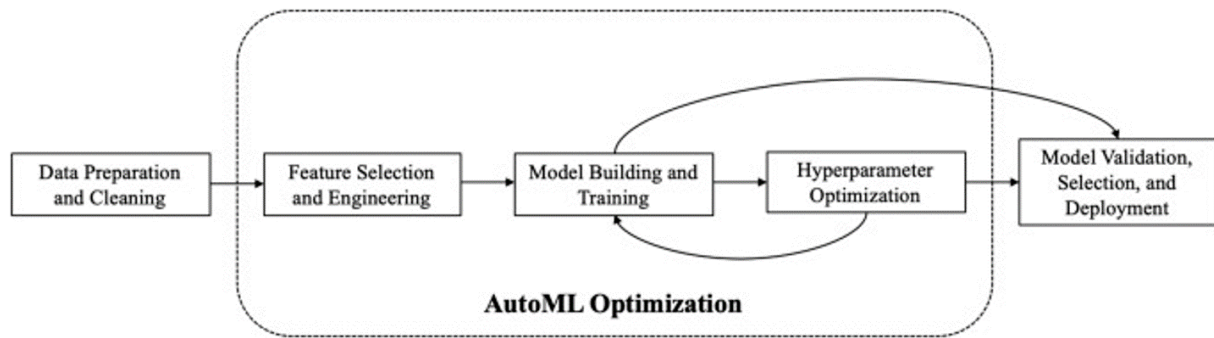


Figure 16: The fundamental stages of an ML pipeline are highlighted, with feature selection and engineering, choosing the appropriate model and optimizing hyperparameters being particularly focused on AutoML, simplifying these processes in an automated way [96].

The history of AutoML models is recent, but since the creation of AutoWEKA [97] in 2013, several tools that realize AutoML have been developed. Some notable examples include:

- Auto-SKlearn [98]: Automates the selection and configuration of algorithms, as well as hyperparameter tuning.
- TPOT [99]: uses genetic algorithms to optimize complete ML pipelines, including feature selection, feature engineering, and hyperparameter tuning.
- H2O [100]: Automates the ML workflow, which includes automatic training and tuning of various models.
- Auto-Keras [101]: automates the design of neural network architectures, simplifying the creation of DL models.
- Auto-PyTorch [102]: AutoML system based on the PyTorch library.
- Auto-Gluon [103]: automates ML tasks on image, text, time series, and tabular data.
- Neural Network Intelligence (NNI) [104]: Open-source AutoML toolkit for feature engineering, neural architecture search, hyperparameter optimization, and model compression.

2.5 Machine and deep learning applied to proteins

This section will focus first on the use of tools to enhance studies on protein properties, where numerical alternatives to represent protein sequences (through descriptions and encodings) are essential, and then on articles that have studied in more detail the classification of a set of proteins according to their antioxidant or allergenic properties.

2.5.1 Protein descriptors

In predicting protein functions and interactions, several properties play a crucial role, such as composition, amino acid distribution, highly mutable locations, and structural organization. These characteristics serve as descriptors in protein studies and are collected during the feature extraction process, providing explicit or implicit information about the protein samples [105]. Table 2 summarises the main types of descriptors, providing examples and corresponding descriptions [106]

Table 2: Main types of descriptors, as well as an example of each and its description [106].

Type of descriptors	Examples of descriptors	Description
Physicochemical	Isoelectric point	The pH at which a protein has zero net charge, indicating the electrical neutrality of the molecule, influencing its solubility and biochemical interactions.
Structural	Van der Waals	Refer to the properties related to the space and interaction between the hydrophobic and hydrophilic regions of molecules.
Sequence order	Sequence order coupling numbers	Statistical descriptors that capture specific correlations between consecutive pairs of amino acids.
Sequence Composition	Aminoacid composition	Represents the proportion of each type of amino acid.

Common descriptors include amino acid, dipeptide, and tripeptide composition, counting the number of each amino acid, pairs, and triples of amino acids in each sequence, respectively. The pseudo-amino acid composition, which provides additional details on the position of the amino acids, is often used. Physicochemical descriptors, covering properties such as charge and hydrophobicity, and autocorrelation descriptors, which consider residue position information, are widely employed, taking the analysis into a more comprehensive protein dimensional space [107].

In addition to physicochemical properties, structural descriptors, such as hydrophobicity scale, Van der Waals volume, among others, are used. Structural representations based on torsion angles, secondary structure elements, and other structural components are valid alternatives [108]. Binary profiles, indicating the presence or absence of sequence motifs (for example, protein motifs from databases such as PFAM), also offer valuable insights. This variety of descriptors highlights the multifaceted approach to protein characterization, especially in the context of ML and DL [106].

2.5.2 Sequence encoding

Proteins are made up of sequences of letters, each representing an amino acid residue. However, ML and DL models are unable to process raw text directly. Given this limitation, it is necessary to transform the text into a numerical tensor, a process known as vectorizing text. Therefore, only by properly representing the protein in a mathematical vector can the models extract crucial information about the composition of these proteins [42].

A common technique for encoding sequences is one-hot encoding, which converts tokens into binary vectors with the length of the vocabulary. In this process, the text is divided into tokens, which can represent individual words or characters. The resulting vector is predominantly made up of zeros, except for a specific index that corresponds to each token. Thus, if the residue corresponds to the corresponding amino acid, the value will be 1, otherwise it will be 0 [42].

Another relevant technique for coding protein sequences is the use of substitution matrices, such as the Block Substitution Matrix (BLOSUM). The score assigned to two residues reflects the probability that they are aligned in a homologous sequence, and is derived from the study of sequence conservation in large databases of related proteins. Matrices with lower numbers are more suitable for aligning sequences with lower identity, while those with higher values excel at identifying highly conserved regions. These substitutions, based on the conservation of protein sequences, provide an effective representation of the evolution and functionality of amino acids, facilitating analyses and modelling in bioinformatics studies applied to molecular biology [109].

The Non-Linear Fisher (NLF) matrix, which represents the supervised transformation of features using the NLF transformation, is another common approach to coding protein sequences. It aims to preserve the distances between amino acid residues by incorporating eighteen specific numerical characteristics [110]. In addition, Z-scales, which cover physicochemical properties calculated for each residue, such as hydrophobicity or charge, is another viable option. In this coding method, each amino acid is replaced by the corresponding list of property values, following the order of the residues in the protein sequence. This strategy provides detailed information about each amino acid, enhancing ML and DL models [111].

The use of embedding layers has emerged as an alternative to conventional one-hot coding. This embedding layer compacts the input feature space into a smaller space by identifying the optimal mapping of each unique token to a vector of real numbers. Techniques such as word2vec and protVec, used in this context, perform distribution coding to characterize the relationships between tokens, making it possible to represent co-occurrence relationships between blocks of amino acids rather than between individual amino acids [112, 113].

Another effective approach involves applying Position-Specific Scoring Matrices (PSSM) or Position-Specific Frequency Matrix (PSFM) to understand evolutionary constraints in protein families [109]. These matrices, derived from multiple sequence alignments, highlight specific residues at certain positions, identifying possible conserved regions. High positive scores indicate high amino acid conservation. These PSSM based descriptors improve prediction in bioinformatics applications and are applicable in sensitive searches for similarities between sequences, as demonstrated by Position-Specific Iterative Basic Local Alignment Search Tool (PSI-BLAST) [114].

Transformers, widely used in sequence data such as text, have found effective applications in protein sequences. Notable examples include ProtTXTL, ProtBert, ProtXLNet, ProtAlbert, ProtElectra, ProtT5-X, and ProtT5-XXL, all derived from transformers and trained on protein sequence data [87]. A notable example is Evolutionary Scale Modeling (ESM), which originated from the BERT model developed by Facebook AI. This model is a transformer designed for tasks such as predicting variant effects, reverse folding, and predicting contacts. During training, a specific task involves predicting a percentage of masked amino acids, forcing the model to learn complex relationships [115]. These representations, similar to BERT embedding layers, serve as information-rich encodings of the characteristics of amino acid sequences. In summary, the application of these transformers as feature extractors to represent protein sequences has demonstrated superior performance in several related tasks [89].

2.5.3 Relevant work

The use of pipelines that employ ML and DL models to predict the properties of proteins has created considerable interest in the scientific community. Table 3 highlights various approaches for predicting antioxidant and allergenic properties, the case studies in this work. These pipelines generally follow five main steps: (1) collection of protein sequences from specific databases and processing to form a dataset, (2) feature extraction to capture various characteristics of the protein sequences, (3) feature selection to enhance model performance, (4) model training using appropriate algorithms, and (5) model evaluation to ensure accuracy and reliability of the predictions [116].

It is important to note that, based on the literature consulted, there could have been other articles included in the table. However, we have specifically selected those that will be used in a later stage of the work to carry out benchmarking using the different tools created and available to the user.

Table 3: Recent studies applying ML and DL methodologies to predict the antioxidant (blue) and allergenic (green) properties of proteins.

Classification Task	Tool Created	Best Strategy	Ref.	Year
Antioxidant	AodPred	Protein descriptors + SVM	[117]	2016
	AOP-SVM	Protein descriptors + SVM	[118]	2019
	PredAoDP	Protein descriptors + SVM	[119]	2022
	DP-AOP	Protein descriptors + SVM	[120]	2023
Allergenic	Allertop V2	Protein descriptors+ KNN	[121]	2014
	AlgPred 2.0	Protein descriptors + Random forest	[122]	2020
	ProAIID	E-descriptors + Auto cross-covariance transformation + LSTM	[123]	2022
	DeepAlgPro	One-hot encoding + CNN	[124]	2023

Overall, the articles mentioned in Table 3 follow similar premises in the data collection phase, although they may differ in the origin of this data. The authors emphasize the importance of keeping the similarity between the amino acid sequences in the dataset below 70%. The exclusion of non-standardized amino acids is necessary, imposing limitations on the dataset, which is already naturally restricted. Thus, the main challenge lies in the need to build a more robust dataset, taking into account the aforementioned premises and the imbalances between positive and negative samples that are commonly observed, especially when the positive classes are numerically limited [125, 123].

In the feature extraction phase, it is imperative to convert each amino acid sequence into a numerical tensor, as detailed in 2.5.1 and 2.5.2. The second stage varies greatly between authors, with different approaches, such as one-hot encoding [124], and the use of protein descriptors [117, 118, 119, 120, 121, 122, 123]. Within the realm of protein descriptors, there are various types addressed across different studies demonstrating that no single set of descriptors is universally applied. However, the common goal remains: to numerically encode amino acid sequences to enable the application of ML and DL models.

Regarding the feature selection stage, some articles use specific techniques, such as Analysis of Variance (ANOVA) [122] or Maximum Relevance Minimum Redundancy (MRMR) [117, 118, 119, 120]. However, other studies, like the others mentioned in the table, choose not to apply any feature selection method. Both approaches are valid, but in the end it is crucial to consider not only performance metrics, but also the time and computational resources involved.

In the model training and evaluation phase, the methodology adopted is fairly consistent among the articles, with variations mainly in the validation techniques used. While some studies apply cross-validation [119, 120, 121, 122, 123, 124], others utilize the leave-one-out method [117, 118]. Although the models explored vary between studies, the general strategy is the same: test multiple ML or DL models, and the table highlights only the model with the best results.

Concerning antioxidant case studies, no available tool was identified that used a DL model as the best strategy. However, Usman et al [126] presented a strategy using DL for predicting antioxidant properties in proteins. This approach employs Composition of k-Spaced Amino Acid Pairs (CKSAAP) as a coding technique to capture patterns in amino acid sequences, alongside a DNN (Deep-LSE) as a decoder, which allows the model to learn abstract, non-linear representations of the encoded features. The combination of CKSAAP as the encoder and the neural network as the decoder is designed to extract a non-redundant feature space, enabling the model to incorporate the original features non-linearly, filter out irrelevant data, and achieve significant improvements compared to other methods. Although the AOP-LSE tool from this work is not available for direct benchmarking, the protein descriptor type and model developed by Usman et al. will be considered during the development of this project.

To summarise, predicting the antioxidant and allergenic properties of proteins involves various strategies, all anchored in the steps mentioned above. Therefore, this subsection highlights the challenges overcome by recent work and points out more advantageous alternatives compared to previous approaches to facilitate and improve future research in this field [116].

2.5.4 Relevant packages and tools

As ML and DL have gained popularity, Python has emerged as a dominant language with a rich ecosystem of packages and tools. This thesis leveraged several key Python libraries, each playing a crucial role in its development and implementation. Below is a brief summary of the essential packages utilized:

- NumPy is a fundamental package for scientific computing in Python, known for its efficient handling of multi-dimensional arrays. It provides a wide range of functions for mathematical operations, logical operations, shape manipulation, sorting, selection, I/O operations, and more. NumPy is highly efficient and is widely used in various fields such as mathematics, physics, engineering, and data science for its performance and ease of use.
- Pandas is a powerful Python library designed for data manipulation and analysis. It provides high-level data structures which are ideal for handling structured and labeled data. Pandas offers functions to load data from various file formats such as CSV, Excel, SQL databases, and more. It allows for fast and efficient data manipulation operations such as filtering, sorting, grouping, merging, and reshaping data.
- Scikit-learn is an open-source library for ML frameworks, offering a versatile architecture for following established pipelines in traditional supervised and unsupervised learning models. Its simplicity gives it a competitive advantage over its predecessors [49].
- TensorFlow stands out as a robust platform designed by the Google Brain team for conducting ML and DL research. It offers a comprehensive ecosystem of libraries, tools, and community resources. TensorFlow excels in developing and training neural network models for various applications. It boasts high performance on powerful servers, supports exporting models to different devices, and scales efficiently on Graphics Processing Units (GPUs), making it widely adopted in the field [49, 58].
- Keras, integrated with TensorFlow, simplifies the process of building NNs. It provides a high-level approach to model building and sequential compilation, emphasizing simplicity, flexibility, and scalability. Keras is renowned for its user-friendly design, making it an excellent choice for both beginners and seasoned experts in ML and DL [49].
- PyTorch is a prominent ML and DL library, often regarded as an enhanced version of the traditional NumPy Python library. It has gained significant popularity among researchers and developers for its intuitive approach to creating DL models. PyTorch excels in tensor operations and offers functionalities like automatic differentiation, making it versatile for building complex models. As an

open-source framework, PyTorch supports the implementation of deep learning models and facilitates efficient manipulation of numerical data, including vectors, matrices, and tensors. Its computational capabilities are accelerated by GPUs, leveraging a majority of its backend operations in C++, which contributes to its high performance. PyTorch is also recognized for its optimization capabilities in scientific computing within the Python ecosystem.

- AutoGluon, developed by Amazon Web Services, is an advanced AutoML package designed to automate the creation of ML and DL models. It simplifies complex tasks such as data preprocessing, hyperparameter tuning, and ensemble model building across various data types including tabular datasets, images, and text. AutoGluon is widely recognized for its robust capabilities in achieving high predictive performance across diverse applications, from image classification to natural language processing and time-series predictions [103].
- NNI is a tool designed to automate and optimize ML and (DL) models. Focussing on automated model creation, it covers automated data processing, feature engineering, and hyperparameter tuning. In addition, it supports various tasks related to neural networks, contributing to the efficient development and improvement of AI models.
- Optuna is an automatic hyperparameter optimization framework for ML tasks. It allows users to define a search space of hyperparameters, specify an objective function to optimize, and automatically search for the best set of hyperparameters. Optuna uses various algorithms such as Tree-structured Parzen Estimator (TPE), Grid Search, and Random Search to explore the hyperparameter space efficiently.
- Fair-esm is a specialized Python package developed by Facebook AI for seamless access to state-of-the-art transformer-based protein language models. These models, including ESM2, ESM1, and MSA Transformer, are pre-trained by Facebook AI Research. Fair-esm enables researchers and developers to effortlessly download and utilize these models to extract per-residue representations, leveraging the advanced capabilities of transformer architectures in protein sequence analysis.
- Transformers is an Application Programming Interface (API) provided by Hugging Face that facilitates the use of pre-trained transformer models across various domains such as NLP, Computer Vision, Audio, and Multimodal applications. It allows users to access state-of-the-art models trained on large generic datasets, enabling tasks like text classification, language generation, image understanding, and more. Additionally, the Transformers API supports interoperability with both PyTorch

and TensorFlow frameworks, making it versatile for integration into different ML pipelines and environments. This flexibility and the extensive range of available models empower researchers and developers to leverage cutting-edge transformer technology efficiently in their applications.

- ProPythia is a versatile Python-based tool designed for applying ML and DL pipelines for protein studies. It stands out for its modular architecture and flexibility, which enable adaptation to various protein-related problems and user needs. Unlike other packages, ProPythia simplifies the implementation of ML/DL pipelines, making it accessible even to users without extensive coding experience. It automates critical steps in protein analysis, including dataset generation, feature calculation and selection, and application of both unsupervised and supervised learning methods. Using the libraries mentioned above, facilitates sequence processing, descriptor generation, and comprehensive data preprocessing. Its capabilities span from traditional ML algorithms to advanced DL models, empowering researchers with powerful tools for protein classification tasks. ProPythia's modular design ensures adaptability without sacrificing overall usability, making it a valuable resource for the scientific community engaged in protein studies. For more information and to download ProPythia, visit its GitHub repository at: <https://github.com/BioSystemsUM/propythia> [106].
- OmniA is an AutoML tool developed by OmniumAI, a spin-off from the University of Minho founded in 2021 specializing in AI and Data Sciences for bioinformatics and biomedical data. This tool leverages the expertise of the OmniumAI team in ML and DL to provide advanced solutions for biological and biomedical data processing, analysis, mining, and integration. It utilizes packages mentioned above to streamline tasks like data processing, feature extraction and selection, model selection and evaluation, and hyperparameter tuning. More details about this package and its applications will be discussed in the following chapter. For more information, visit <https://www.omniumai.com/home>.

Chapter 3

Methods

This chapter describes the datasets used, as well as the architecture of the *OmniA* platform, with a specific focus on the *Proteins* and *Generics* sub-packages, as these are the ones related to the project under study. It also makes reference to other tools discussed in the literature, namely the Propytha framework developed at the University of Minho in the Bioinformatics and Systems Biology research group.

3.1 Data Collection

The first critical step in training ML and DL models is identifying or creating datasets that meet the specific needs of the task. In this project, the focus is on addressing protein classification challenges, particularly those related to antioxidant and allergenic properties. To effectively address these challenges, a thorough literature review was conducted to identify a single, well-established dataset for each case study allowing for more generalized conclusions to be drawn from the outcomes of this work.

3.1.1 Antioxidant dataset

The dataset selected for this case study was the same used by Meng et al. [120]. To construct the dataset, protein sequences were collected from the Universal Protein Resource (UniProt) database, and sequences containing non-standard amino acids were removed. This resulted in 710 protein sequences being retained as the original positive samples for the experiment.

To generate the negative sample set, 1567 protein sequences were extracted from the Protein Data Bank (PDB) database using the PISCES program, ensuring that no more than 20% sequence similarity was allowed between proteins [127].

Furthermore, sequences with more than 60% identity were filtered out from the benchmark dataset using the CD-HIT program [128]. After this preprocessing, a final dataset of 1805 protein sequences was obtained, consisting of 253 antioxidant proteins (positive samples) and 1552 non-antioxidant proteins (negative samples). Therefore, the resulting dataset was highly imbalanced, with the ratio of positive to negative samples being approximately 1:6.

3.1.2 Allergenic dataset

The dataset selected for this case study was the same used by He et al. [124]. It consisted of 3550 allergen and 3550 non-allergen protein sequences, ensuring a balanced dataset.

The allergen sequences were extracted from the Structural Database of Allergenic Proteins (SDAP), International Union of Immunological Societies (IUIS) Allergen Nomenclature, Comprehensive Protein Allergen Resource (COMPARE), AllergenOnline and UniProt databases using specific allergen-related keywords. Additionally, datasets from previous studies (AllerTOP [121] and AlgPred [122]) were also included. CD-HIT was then used to remove sequences with more than 40% identity and an alignment coverage greater than 0.8 for the shorter sequence [128]. Sequences longer than 1000 amino acids and those containing non-standard characters were excluded.

The non-allergen sequences were collected from UniProt using the query 'Protein Existence: Evidence at protein level NOT allergen NOT allergenic NOT allergy NOT cancer NOT antigen AND reviewed: yes'. These sequences were further filtered to exclude proteins with more than 40% similarity to allergens and among themselves. Sequences containing non-standard characters were also removed.

3.2 OmniA

As this project aims to enhance the capabilities of *OmniA*, it is crucial to understand the existing functionalities integrated into its architecture. *OmniA*, with its modular design illustrated in Figure 17, includes a range of methods for processing and analyzing biological data. These functionalities are organized into specific sub-packages developed in Python:

- *Generics*: contains basic and universal data handling and transformation methods. It provides the ability to load data, transform data, and train ML models. All sub-packages listed below depend on this sub-package.
- *Compounds*: Contains specific methods for compound data. It is capable of performing compound feature extraction, compound standardization, and data splitting based on molecular information.
- *Genes*: contains specific methods to process DNA sequences. It is capable of performing DNA feature extraction.
- *Metabolomics*: contains specific methods for processing and scaling metabolomics data.
- *Transcriptomics*: Provides methods for processing transcriptomics data, including RNA sequence (RNA-seq) and single-cell RNA sequence (scRNA-seq), with capabilities for parsing, scaling, feature selection, and instance selection.
- *Text mining*: contains methods to process text data. It is capable of performing text feature extraction and text processing.
- *Proteins*: Contains specific methods to handle protein sequences. It provides a set of methods implemented to perform protein standardization, calculate protein descriptors and perform the encoding of protein sequences.

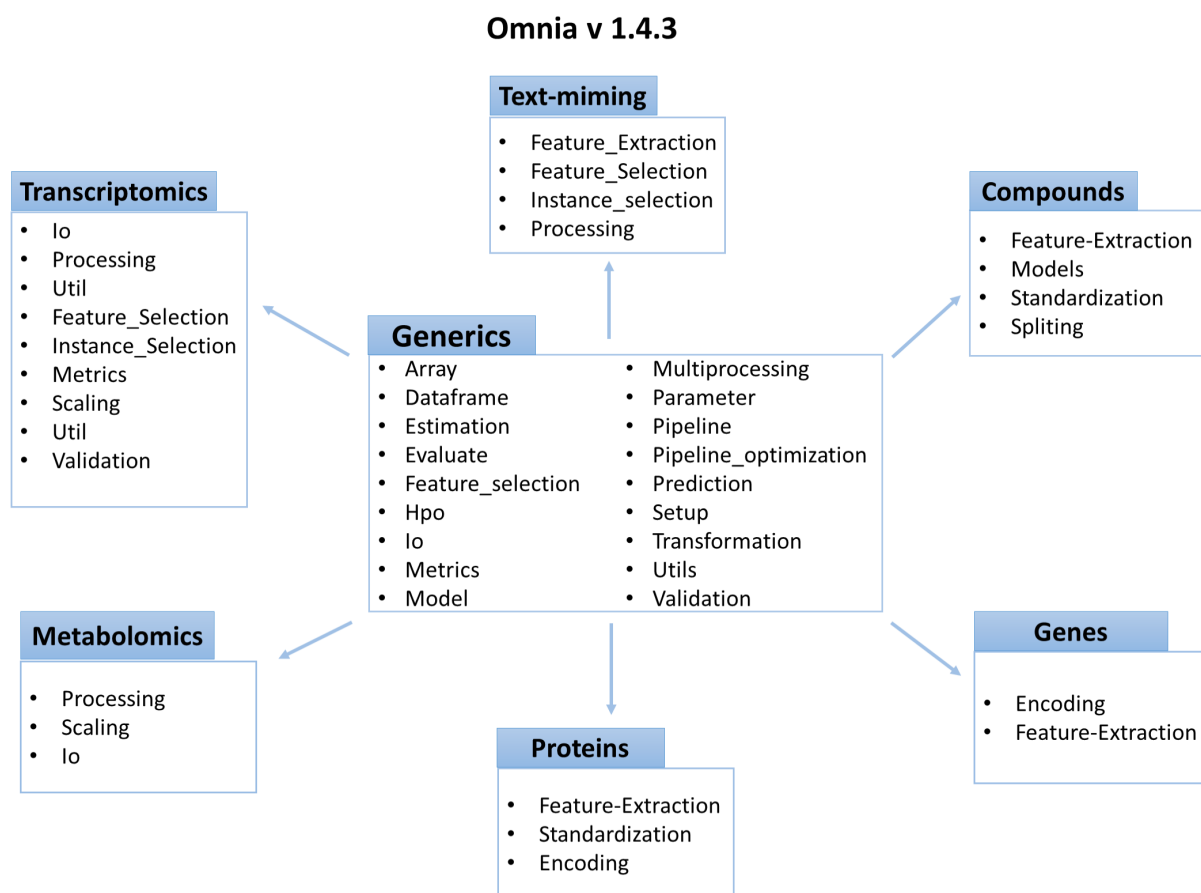


Figure 17: *Omnia* architecture and respective methods implemented in each subpackage.

To achieve the project's goal mentioned above, the primary focus will be on further developing the *Proteins* and *Generics* sub-packages. This enhancement is particularly valuable as it aligns with the development of an AutoML platform.

3.2.1 Proteins sub-package

The *Proteins* sub-package plays a pivotal role in this project. To fully understand its significance, it is essential to delve into its structure and functionalities, which are illustrated in Figure 18.

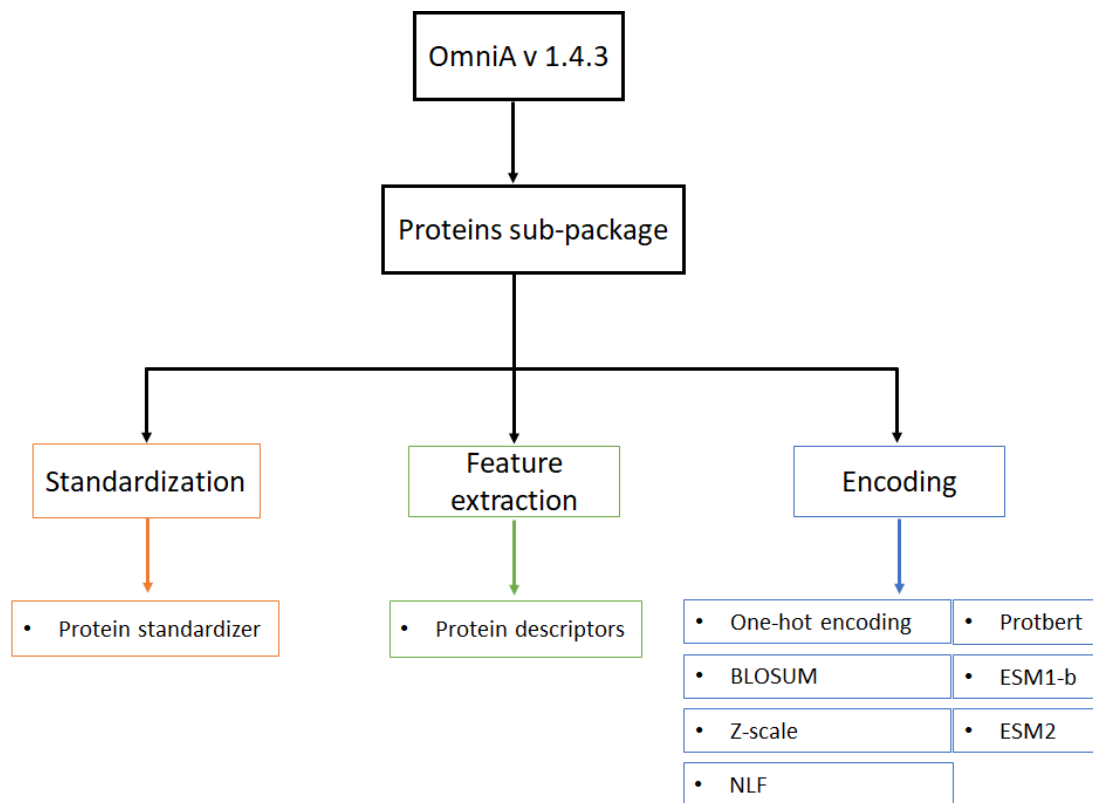


Figure 18: The Proteins sub-package in *OmniA* and its implement modules.

As illustrated in Figure 18, the *Proteins* sub-package is organized into three primary modules: Protein Standardization, Feature Extraction, and Encoding. Each method addresses specific aspects of protein sequence analysis. This modular organization ensures flexibility and usability, allowing researchers to employ specific methods as required. Detailed discussions on each component will follow below.

Protein standardization

Protein sequencing can present limitations in identifying the amino acids in a protein, which results in sequences containing non-standard amino acids, such as selenocysteine (U) and pyrrolysine (O), or letters representing ambiguities between amino acids.

The letter B represents the uncertainty between aspartic acid (D) and asparagine (N), Z between glutamic acid (G) and glutamine (Q) and J between leucine (L) and isoleucine (I). X represents an unknown or unspecified amino acid in the sequence. In such cases, it is often necessary to standardize protein

sequences by removing or substituting these non-standard amino acids with their closest standard equivalents. For instance, B may be replaced by asparagine (N), Z by glutamine (Q), selenocysteine (U) by cysteine (C), and pyrrolysine (O) by lysine (K). X is typically removed during this process [116].

Standardizing protein sequences is essential for preparing data, and ensuring consistency and compatibility in subsequent analyses like feature extraction and model training [1].

Feature extraction

Protein descriptors are features derived from the protein sequence that capture various biochemical properties of the amino acid residues, as discussed in subsection 2.5.1. The Feature extraction module facilitates the calculation of diverse descriptors essential for ML applications. These descriptors encompass physico-chemical properties, amino acid composition, pseudo amino acid composition, auto-correlation, sequence order, and other relevant characteristics [106].

Encoding

Besides using protein descriptors, another approach to achieve this is through encoding methods. The encoding methods present in *OmniA*, which have already been detailed in subsection 2.5.2, include one-hot encoding, BLOSUM, NLF, Z-scales, and attention-based transform.

For one-hot encoding, each amino acid residue is represented as a binary vector of length 21, covering the 20 standard amino acids and one ambiguous residue (X). Consequently, this results in a matrix with 21 columns and N rows, where N corresponds to the sequence length.

The BLOSUM62 matrix encodes standard and non-standard amino acid residues, excluding the ambiguous amino acid J, into a vector of length 24, resulting in a 24 x N matrix. BLOSUM50 encodes all amino acid residues into a vector of length 25, producing a 25 x N matrix. The columns represent the substitution likelihood scores [109].

The NLF encoding is applied exclusively to protein sequences that have undergone a standardization process, ensuring they contain neither non-standard residues nor ambiguous amino acids. This encoding generates a matrix with 18 columns and N rows. Similarly, the Z-scale encoding is applied to standardized sequences, where each amino acid residue is represented by five attributes: Z1 (lipophilicity), Z2 (steric properties), Z3 (electronic properties), Z4 (electronegativity), and Z5 (electrophilicity). This encoding generates a matrix with 5 columns and N rows [42].

ProtBert and ESM models are transformers with attention-based architectures used for encoding protein sequences. ProtBert, trained on approximately 216 million proteins from the UniRef100 dataset, captures biophysical properties related to protein shape. Each amino acid residue is transformed into a numeric vector with a dimension of 1024, resulting in a $1024 \times N$ representation for a protein sequence [89].

The ESM models offer various dimensions based on the number of trainable parameters. For instance, ESM2 with 8 million parameters generates a vector of 320 features per amino acid residue, while ESM2 with 15 billion parameters produces a vector of 5120 features per residue. The default ESM model in Omnia, ESM2-650, comprises 33 layers and 650 million trainable parameters, producing a representation similar to the ESM1-b model but trained with updated reference data.

3.2.2 Generics sub-package

The *Generics* sub-package provides a range of models from the AutoGluon framework, including both traditional ML algorithms and NN enabling efficient model selection and tuning for various tasks. The models included in this sub-package are:

- **RandomForestModel:** A widely used ensemble model that constructs multiple decision trees during training and outputs the mode of the classes (for classification) or mean prediction (for regression) of the individual trees.
- **CatboostModel:** Is an algorithm for gradient boosting on decision trees, where loss is reduced with each consecutively tree.
- **KNNModel:** A KNN algorithm that classifies samples based on the majority class of their nearest neighbors in the feature space.
- **LGBModel:** is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with advantages in regard to training speed, efficiency, accuracy, among others.
- **LinearModel:** Implements linear regression or classification models. These models serve as simple yet effective baselines, particularly in cases where the relationship between features and the target variable is linear or approximately linear.

- **FastAIINN:** A DL model built on the FastAI library, which leverages NN to capture complex patterns in the data. This model is well-suited for tasks where data has intricate, non-linear relationships, though it typically requires a sufficient amount of data to perform effectively.
- **VowpalWabbitModel:** A fast online ML model designed to work efficiently with large datasets. It excels in scenarios requiring fast learning and prediction, often used in recommendation systems and real-time applications.
- **XGBoostModel:** An advanced implementation of gradient boosting that supports both classification and regression tasks. Is highly efficient and capable of handling large datasets with high performance.
- **XTModel:** Also known as extra trees, this ensemble learning method builds multiple unpruned decision trees with randomized splits in feature selection, reducing variance and controlling overfitting while improving predictive accuracy.
- **MultilayerPerceptron:** A type of NN composed of multiple layers of nodes. MLP are powerful models that can capture complex, non-linear relationships in the data and are suitable for a wide range of prediction tasks.

In addition to these individual models, AutoGluon offers powerful techniques for model optimization and ensemble learning. AutoGluon's Tabular Predictor automatically manages model selection and hyperparameter optimization across these algorithms. It incorporates weighted ensembling, allowing it to combine the strengths of different models by assigning weights based on their individual performance, thus improving generalization on unseen data. Furthermore, AutoGluon employs techniques such as bagging, which trains multiple models on random subsets of the data to reduce variance, and stacking, where models are combined in a layered architecture to enhance predictive accuracy.

A key enhancement to this process is the integration of Optuna, a hyperparameter optimization framework that can work in synergy with AutoGluon. Optuna allows for more efficient hyperparameter tuning through advanced optimization strategies such as TPE. By leveraging Optuna, AutoGluon can intelligently search for the best hyperparameter settings across models, reducing the number of trials needed and speeding up the optimization process. This ensures that critical parameters like learning rates, regularization terms, among others are adjusted to maximize model performance while minimizing overfitting risks.

By integrating Optuna and AutoGluon, *OmniA* significantly improves the efficiency of model selection and hyperparameter tuning across a diverse range of algorithms, as previously discussed. This combination offers strong adaptability for both simple and complex machine learning tasks, ensuring robust and scalable performance. While the platform optimizes each model's parameters to achieve optimal results, there remains potential for further enhancement, particularly through the integration of additional DL models, which could further boost performance in more demanding scenarios.

3.2.3 Global pipeline

It is important to note that the primary objective of *OmniA* is to serve as an AutoML tool. This involves not only automatically selecting the best ML or DL model but also choosing the entire pipeline that ensures the best final result and optimal predictive capacity. In this context, Figure 19 demonstrates the sequence of events during the execution of the AutoML pipeline created by *OmniA*, as well as the different elements required to execute the pipeline effectively.

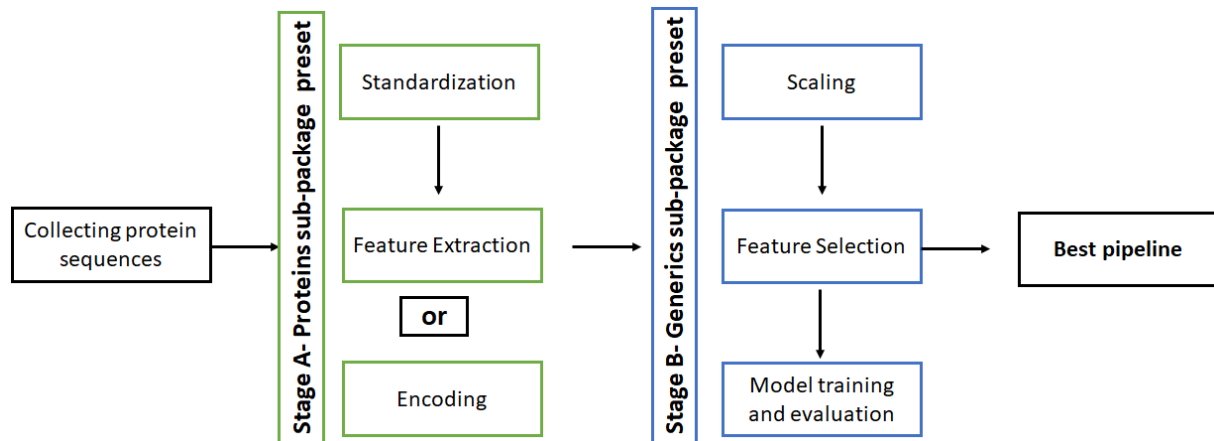


Figure 19: Sequence of steps in the AutoML pipeline implemented in *OmniA*, emphasizing the role of presets in the Proteins (Stage A) and Generics sub-packages (Stage B).

The pipeline encompasses various stages, where the sub-packages for *Proteins* and *Generics* play a crucial role. Additionally, to better understand the pipeline's operation, it is essential to be familiar with the concept of presets. Presets refer to predefined configurations or settings that simplify and standardize the use of certain functionalities. They provide users with optimized options for specific tasks or scenarios, reducing the need for manual customization and ensuring consistency across different runs or applications.

The pipeline assumes that a set of sequences and their respective labels are already available, forming the foundation for the subsequent steps in dataset processing and model development. In Stage A (Figure 19 green), the *Proteins* sub-package provides the necessary methods to transform the dataset through standardization, feature extraction, or encoding techniques. Stage B (Figure 19 blue), managed by the *Generics* sub-package, guides the steps for scaling, feature selection, model training, and evaluation.

The presets control how these steps are applied, offering three levels of complexity: light, medium, and heavy. The choice of complexity allows the pipeline to be tailored to the specific requirements and constraints of the experiment, balancing performance and computational resource usage.

3.3 Propythia

ProPythia, a tool previously presented in subsection 2.5.4, is a powerful, Python-based, free tool designed for applying ML and DL models, with a strong emphasis on protein classification [106]. Given this context, to perform a comparative analysis and truly verify the advantages of *OmniA*, we decided to compare it with ProPythia.

Once the dataset, containing the set of sequences and their respective labels, is available, the numerical representation of each sequence (with feature extraction or encoding techniques) is performed. A grid search space was then defined with the parameters to be optimized (as detailed in Table 4) for the shallow ML models implemented in ProPythia. Hyperparameter optimization was carried out using a randomized search with 50 iterations and 5-fold cross-validation.

Table 4: Machine learning models used in ProPythia alongside their respective hyperparameters, which were tuned using randomized search optimization.

Models	Parameter Grid	
SVC	C	0.01, 1.0, 10
	kernel	rbf, linear
	gamma	'auto', 'scale'
	class_weight	None, balanced
Linear-SVC	C	0.01, 1.0, 10
	penalty	l2
	class_weight	None, balanced

Random-Forest	n_estimators	10, 100, 500
	max_features	sqrt, log2
	penalty	l2
	criterion	gini, entropy
	class_weight	None, balanced
GBoosting	n_estimators	10, 100, 500
	max_features	0.6, 0.9
	max_depth	1, 3, 5, 10
	learning_rate	0.1, 1
KNN	n_neighbors	2, 5, 10, 15
	weights	uniform, distance
	leaf_size	15, 30, 60
SGD	loss	hinge, log, modified_huber, perceptron
	alpha	0.00001, 0.0001, 0.001, 0.01
	n_iter_no_change	5, 10
	class_weight	None, balanced
Logistic regression	C	0.01, 0.1, 1.0, 10.0
	solver	liblinear, lbfgs, sag
	class_weight	None, balanced
NN	hidden_layer_sizes	(50,), (100,), (200,)
	activation	tanh, relu
	solver	adam, sgd
	alpha	0.00001, 0.0001, 0.001
	learning_rate_init	0.0001, 0.001, 0.01

Alongside the use of ProPythia's shallow ML models, we also developed a DL model for further experimentation, which will be described in the development section (4.4.3). In summary, Propythia includes functionalities for building predictors, optimizing hyperparameters, evaluating models, and making predictions. Additionally, it supports a range of models, including SVM, Random Forest, KNN, as well as neural network-based approaches like NNs and RNN, among others.

Chapter 4

OmniA development

This chapter outlines the enhancements made to *OmniA*, focusing on the modules developed within the *Proteins* and *Generics* sub-packages. Most of the code was created for OmniumAI and is subject to sharing policies; therefore, the GitHub repository (<https://github.com/GuilhermeLoboSousa/thesis>) contains only scripts for data exploration, comparative analysis with Propytha, and the general *OmniA* pipeline. Python 3 was chosen for development due to its compatibility with the existing OmniumAI infrastructure and its extensive libraries for implementing AutoML software.

The enhancements to *OmniA* involve integrating new functionalities across two main sub-packages: *Proteins* and *Generics*. We will first discuss the implementations in the *Proteins* sub-package (subsections 4.1, 4.2, 4.3) followed by a detailed overview of the improvements in the *Generics* sub-package (4.4).

4.1 Feature extraction

The *Proteins* sub-package includes a module dedicated to feature extraction, particularly for protein descriptors. This module is organized into files, with each file handling specific categories of descriptors and containing classes that compute various protein descriptors. New descriptors— Grouped Amino Acid Composition (GAAC), CKSAAP, Conjoint Triad (CT), and Dipeptide Deviation from Expected Mean (DDE)—were implemented and integrated into this module. These descriptors were chosen for their frequent use in relevant case studies related to this work. Their inclusion in the *Proteins* package is significant as they provide new methods for analyzing protein sequences, uncovering complex structural and functional relationships that were previously not addressed.

4.1.1 Grouped amino acid composition

As known, the 20 standard amino acids can be categorized into 5 groups based on their properties: the aliphatic group (g1: GAVLMI), aromatic group (g2: FYW), positively charged group (g3: KRH), negatively charged group (g4: DE), and uncharged group (g5: STCPNQ). In this context, the class called GAAC was created to calculate the frequency of each amino acid group, resulting in 5 descriptors, as described in Equation (1).

$$\text{Group Amino Acid Composition } (g) = \frac{N(g)}{N} \quad (1)$$

where $N(g)$ is the number of amino acids in group g , and N is the total length of the protein sequence.

4.1.2 Composition of K-spaced amino acid pairs

The CKSAAP class allows to calculate the frequency of each pair of amino acids separated by any k residues. There are 400 possible amino acid pairs, so the feature vector created by this class will have a length of 400, unless more than one value for k is chosen. In that case, the feature vector length will be 400 multiplied by the number of k values selected. Equation (2) illustrates how each descriptor associated with this class is calculated:

$$\text{Frequency}(AA) = \frac{N_{AA}}{N - (k + 1)} \quad (2)$$

where N is the total length of the protein sequence and N_{AA} is the number of times the amino acid pair "AA" appears in the protein sequence separated by k residues. Thus, as an example, in the sequence "AYAYKDAGAAC" with a length of 11, the CKSAAP descriptor for the pair "AA" would have a value of $\frac{2}{11-(1+1)}$ for $k = 1$.

4.1.3 Conjoint triad

The CT class categorizes amino acids into 7 groups based on their properties: g1 (AGV), g2 (ILFP), g3 (YMTS), g4 (HNQW), g5 (RK), g6 (DE), and g7 (C). The method counts how often combinations of three groups occur together. Since there are 7 possible groups for each position in the combination, the descriptor vector size is 343 ($7 \times 7 \times 7$).

However, as the protein sequence length increases, the probability of the descriptor vector containing higher values compared to smaller proteins also increases. To mitigate this, a normalized descriptor vector d is defined, as shown in Equation (3):

$$d = \frac{f - \min(f_1, f_2, \dots, f_{343})}{\max(f_1, f_2, \dots, f_{343})} \quad (3)$$

where f is the descriptor vector for a specific combination of three units of amino acid groups. Normalization is carried out considering the minimum and maximum values obtained in the descriptors.

4.1.4 Dipeptide deviation from expected mean

The DDE class allows for the extraction of descriptors that explore the relationship between the observed frequency of dipeptides and the expected frequency based on a random distribution of amino acids within the protein sequence. To achieve this, three parameters need to be computed: dipeptide composition (D_C), theoretical mean (T_m), and theoretical variance (T_v). Using the dipeptide "AC" as an example, we demonstrate the computation of these parameters in Equations (4), (5), and (6), respectively.

$$\text{Dipeptide composition } (D_C) = \frac{N_{AC}}{N - 1} \quad (4)$$

where N_{AC} is the number of occurrences of the dipeptide AC in the protein sequence, and N is the length of the protein sequence.

$$\text{Theoretical mean } (T_m) = \frac{C_A}{C_N} \cdot \frac{C_C}{C_N} \quad (5)$$

where C_A represents the number of times the amino acid 'A' occurs, C_C represents the number of times the amino acid 'C' occurs, and C_N is the total number of amino acids in the protein sequence.

$$\text{Theoretical variance } (T_v) = \frac{T_m \cdot (1 - T_m)}{N - 1} \quad (6)$$

The DDE is calculated by combining these parameters, as shown in Equation (7):

$$\text{Dipeptide deviation expected mean (DDE)} = \frac{D_C - T_m}{\sqrt{T_v}} \quad (7)$$

where D_C , T_m , and T_v correspond to dipeptide composition, theoretical mean, and theoretical variance, respectively.

4.2 Encoding

In most datasets, including those used in this project, proteins vary in sequence length. To ensure consistency during processing, padding and truncation techniques are used. Padding involves adding zeros or predefined values to shorter sequences to match either the length of the longest sequence or a predefined maximum length. Conversely, truncation shortens longer sequences by removing tokens until a predefined maximum length is achieved. These techniques are critical for ensuring uniform sequence lengths, facilitating batch processing, and maintaining compatibility with models that require fixed-length inputs. In this work, padding and truncation were applied across all three encoding methods: NLF, ESM, and ProtBert.

4.2.1 NLF encoder

The NLFEncoder class supports truncation in three distinct ways: by removing tokens from the start, from the end, or by discarding tokens from the middle, focusing on the terminal regions of the protein sequence. The latter method is particularly relevant since key functional and structural characteristics of proteins often reside at their terminals. This flexibility allows us to explore how different truncation strategies affect model performance while preserving essential biological features.

4.2.2 ESM-2 encoder

The Esm2Encoder class also includes parameters that allow different encoding approaches. A key parameter is *pretrained_model*, which generates a transformation matrix with dimensions that depend on the selected model. These dimensions are indicative of the model's capacity to capture complex protein characteristics. When choosing a *pretrained_model*, it is essential to balance the accuracy of the protein representation with the available computational resources.

Another important parameter is *two_dimensional_embeddings*, a boolean flag that dictates the output format. When set to True, the encoded data is presented as a matrix; when set to False, it is output in a tabular format. This flexibility is crucial, as it impacts the compatibility with downstream models and influences whether ML or DL approaches will be more effective in utilizing the encoded data.

4.2.3 ProtBert

The ProtBert class represents a model pretrained on protein sequences using the Masked Language Modelling (MLM) technique. Based on the BERT model, it was trained in a self-supervised way on a large corpus of protein sequences, without the need for human labelling. This approach allows the model to learn and generate protein representations automatically by leveraging large public datasets [129].

4.3 Presets

The *OmniA* platform is an AutoML solution, making the creation of presets essential for simplifying its configuration and use, particularly within the *Proteins* sub-package. Three main types of presets have been implemented, each orchestrating different aspects of the platform: one more specific for the feature extraction module, and two associated with the pipeline (highlighted in green and blue in Figure 19).

The feature extraction preset streamlines the selection of descriptors. Table 5 lists the 10 descriptor presets implemented, along with an explanation of the descriptor groups that each one extracts. This preset facilitates quick selection, enabling users to easily choose an appropriate set without the need for manual configuration.

Table 5: Descriptor preset string values that generate a set of descriptors from protein sequences.

Presets	Description
all	All descriptors available in <i>OmniA</i>
performance	Descriptors that tend to generate high-performing models
physico-chemical	Physicochemical characteristics of proteins
aac	Single amino acid, dipeptide and tripeptide composition
paac	Pseudo amino acid composition - returns the amino acid composition as well as the sequence order information
auto-correlation	Physicochemical and molecular structure-based descriptors
composition-transition-distribution	Calculates the composition, transition rate, and quartile distribution of amino acids for each physicochemical property.
seq-order	Calculates dissimilarity values for all residue pairs.
modlamp-correlation	Correlation descriptors from the modlAMP python package
modlamp-all	All descriptors available from the modlAMP python package

The two pipeline presets are designed to cater to different user needs, featuring increasing levels of complexity. While they offer a more tailored user experience, this added complexity may lead to decreased processing efficiency. Users can choose the preset that best aligns with their requirements, balancing complexity and efficiency. These presets are available in three distinct levels of complexity:

- *Light*: Offers fast configurations ideal for preliminary tests or when computational resources are limited.
- *Medium*: Balances performance and processing time with a moderate number of configurations for more robust analyses;
- *Heavy*: Involves complex and computationally intensive configurations, maximizing performance through a comprehensive set of choices.

The capabilities of each general preset and the distinctions between these levels of complexity are illustrated in Table 6, highlighting how the choices and computational demands increase with each level.

Table 6: Overview of preset configurations for protein sequence analysis, detailing the complexity levels (Light, Medium, and Heavy) and the respective choices available for standardization and feature extraction or encoding.

Preset	Standardization	Feature extraction or Encoding
Light	Protein Standardizer	Protein Descriptor (all) NLFEncoder ESM2Encoder ("8M" and "35M")
Medium	Protein Standardizer	Protein Descriptor (all) NLFEncoder ESM2Encoder ("35M" and "150M" and "650M")
Heavy	Protein Standardizer	Protein Descriptor (all) NLFEncoder ESM2Encoder ("150M" and "650M" and "3B") ProtbertEncoder

In conclusion, the implementation of both descriptor presets facilitate the quick and effective selection of descriptor groups, while the general presets offer flexibility in choosing the desired level of complexity, adjusting the balance between the number of choices, computation time, and performance as required.

4.4 Deep learning models

The *Generics* sub-package was improved through the addition of four DL models: an Autoencoder and a 1D CNN for handling tabular data, and a CNN-RNN and RNN for working with matrix data. Table 7 provides a detailed overview of the default key hyperparameters for each model, including model-specific and general, relevant to our classification problem. These values can be significantly customized, allowing the models to be highly adaptable and flexible for various data types and tasks.

Table 7: The developed deep learning models feature hyperparameters with default values that can be optimized during training, including individual for each model and globalshared across all models.

Models	Hyperparameters	
	Model-specific	Global
AutoEncoder	num_dense_unit_encoder (2) num_dense_layers (2) latent_dim (32) first_hidden_dense_unit (256) num_dense_layers_classifier (32)	n_tasks (1) n_classes (1) activation (ReLU) last_layer_activation (sigmoid) epoch (100) loss (binary-crossentropy) batch_size (32)
CNN1D	n_conv_layers (2) first_filter_size (128) kernel_size (3) stride (1) pool_size (2) n_dense_layers (1) first_dense_units (64)	
RNN	RNN_type (LSTM) bidirectional (False) n_rnn_layers (3) hidden_units_rnn_layers (64) num_dense_layers (1) hidden_units_dense_layers (32)	
CNN-RNN	All model-specific and global parameters are common with the CNN and RNN models	

In addition, regularization techniques, such as dropout layers, have been incorporated to prevent overfitting and improve generalization, ensuring reliable performance across diverse applications.

The inclusion of DL models in the *Generics* sub-package has greatly expanded its scope. These new models provide flexibility for handling both tabular and matrix data, allowing for more efficient and adaptable solutions to a wider range of tasks.

4.4.1 AutoEncoder

The AutoEncoderMLP class is a NN model that integrates an autoencoder with a Multi-Layer Perceptron (MLP). This architecture is designed to simultaneously reduce data dimensionality and perform classification or regression tasks. The model comprises three main components: the encoder, the decoder, and the MLP, as illustrated in Figure 20.

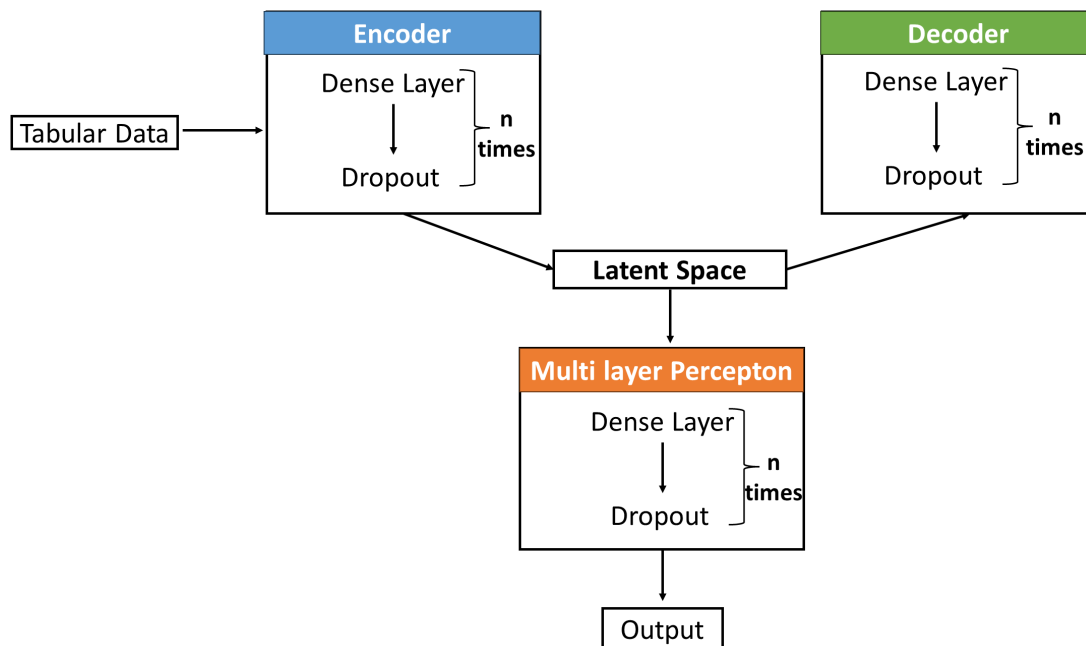


Figure 20: Architecture of the AutoEncoderMLP model, illustrating the main components: encoder, decoder, and Multi-layer Perceptron

The encoder compresses the input data into a smaller latent space by reducing its dimensionality, while the decoder mirrors the encoder to reconstruct the original input from this latent space. The compressed data is then passed through the MLP network, which executes the final classification or regression task. The MLP ends with a dense layer, applying an appropriate activation function, such as sigmoid for binary classification.

To ensure effective dimensionality reduction and accurate task performance, the model utilizes a combination of loss functions during training. The MSE loss function is specifically employed by the autoencoder to measure the difference between the input and output, ensuring that the reconstructed output closely resembles the original input. For the other models, the choice of loss function depends on the task: binary cross-entropy loss is applied for classification tasks, while MSE is used for regression tasks. This approach allows all models to optimize their performance according to the specific requirements of the task at hand.

4.4.2 CNN1DModel

The CNN1DModel class is a one-dimensional CNN designed for classification or regression tasks. The network's architecture, detailed in Figure 21, is structured to handle tabular data by applying 1D convolutional layers followed by dense layers for final prediction.

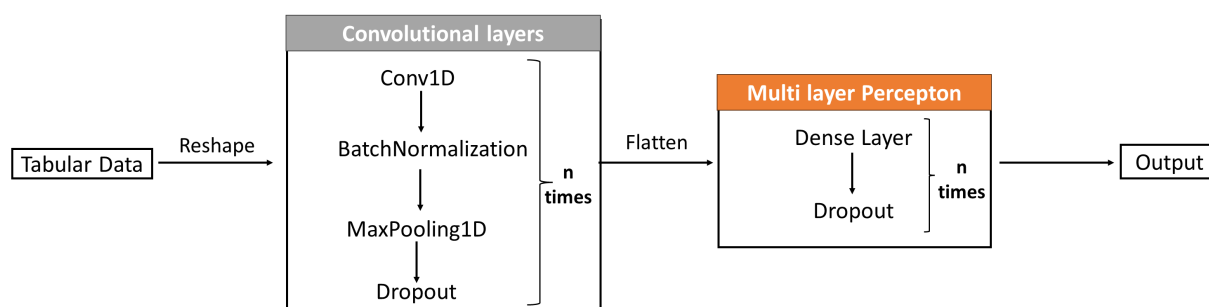


Figure 21: Architecture of the CNN1DModel, showing the 1D convolutional neural network and dense layers used for processing tabular data.

The input data undergoes a reshaping process to make it suitable for the 1D convolutional layers. Each Conv1D layer is followed by batch normalization and max pooling layers, which stabilize the learning process and reduce the data's dimensionality while retaining essential features. After the convolutional layers, the data is passed through a flattening layer that converts the multi-dimensional output into a one-dimensional vector. This vector is then fed into a series of dense layers that perform the final prediction, whether for classification or regression.

The CNN1D model combines the strengths of convolutional and dense layers, making it a powerful tool for capturing local and sequential patterns in the data from protein sequence descriptors and generating accurate predictions.

4.4.3 RNNModel

The RNNModel class is a RNN designed for classification or regression tasks on matrix data. The model's architecture, as shown in Figure 22, offers flexibility by allowing users to choose between various RNN layers like LSTM, GRU, or SimpleRNN. Additionally, these layers can be configured as bidirectional, enabling the capture of information from both forward and reverse sequences.

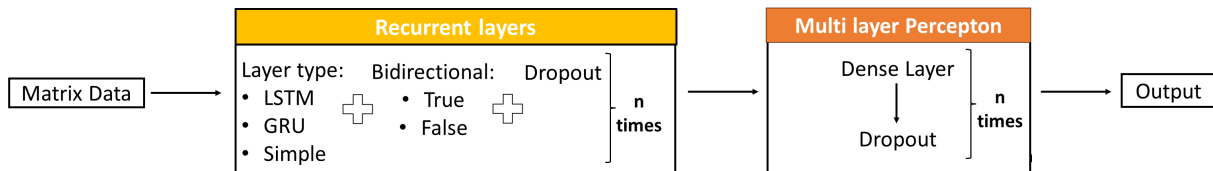


Figure 22: Architecture of the RNNModel, illustrating the use of RNN layers, including options for LSTM, GRU, or simpleRNN, with bidirectional configuration.

The RNN layers excel at extracting and consolidating temporal dependencies, whether they are local or more distant. After processing through these layers, the outputs are passed to dense layers that utilize the extracted features to generate predictions.

The RNNModel is well-suited for effectively handling and analyzing sequential data. The model is trained using techniques like backpropagation through time and optimized through gradient descent methods. With its extensive configuration options, this model serves as a versatile tool for a wide range of applications. However, it is important to note that while RNN are powerful, they may struggle with long-range dependencies due to issues such as vanishing gradients, necessitating careful consideration during model training.

4.4.4 ConvRNNModel

The ConvRNNModel class is a NN that combines convolutional, recurrent, and dense layers, similar to the architectures of previously mentioned models. This model, illustrated in Figure 23, is designed for classification or regression tasks on matrix data.

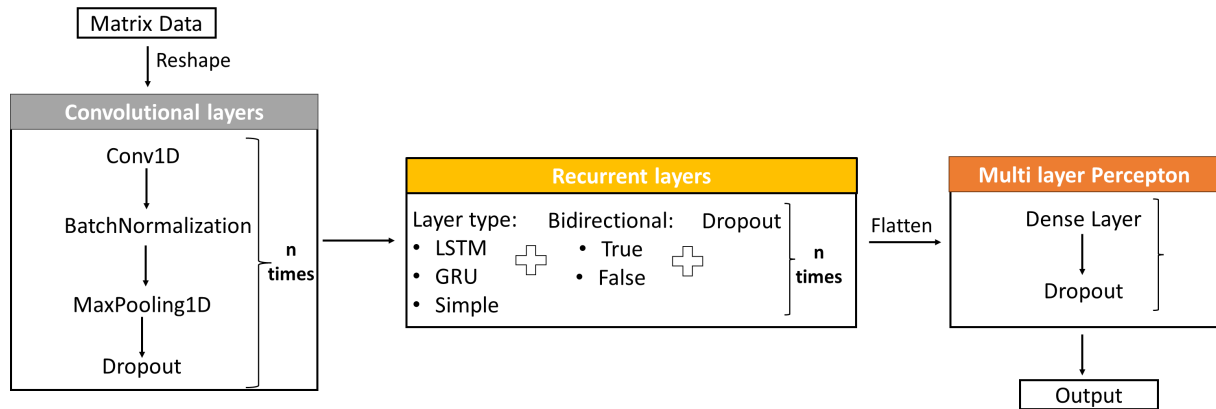


Figure 23: Architecture of the ConvRNNModel, combining convolutional and recurrent layers to extract spatial and temporal features from matrix data.

The convolutional layers are designed to extract spatial patterns, by applying filters across the input. This step is crucial for detecting spatial hierarchies and reducing dimensionality while retaining important information. Next, the recurrent layers capture temporal dependencies across the sequence of spatial features, allowing the model to learn from the order and progression of data over time. After the convolutional and recurrent layers, the output is passed through dense layers, which integrate the learned features and produce the final prediction.

In summary, the ConvRNNModel is a versatile architecture well-suited for tasks that involve matrix data with inherent spatial and temporal patterns. This makes it applicable to a wide range of problems.

4.5 Global Pipeline

The presets mentioned above describe three possible levels of complexity. However, given the computational resources available, it was decided to apply only the light preset. In the Generics sub-package, the preset allows the use of all the ML and DL models, but only with the hyperparameters set by default. To make this pipeline easier to understand, Figure 24 illustrates how it works with the light preset.

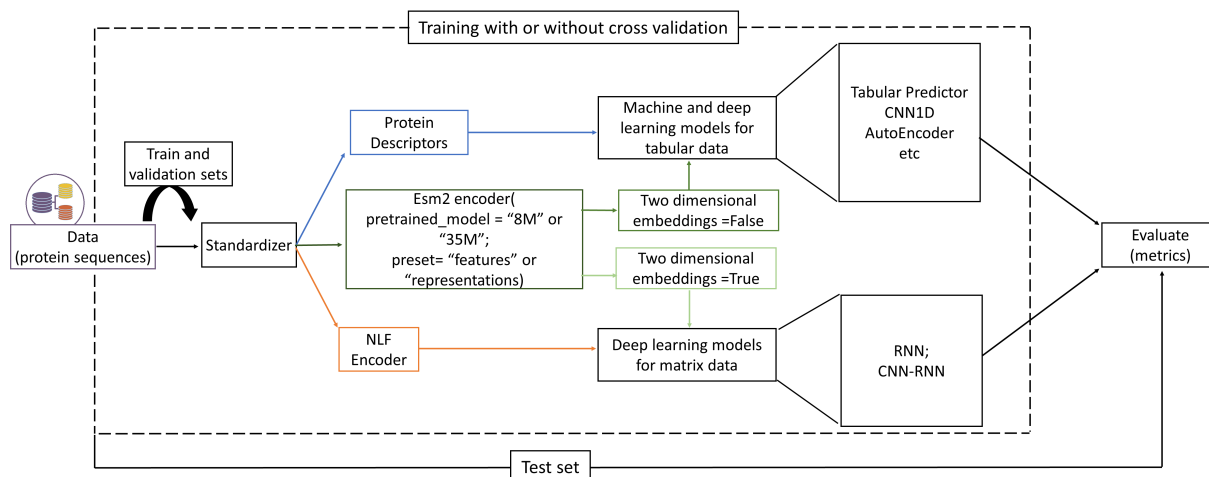


Figure 24: Representation of the *OmniA* pipeline using the light configuration preset. In the image, black represents fixed default settings, while colored sections indicate the parameters that Optuna can optimize within the light preset. This configuration streamlines the process, automating adjustments while retaining flexibility for specific pipeline configurations.

An analysis of Figure 24 clearly demonstrates how the AutoML platform operates with the least complex preset, and is notable for its “hands-off” approach. This means that it can be used by users with specialist knowledge and those with less experience, simply by providing the data set. Furthermore, it can be seen that although the light preset uses fewer computing resources, it still allows for a considerable range of different configurations (more than 700), guaranteeing flexibility without compromising computing efficiency.

Chapter 5

Results and discussion

This chapter presents the results and discussion of the aforementioned case studies, focusing on the distribution of datasets in terms of sequence length and binary classification classes, as well as on the application of the updated AutoML platform, which integrates all the improvements discussed in Chapter 4. The results obtained with the AutoML platform were compared with models from the literature and with the Propythia tool, providing a comprehensive performance analysis. This approach aims to highlight the competitiveness of the solution developed in relation to the state of the art, identifying its strengths and areas for improvement in different contexts, providing a complete assessment of the platform's capabilities.

5.1 Data analysis

5.1.1 Sequence length

First of all, it is important to note that in the allergen study, both the training and test sets are provided. In contrast, only the global dataset is available in the antioxidant study, without a pre-defined training-test split. Therefore, although the main dataset is the same in both case studies, when compared to the data used in the original publications, we can only guarantee an exact match to the training and test sets used for the allergen study. This discrepancy makes it impossible to ensure a fully fair comparison in the antioxidant case study, as the test split may differ from the original setup, potentially affecting the results.

To better understand the datasets to be explored, we analyzed the length distribution of the protein sequences, as shown in Figure 25. This analysis was essential for ensuring consistency across the datasets and for gathering valuable insights for the later stages of the work. The length distribution directly influences the choice of preprocessing techniques, such as padding or truncation, which are necessary for models that require fixed-length inputs. By understanding this distribution, we can apply appropriate padding or truncation to preserve key biological information.

Additionally, the selection of the encoding method plays a crucial role. For example, complex models like ProtBert are well-suited for handling long or varied sequences, while simpler encoders may be sufficient for shorter ones. This understanding also impacts computational efficiency, as longer sequences demand more memory and processing power. Furthermore, significant variations in sequence lengths can introduce bias during model training, leading to certain lengths being over- or underrepresented. Thus, this analysis ensures that our preprocessing steps effectively mitigate these risks, resulting in models that generalize better to new data.

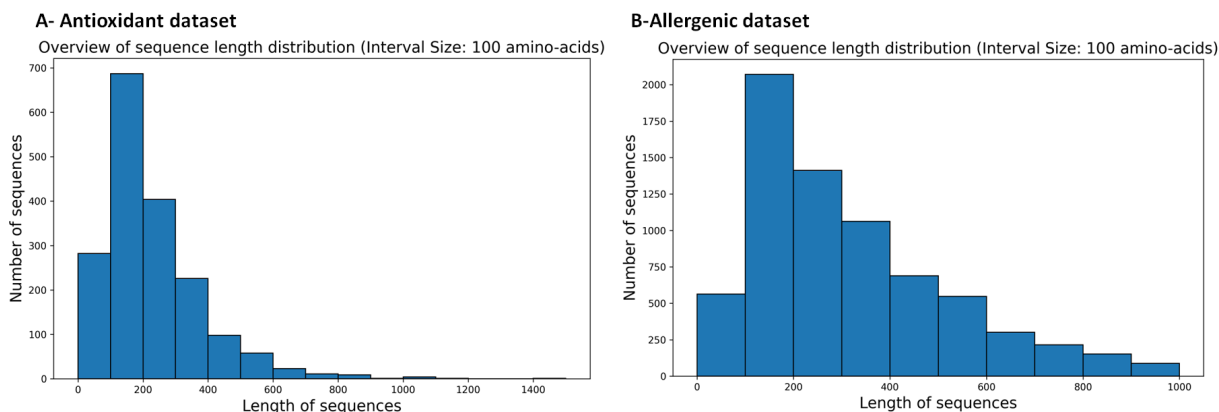


Figure 25: Sequence length distribution histograms for (A) the antioxidant dataset and (B) the allergen dataset.

For both antioxidant (Figure 25-A) and allergen (Figure 25-B) datasets, the histograms indicate that most sequences fall within the 100 to 200 amino acid range, exhibiting a left-skewed distribution. Notably, only around 10% of the sequences exceed 600 amino acids in the allergen dataset and 400 amino acids in the antioxidant dataset.

Based on this analysis and in line with the literature, a crucial parameter was defined for both the *OmniA* pipeline and Propythia: the maximum allowed sequence length. This parameter is essential for tasks like padding or truncating sequences, and the limit was set at 600 amino acids. This threshold was chosen because 600 amino-acid length strikes a balance between computational efficiency and the retention of meaningful information.

While this limit preserves most sequences without truncation and ensures efficient processing, the issue of padding arises, particularly because many sequences are significantly shorter than 600 amino acids. This is especially relevant for the antioxidant dataset, where a limit of 400 amino acids might have been sufficient. Excessive padding could introduce noise into the models, a concern that will be addressed in more detail in later sections.

5.1.2 Labels

Another crucial aspect to consider is the distribution of labels, particularly in the context of binary classification. It is essential to assess whether the dataset is balanced, and if not, to understand the class proportions, identifying the majority and minority classes. Figure 26 provides a circular graph that shows these proportions, offering an intuitive view that is valuable for subsequent stages of analysis.

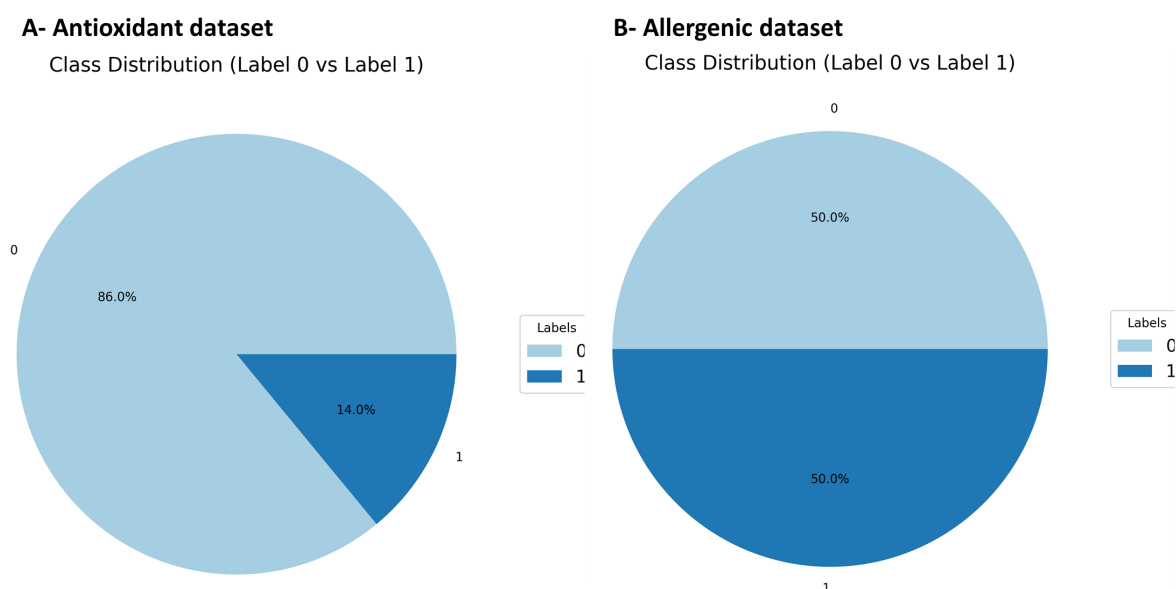


Figure 26: Class distribution for (A) the antioxidant dataset and (B) the allergen dataset. The allergen dataset shows balanced classes, while the antioxidant dataset displays a significant class imbalance, with the positive class (antioxidant properties) being the minority.

The label distribution reveals two distinct scenarios: in the allergen dataset (Figure 26-B), the classes are balanced, while in the antioxidant dataset (Figure 26-A), there is a significant imbalance, with a ratio of approximately 1:6. In this case, the positive class (class 1), which represents antioxidant properties, is the minority. This imbalance will necessitate specialized strategies during model training to ensure robust performance across both classes.

5.1.3 Distribution of sequence length by label

An essential aspect to consider when processing protein sequences is the sequence length distribution associated with each label, as shown in Figure 27. This analysis aims to assess whether the sequence lengths are balanced across the labels in the case studies. An imbalance in sequence lengths can introduce challenges for ML models, particularly due to the effects of padding and truncation, which can influence model performance.

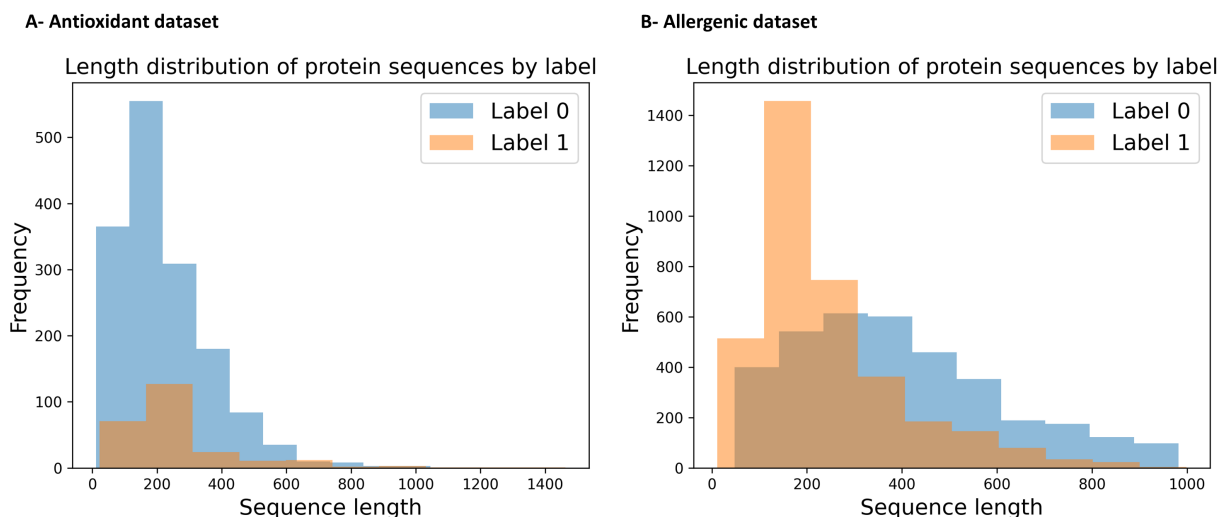


Figure 27: Sequence length distribution for (A) the antioxidant dataset and (B) the allergen dataset, categorized by class label. Both panels reveal an imbalance in sequence lengths across the labels.

In both case studies, no explicit effort was made to balance sequence lengths between the labels. In the allergenic dataset, represented in Figure 27-B, this issue becomes more pronounced. With a 600-amino-acid threshold, most label 1 sequences require padding, while label 0 sequences face a mix of padding and truncation. This imbalance leads to label 1 sequences containing significant amounts of padding, potentially introducing considerable noise. Meanwhile, label 0 sequences undergo both padding and truncation, resulting in more heterogeneous processing. This asymmetry could hinder the model's ability to generalize well and affect prediction accuracy.

5.2 OmniA

The *OmniA* platform utilizes Optuna for hyperparameter tuning and pipeline selection, optimizing the combination of each model's architecture and feature representation or encoding method to enhance performance across 100 trials.

The models were evaluated using a range of performance metrics, including accuracy, recall, F1-score, and either Matthews Correlation Coefficient (MCC) or precision. These specific metrics were chosen based on the literature, where MCC is commonly reported for antioxidant datasets and precision for allergenic datasets, alongside the other referred metrics.

In addition, for the antioxidant dataset, which is imbalanced, the F1-score was selected as the optimization metric because it balances precision and recall, making it more suitable for handling class imbalances. Conversely, in the allergenic case study, where the dataset is balanced, accuracy was chosen as the primary optimization metric, as it provides a straightforward measure of overall performance when the classes are evenly distributed.

The following results highlight the optimized pipelines along with the final evaluation metrics obtained from testing the models on unseen data, as well as the optimization metric derived from training the models.

5.2.1 Antioxidant case study

In this study, two evaluation approaches were used for the antioxidant case study: the hold-out approach and cross-validation.

In the hold-out approach, the dataset was split into a training set, a validation set, and a test set. The model was trained only once using the training set, validated on the validation set, and then evaluated on the test set, which remained unseen throughout the training and validation phases.

In contrast, the cross-validation approach involved splitting the training data into multiple folds (k -folds= 5), where the model was trained and validated iteratively across different subsets of the data. This provided a more robust estimate of the model's performance by averaging the results across the folds. After cross-validation, the best-performing models were then tested on a separate test set, to assess their final performance.

Thus, Table 8 presents the metrics obtained from the test set for both the hold-out and cross-validation approaches. It also includes the optimization metric used during the training and validation phases.

Table 8: Top 4 pipelines selected for the antioxidant case study, with and without cross-validation. Metrics presented include: accuracy, recall, Matthews correlation coefficient, F1-score in test set and optimization F1_score in the validation set (hold-out) or calculated from a cross-validation process.

Pipeline Configuration	Optimization F1_score	Accuracy	Recall	Matthews Correlation Coefficient	F1-score
Hold-out					
Protein descriptor + Tabular Predictor	1.000	0.893	0.342	0.465	0.473
ESM (pretrained_model="35M" and preset="features") + Tabular Predictor	0.842	0.923	0.500	0.638	0.644
Protein descriptor + MLPClassifier	0.612	0.882	0.421	0.446	0.500
Protein descriptor + GBoosting	0.609	0.908	0.342	0.556	0.510
With cross validation					
ESM (pretrained_model="35M" and preset="features") + Tabular Predictor	0.690	0.930	0.632	0.685	0.716
ESM (pretrained_model="8M" and preset="representations") + GBClassifier	0.662	0.911	0.500	0.525	0.613
Protein descriptor + Tabular Predictor	0.552	0.900	0.395	0.496	0.517
ESM (pretrained_model="35M" and preset="features") + GBClassifier	0.488	0.915	0.421	0.597	0.582

These results highlight the importance of implementing cross-validation to prevent overfitting in small datasets with unbalanced label distributions. Initially, the best optimization score achieved on the validation set, applying hold-out, was 1.00. Although this score might suggest perfect performance, it actually reflects overfitting to the validation. Additionally, running a large number of trials—especially with Optuna’s TPE sampler, which builds on information from previous trials—probably intensified the overfitting issue.

As evidence of this overfitting, the model’s performance dropped significantly when evaluated on the test set, indicating that it had learned to fit the validation set rather than generalizing to unseen data. This underscores the necessity of employing more robust validation techniques, such as cross-validation, to obtain a more reliable measure of a model’s ability to generalize.

The implementation of cross-validation was therefore fundamental in correcting this behavior, balancing the model's fit to the data and allowing a more accurate assessment of its generalization capacity. After applying cross-validation, the models with the best optimization scores were also those that obtained the best metrics in the test set, which suggests that the model is actually learning patterns and not just memorizing specific examples.

Finally, considering the overfitting issue, we decided to select the best pipeline based on cross-validation approach, using the highest optimization score as a reference. The pipeline that performed best after cross-validation was the one using ESM (*pretrained_model*= "35M" and *preset*="features") combined with a Tabular Predictor. It achieved an accuracy of 0.93, a recall of 0.63, an MCC of 0.69, and an F1-score of 0.72 on the test set. Although these metrics are not exceptionally high, this pipeline outperformed the alternatives, demonstrating its robustness in addressing the dataset's challenges. The complexity in this case stems primarily from the limited amount of data and its imbalance, which made it difficult for most models to generalize effectively. Despite these challenges, the selected pipeline consistently delivered the best performance, underscoring its resilience under these conditions.

5.2.2 Allergenic case study

Table 9 shows the results for the top four pipelines obtained with *OmniA* for the allergenic dataset. In this case, we did not implement the cross-validation technique but instead employed the hold-out approach, splitting the dataset into training, validation, and test sets. The dataset was sufficiently large to ensure robust performance without the need for cross-validation. Additionally, the results from both the validation and evaluation stages were consistent and similar, suggesting that the absence of cross-validation did not compromise the reliability of the selected pipelines. Although cross-validation could have provided even more robust results, it would have significantly increased computational time, which was not ideal for this analysis.

Table 9: Top 4 pipelines selected for the allergenic case study. Metrics presented include: accuracy, precision, recall, F1-score on test set and optimization accuracy as validation score.

Pipeline	Optimization Accuracy	Accuracy	Precision	Recall	F1-score
ESM (pretrained_model="8M" and preset="representations") + Tabular Predictor	1.000	0.941	0.940	0.942	0.941
Protein descriptor + Tabular Predictor	1.000	0.926	0.948	0.901	0.924
ESM (pretrained_model="8M" and preset="representations") + AutoEncoderMLP	0.928	0.927	0.942	0.911	0.926
ESM (pretrained_model="8M" and preset="representations") + CNN1D	0.916	0.915	0.936	0.890	0.913

Another interesting aspect to mention is that the DL models, such as RNN, were also not selected as the best options for this case study. However, we do not believe that this is due to the dataset size, as it is sufficiently large. Instead, the selected models demonstrated such robust performance that other pipeline configurations, including DL models, were not prioritized.

It is immediately clear that, among the top 4 configurations, the one using ESM (*pretrained_model="8M"* and *preset="representations"*) is the most recurrent, underscoring that the choice of ESM with this specific setup was key to optimizing the model's performance. Despite having two pipelines with the highest optimization score, we ultimately selected this encoding combined with the ML Tabular Predictor model as the best pipeline. This configuration achieved an accuracy, precision, recall, and F1-score of 0.94 on the test set.

5.3 Propythia

The best hyperparameters for each model were selected using 5-fold cross-validation with Propythia, employing the same data split and performance metrics as the *OmniA* pipelines in both case studies. Once the optimal hyperparameters were determined, the models were evaluated on the unseen test set.

5.3.1 Antioxidant case study

The results and the best-optimized hyperparameters for each model of Propytha for the antioxidant case study are presented in Table 10, which specifies the type of numerical data representation (feature extraction or encoding technique) utilized for each model, providing essential context for interpreting the results effectively. In this case, the F1-score was used as the optimization metric to obtain the best hyperparameters, due to the imbalanced nature of the dataset.

Table 10: Optimized hyperparameters and performance metrics (accuracy, recall, MCC, F1-score) on test set for each model applied to the antioxidant case study. The table also specifies the type of numerical data representation used for the protein sequences.

Note: Hyperparameters not explicitly mentioned are set to their default values as specified in the scikit-learn library.

Models	Numerical data representation	Accuracy	Recall	Matthews Correlation Coefficient	F1-score
Linear-SVC(C=0.01, class_weight='balanced')	Protein descriptors	0.911	0.970	0.596	0.636
RandomForest(n_estimators=10)	Protein descriptors	0.782	0.184	0.06	0.192
KNN(leaf_size=15, n_neighbors=2, weights='distance')	Protein descriptors	0.889	0.342	0.447	0.464
GradientBoosting(max_features=0.6, n_estimators=500)	Protein descriptors	0.668	0.763	0.294	0.392
LogisticRegression(C=0.01, class_weight='balanced', solver='liblinear')	Protein descriptors	0.882	0.526	0.489	0.555

Neural networks(activation='tanh', alpha=0.001, hidden_layer_sizes=(50,), learning_rate_init=0.01, solver='sgd')	Protein descriptors	0.911	0.395	0.575	0.555
SGD (alpha=0.001, class_weight='balanced', early_stopping=True, loss='modified_huber')	Protein descriptors	0.860	0.632	0.481	0.558
SVC(C=0.01, class_weight='balanced', gamma='auto', kernel='linear')	Protein descriptors	0.900	0.473	0.532	0.571
AutoEncoder(num_dense_unit_encoder=128, num_dense_layers=2, latent_dim=6, first_hidden_classifier_unit=20, num_dense_layers_classifier=3, dropout=0.1)	Protein descriptors	0.889	0.211	0.432	0.348
RNN(RNN_type=LSTM, bidirectional=False, n_rnn_layers=2, hidden_units_rnn_layers=64, num_dense_layers=2, hidden_units_dense_layers=64, dropout=0.3)	ESM (150 M)	0.860	0.000	0.000	0.000

RNN(RNN_type=GRU, bidirectional=False, n_rnn_layers=3, hidden_units_rnn_layers=128, num_dense_layers=3, hidden_units_dense_layers=64, dropout=0.1)	NLFEncoder	0.857	0.000	0.000	0.000
---	------------	-------	-------	-------	-------

Based on the metrics presented, with a focus on F1-score and MCC due to the imbalanced nature of the dataset, it is evident that the shallow ML models outperformed the DL models. This outcome is expected, as DL models typically excel with larger datasets, where they can leverage their ability to learn complex patterns. In contrast, simpler ML models are better suited for smaller datasets like the one analyzed, due to their reduced risk of overfitting and stronger generalization capacity.

Among the tested models, Linear-SVC achieved the best performance with an F1-score close to 0.64 and an MCC around 0.60. This result, however, does not imply that the data is perfectly linearly separable. Although Linear-SVC aims to separate classes through a linear hyperplane, its effectiveness, in this case, can be attributed to regularization (with an l2 penalty) and hyperparameter adjustments, such as automatic class balancing. In scikit-learn, the "balanced" mode automatically adjusts class weights inversely proportional to their frequencies, ensuring that underrepresented classes have a higher weight during training, which helps mitigate the effects of class imbalance and improves model performance on minority classes.

Even with moderate complexity, Linear-SVC can perform well, though perfect linear separation is not guaranteed with the available data. The model's success highlights the importance of fine-tuning key hyperparameters. Additionally, optimizing other parameters like tolerance for stopping criteria (*tol*) and maximum iterations (*max_iter*) could further improve performance by enhancing convergence and preventing premature stopping, potentially leading to more accurate and stable results.

5.3.2 Allergenic case study

The results for the models created with Propythia for the allergenic dataset are shown in Table 11, detailing the best-optimized hyperparameters and the numerical data representation selected for each model. In this case, the accuracy score was used as the optimization metric to obtain the best hyperparameters.

Table 11: Optimized hyperparameters and performance metrics (accuracy, recall, precision, F1-score) on test set for each model applied to the allergenic case study. The table also specifies the type of numerical data representation used for the protein sequences.

Note: Hyperparameters not explicitly mentioned are set to their default values as specified in the scikit-learn library.

Models	Numerical data representation	Accuracy	Precision	Recall	F1-score
Linear-SVC(C=0.01)	Protein descriptors	0.897	0.949	0.839	0.891
RandomForest(n_estimators=500)	Protein descriptors	0.700	0.646	0.886	0.747
KNN(leaf_size=15, n_neighbors=2, weights='distance')	Protein descriptors	0.641	0.586	0.956	0.723
GradientBoosting(max_features=0.6, n_estimators=500, max_depth=5)	Protein descriptors	0.802	0.933	0.651	0.767
LogisticRegression(C=0.01, solver='sag')	Protein descriptors	0.893	0.956	0.823	0.885
Neural networks(activation='logistic', alpha=0.00001, hidden_layer_sizes=(50,), learning_rate_init=0.0001)	Protein descriptors	0.894	0.953	0.828	0.886
SGD (alpha=0.001, class_weight='balanced', early_stopping=True, loss='modified_huber', n_iter_no_change=10)	Protein descriptors	0.891	0.962	0.814	0.882

SVC(C=10)	Protein descriptors	0.895	0.950	0.834	0.888
AutoEncoder(num_dense_unit_encoder=64, num_dense_layers=2, latent_dim=6, first_hidden_classifier_unit=20, num_dense_layers_classifier=2, dropout=0.1)	Protein descriptors	0.111	0.145	0.159	0.152
RNN(RNN_type=GRU, bidirectional=True, n_rnn_layers=2, hidden_units_rnn_layers=128, num_dense_layers=3, hidden_units_dense_layers=64, dropout=0.1)	NLFEncoder	0.842	0.861	0.817	0.838
RNN(RNN_type=GRU, bidirectional=True, n_rnn_layers=2, hidden_units_rnn_layers=256, num_dense_layers=2, hidden_units_dense_layers=128, dropout=0.3)	ESM (150 M)	0.939	0.939	0.939	0.939

For the balanced dataset of allergens, accuracy becomes a more reliable metric. In this context, the RNN model, utilizing features from the ESM encoding with a "150M" pre-trained model, achieved the highest performance. It obtained an accuracy, precision, recall and F1-score close to 0.94 demonstrating an excellent ability to correctly classify both classes. This result aligns with the general understanding that DL models, like RNN, tend to have good performance with larger datasets, as they can capture more complex patterns and adjust their parameters effectively.

Interestingly, while the RNN model paired with the ESM encoder delivered the best performance, it was significantly worse when combined with the NLF encoder. Furthermore, utilizing protein descriptors, the Linear-SVC model outperformed the RNN with the NLF encoder. This highlights that it is not correct to draw definitive conclusions from a single result without considering the interaction between the model and the chosen numerical data representation.

The optimal choice is not determined solely by the model's architecture or the feature representation, but by the synergy between both. This highlights the value of AutoML approaches, which systematically explore various combinations of models and numerical data representations in a flexible and automated manner. By doing so, it adapts to the unique characteristics of each dataset without requiring extensive manual input from the user, ensuring a tailored and effective solution.

5.4 Comparison with different studies

Finally, the performance of *OmniA* and Propythia models was compared to the performance of published tools using the same datasets and metrics as before.

5.4.1 Antioxidant case study

Figure 28 shows the results obtained for the antioxidant dataset, revealing that, although the *OmniA*'s AutoML platform demonstrated competitive performance, it did not achieve the highest results in the F1-score and MCC metrics, which are particularly well-suited for evaluating unbalanced datasets.

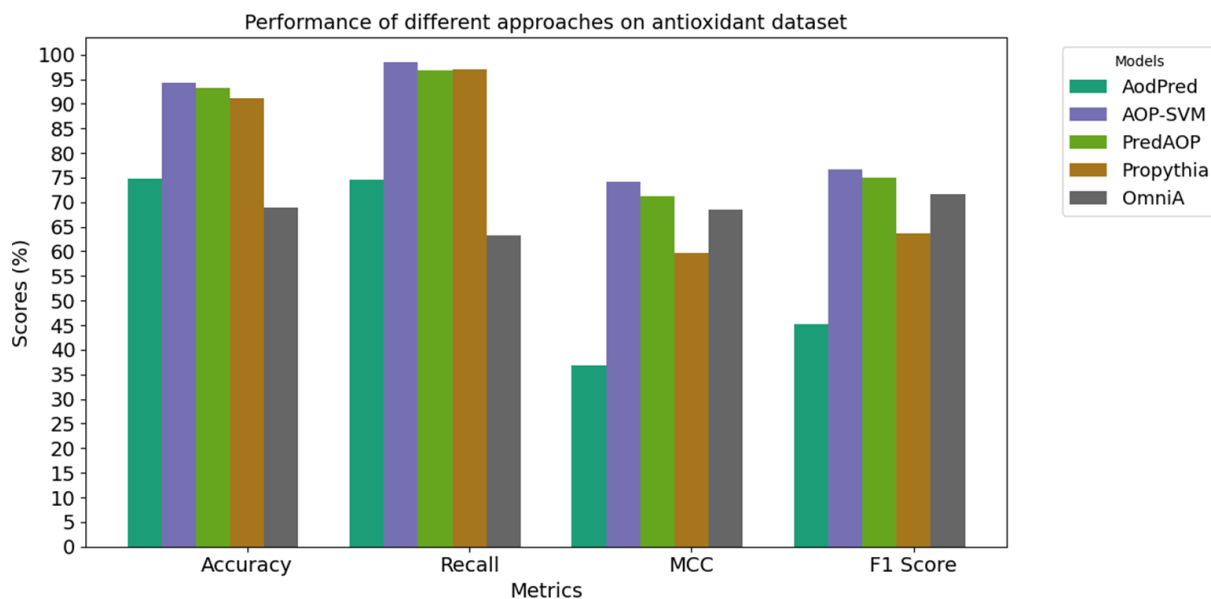


Figure 28: Bar chart comparing the performance metrics (Accuracy, Recall, MCC, and F1-score) of various approaches ('AodPred', 'AOP-SVM', 'PredAOP', 'Propythia', and 'OmniA') in the antioxidant case study, with scores presented as percentages.

When comparing *OmniA* to the AOP-SVM and PredAOP approaches, both showed very similar results. However, *OmniA* did not achieve the best performance in any of the evaluated metrics. A notable distinction is that these alternative approaches employ the leave-one-out validation technique, where the model is trained on all but one sample, which is then used for testing. This process is repeated until every sample has been used as a test case [118, 119].

The leave-one-out approach has the advantage of making maximum use of the data set, which is particularly advantageous for small data sets. However, it also has notable drawbacks. In particular, its execution time is considerably longer than that of the k-fold cross-validation used by *OmniA*, which makes it more computationally expensive.

Initially, it was expected that the approach by Meng et al., from which our dataset was obtained, would be included for comparison in Figure 28. However, according to the authors, this method uses an oversampling technique to address the imbalance in the dataset [120]. A critical point to note is that oversampling was applied not only to the training data but also to the test set. As a result, it did not make sense to include this method for comparison with the others, since its test set would differ from ours due to the oversampling. Therefore, it was excluded from the comparative analysis.

Oversampling the entire dataset introduces artificial samples into the evaluation process, which can distort performance metrics. This leads to an inflated sense of the model's generalizability, as the artificial samples may not reflect real-world data, creating a false impression of robustness. Ideally, oversampling should only be applied to the training set to ensure the test set consists solely of real data, providing an accurate assessment of the model's performance. This methodological oversight may explain part of DP-AOP's superior performance, but it also raises concerns about the real-world applicability of its results.

In datasets with a pronounced class imbalance, such as the 1:6 ratio in our case, a high accuracy score may simply reflect the model's ability to predict the majority class (true negatives) correctly, while overlooking the minority class (false negatives), which is undesirable. This gives a misleading sense of good performance, as the model may fail to identify the positive class effectively.

Furthermore, it is important to highlight that when examining metrics such as accuracy, *OmniA* showed the weakest performance. However, this can be explained by the fact that *OmniA* was specifically optimized for the F1-score, which is a more appropriate metric for imbalanced datasets. As a result, its lower accuracy compared to other approaches is expected, as the model prioritizes balancing precision and recall rather than focusing on overall accuracy.

In this context, *OmniA* stood out by offering a more effective balance between stability and computational efficiency, presenting a more robust and consistent methodological approach. By employing rigorous validation techniques and avoiding procedures that could compromise the integrity of the test set, such as oversampling during evaluation, this AutoML platform produced more reliable results. Consequently, *OmniA* demonstrated greater robustness in the scenarios analyzed, highlighting its potential as a tool for handling unbalanced datasets in a methodologically sound manner.

5.4.2 Allergenic case study

In the case of allergens (Figure 29), three approaches, *OmniA*, Propythia, and AllerCatPro, achieve robust and similar performance metrics, with *OmniA* showing a slightly higher performance, although only by a small margin. However, the AllerCatPro approach is distinct from the methods explored in this project and is therefore not referenced in subsection 2.5.3. Unlike the ML and DL models employed in this work, AllerCatPro uses a k-mer hit principle and 3D epitope similarity to predict allergens.

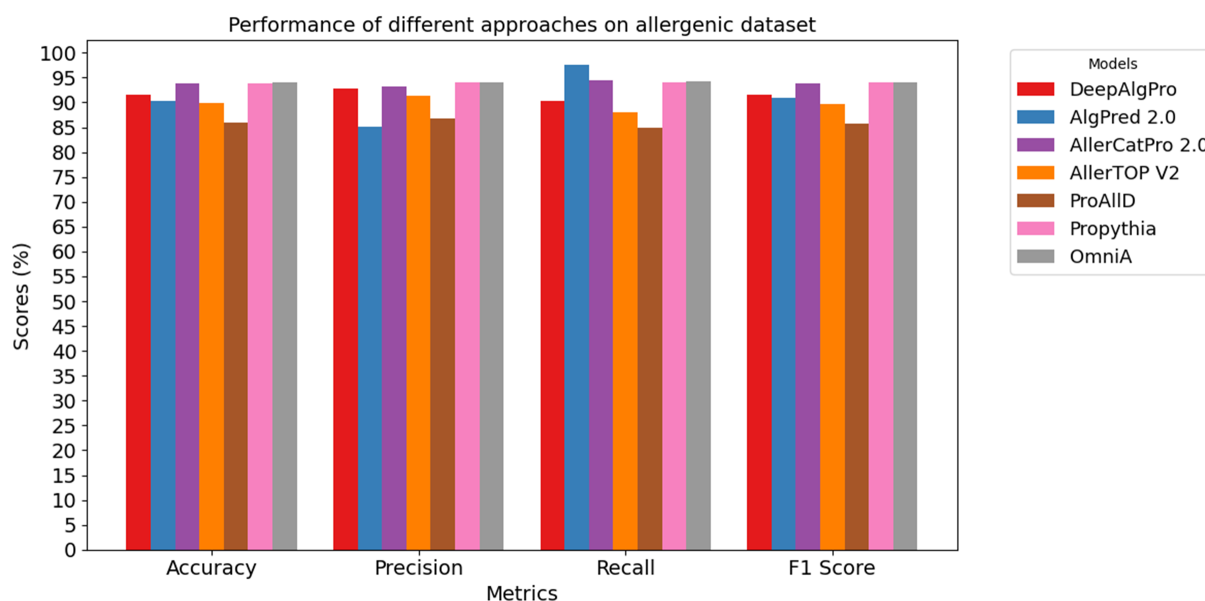


Figure 29: Bar chart comparing the performance metrics (accuracy, precision, recall, and F1-score) of various approaches ('DeepAlgPro', 'AlgPred 2.0', 'AllerCatPro 2.0', 'AllerTOP V2', 'ProAllD', 'Propythia', 'OmniA') in the allergenic case study, with scores presented as percentages.

Algpred 2.0 stands out as the only approach where the recall score (97.61%) is significantly higher than the other performance metrics, indicating that the model is better at identifying the positive class compared to its overall accuracy or precision.

While a high recall is generally advantageous, it is important to consider this metric in relation to others, such as the precision of 85%. The relatively lower precision suggests that although the model is highly effective at identifying positive cases, it also misclassifies a significant number of negative cases as positives. In the context of allergen prediction, this may not be overly problematic, as the model tends to classify proteins as allergenic even when they are not. This can be acceptable in terms of food safety, ensuring that potential allergens are not missed. However, this conservative approach could hinder innovation. Proteins that are flagged as allergens but are in fact non-allergenic might be overlooked as viable alternatives for use in food products. These proteins could offer similar characteristics to allergenic ones but without triggering allergic reactions, potentially offering safe and innovative solutions for the food industry.

In the context of allergen prediction, this may not be overly problematic, as the model tends to classify proteins as allergenic even when they are not. While this cautious approach ensures that potential allergens are less likely to be missed, which is important for food safety, it may also limit innovation. Proteins incorrectly flagged as allergens could be disregarded as potential alternatives for use in food products, even though they are non-allergenic.

Thus, the ideal approach in a balanced dataset like the one used here should aim for a more even distribution of high scores across all metrics. This supports both safety and innovation, enabling the identification of non-allergenic proteins that still meet the desired product characteristics.

In short, the *OmniA* platform developed stood out in this case study by presenting more robust and balanced metrics, offering significant advantages as an AutoML solution. Unlike manual approaches such as Propytha, which require individual model optimization, *OmniA* automates the process, making it more efficient and accessible to both experts and users without technical knowledge. In addition, the platform offers flexibility through different presets, and the use of more advanced configurations, such as “medium” or “heavy”, could even improve the performance obtained, as long as the cost-benefit in terms of consumption of computing resources to guarantee the performance increase is justified.

Chapter 6

Conclusions and future work

This final chapter summarizes the main results of the project, highlighting the main developments, conclusions and contributions of the work. It also explores potential future directions for improving the performance of the *OmniA* platform and expanding its applicability, not only at the protein level, but also in broader contexts.

6.1 Conclusions

This project involved the extension of the *OmniA* AutoML platform, with a focus on backend development to address protein classification challenges based on amino acid sequences. The platform was applied and validated for predicting antioxidant and allergenic properties.

As part of this initiative, five new protein descriptors were implemented in the feature extraction module of the *Proteins* sub-package. These descriptors include GAAC, CKSAAP, CT, and DDE. Additionally, three specific encoders— NLF, ESM, and ProtBert— were developed within the encoder module, ensuring compatibility with padding and truncation for sequences up to 600 amino acids.

To explore different approaches for representing protein sequences, either through protein descriptors or encoders, four deep learning models were developed: an Autoencoder, a CNN1D for tabular data, an RNN, and a hybrid CNN1D-RNN for matrix data. These models were implemented in the *Generics* sub-package, ensuring their applicability extends beyond protein classification problems.

The implementation of presets was crucial for the overall functionality of the *OmniA* pipeline. For this project, presets were developed for both the Protein and *Generics* sub-packages, offering three levels of complexity: light, medium, and heavy. The analysis primarily focused on the light preset, which, while not optimizing model hyperparameters, effectively enhanced the selection of numerical data representations— specifically protein descriptors, or ESM, or NLF— alongside their parameters and the appropriate ML and DL models.

To further improve model robustness, cross-validation techniques were employed during pipeline optimization to mitigate overfitting issues, particularly those arising from the use of unbalanced datasets, as seen in the antioxidant case study.

The aforementioned development made it possible to run the *OmniA* platform's global pipeline with the light preset, allowing benchmarking against other approaches, based on the same data sets selected for each case study.

In the case of the study on antioxidant properties, *OmniA*'s results were compared to those obtained with literature approaches such as DP-AOP, AodPred, AOP-SVM, PredAOP and Propythia. The best-performing pipeline on the *OmniA* platform was obtained after cross-validation with the ESM model (pretrained_model="35M" and preset="features"), combined with a tabular predictor. This configuration achieved an accuracy of 0.69, recall of 0.63, MCC of 0.69 and F1-score of 0.72. Although these results did not surpass the performance of other approaches, they offer greater confidence due to the techniques and procedures performed, which preserved the integrity of the test set. In addition, this approach presented an adequate theoretical balance in terms of computational efficiency, avoiding methods such as leave-one-out in favour of cross-validation.

As for the study of allergenic properties, *OmniA* was compared to tools such as DeepAlgPro, AllgPred 2.0, AllerCatPro 2.0, AllerTOP V2, ProAllID and Propythia. For *OmniA*, the ESM configuration (pretrained_model="8M" and preset="representations"), in conjunction with a tabular predictor, obtained impressive results, with accuracy, recall, precision, and F1-score around 0.94. In this context, *OmniA* stood out with a more robust performance compared to the other approaches, as did Propythia, another tool developed at the University of Minho in the Bioinformatics and Systems Biology research group.

That said, it is fair to conclude that the development of *OmniA* has enabled a more effective balance between stability and computational efficiency for both case studies explored, presenting a robust and consistent methodological approach. Furthermore, as an AutoML platform, *OmniA* offers significant advantages, as it can be used by both users with advanced knowledge and those with no experience in the field. This makes it an automatic and flexible process, capable of competing with more traditional approaches that rely on manual trial and error, such as Propythia.

6.2 Prospect for future work

During the course of the work, some aspects were identified that could be improved or implemented. The main suggestions include:

- Exhaustive research: More in-depth research, covering all relevant databases and available literature, could increase the quality and diversity of the datasets used in the case studies. This would result in more robust and generalizable models.
- Implementing sequence similarity filtering: Incorporating the ability to detect and filter out highly similar sequences within the dataset is crucial to avoid redundancy and ensure that the model is not trained on overlapping information.
- Implementation of oversampling and undersampling techniques: To deal with unbalanced datasets, the inclusion of oversampling and undersampling tools would help improve model performance by balancing out minority classes and avoiding bias in the results.
- Explainability of Models: Increasing the transparency of the models developed is fundamental to ensuring that users can understand and trust the predictions made by the platform.
- Implementation of more feature selection and data normalization methods: Adding a variety of feature selection and data normalization methods can improve model performance, particularly when dealing with datasets that exhibit specific characteristics.
- Analysis of results with medium and heavy presets: Exploring the results obtained with more complex presets can provide additional insights into the performance of models in different scenarios and would help assess the consumption of computing resources required for these more advanced presets.
- Exploration of 3D Features and spatial conformation of proteins: The exploration of tools that consider the 3D structure of proteins and their spatial conformations can bring a new dimension to protein analysis. Models that integrate 3D representations can improve the accuracy of predictions, capturing crucial information about structural interactions that are not evident from linear amino acid sequences alone.

- Integration of Graph-based models: The implementation of models using graphs could significantly enrich data analysis. This type of approach is especially useful in cases where there are complex intrinsic relationships between elements, such as interactions between protein residues. Graph models can capture these interconnections, providing a more in-depth view and revealing patterns that traditional methods may not detect.
- Development of a graphical interface: The creation of a graphical interface for the *OmniA* platform would be an important step forward, making it more accessible to a wide range of users regardless of their technical level. This would allow non-expert users to use the platform intuitively, without the need to run the pipeline through code, significantly extending its reach.

Bibliography

- [1] Solveig Badillo, Balazs Banfai, Fabian Birzele, Iakov I Davydov, Lucy Hutchinson, Tony Kam-Thong, Juliane Siebourg-Polster, Bernhard Steiert, and Jitao David Zhang. An Introduction to Machine Learning. *Clinical Pharmacology & Therapeutics*, 107(4):871–885, 2020.
- [2] Peter W Cook and Kendra K Nightingale. Use of omics methods for the advancement of food quality and food safety. *Animal Frontiers*, 8(4):33–41, 2018.
- [3] Travers Ching, Daniel S Himmelstein, Brett K Beaulieu-Jones, Alexandr A Kalinin, Brian T Do, Gregory P Way, Enrico Ferrero, Paul-Michael Agapow, Michael Zietz, Michael M Hoffman, Wei Xie, Gail L Rosen, Benjamin J Lengerich, Johnny Israeli, Jack Lanchantin, Stephen Woloszynek, Anne E Carpenter, Avanti Shrikumar, Jinbo Xu, Evan M Cofer, Christopher A Lavender, Srinivas C Turaga, Amr M Alexandari, Zhiyong Lu, David J Harris, Dave DeCaprio, Yanjun Qi, Anshul Kundaje, Yifan Peng, Laura K Wiley, Marwin H S Segler, Simina M Boca, S Joshua Swamidass, Austin Huang, Anthony Gitter, and Casey S Greene. Opportunities and obstacles for deep learning in biology and medicine. *Journal of The Royal Society Interface*, 15(141):20170387, 2018.
- [4] Ashfaq Ahmad and Syed Salman Ashraf. Sustainable food and feed sources from microalgae: Food security and the circular bioeconomy. *Algal Research*, 74:103185, 2023.
- [5] Yufeng Jane Tseng, Pei-Jiun Chuang, and Michael Appell. When Machine Learning and Deep Learning Come to the Big Data in Food Chemistry. *ACS Omega*, 8(18):15854–15864, may 2023.
- [6] Ayten Aylin Alsaffar. Sustainable diets: The interaction between food industry, nutrition, health and the environment. *Food Science and Technology International*, 22(2):102–111, 2016.
- [7] OmniumAI. OmniumAI-OmniA.
- [8] Emiko Fukase and Will Martin. Economic growth, convergence, and world food demand and supply. *World Development*, 132:104954, 2020.

- [9] H Charles J Godfray, John R Beddington, Ian R Crute, Lawrence Haddad, David Lawrence, James F Muir, Jules Pretty, Sherman Robinson, Sandy M Thomas, and Camilla Toulmin. Food Security: The Challenge of Feeding 9 Billion People. *Science*, 327(5967):812–818, 2010.
- [10] Sara N Garcia, Bennie I Osburn, and Michele T Jay-Russell. One Health for Food Safety, Food Security, and Sustainable Food Production. *Frontiers in Sustainable Food Systems*, 4, 2020.
- [11] Alexandre Meybeck and Vincent Gitz. Sustainable diets within sustainable food systems. *Proceedings of the Nutrition Society*, 76(1):1–11, 2017.
- [12] Abdo Hassoun, Abderrahmane Ait-Kaddour, Adnan M Abu-Mahfouz, Nikheel Bhojraj Rathod, Farah Bader, Francisco J Barba, Alessandra Biancolillo, Janna Cropotova, Charis M Galanakis, Anet Režek Jambrak, José M Lorenzo, Ingrid Måge, Fatih Ozogul, and Joe Regenstein. The fourth industrial revolution in the food industry-Part I: Industry 4.0 technologies. *Critical reviews in food science and nutrition*, 63(23):6547–6563, 2023.
- [13] Abdo Hassoun, Alaa El-Din Bekhit, Anet Režek Jambrak, Joe M Regenstein, Farid Chemat, James D Morton, María Gudjónsdóttir, María Carpena, Miguel A Prieto, Paula Varela, Rai Naveed Arshad, Rana Muhammad Aadil, Zuhaib Bhat, and Øydis Ueland. The fourth industrial revolution in the food industry-part II: Emerging food trends. *Critical reviews in food science and nutrition*, pages 1–31, aug 2022.
- [14] Celso F Balthazar, Jonas F Guimarães, Nathália M Coutinho, Tatiana C Pimentel, C Senaka Ranadheera, Antonella Santillo, Marzia Albenzio, Adriano G Cruz, and Anderson S Sant’Ana. The future of functional food: Emerging technologies application on prebiotics, probiotics and postbiotics. *Comprehensive reviews in food science and food safety*, 21(3):2560–2586, may 2022.
- [15] Dominika Teigiserova, L Hamelin, and Marianne Thomsen. Towards transparent valorization of food surplus, waste and loss: Clarifying definitions, food waste hierarchy, and role in the circular economy. *Science of The Total Environment*, 706:136033, 2019.
- [16] Abdo Hassoun, Miguel A Prieto, María Carpena, Yamine Bouzembrak, Hans J P Marvin, Noelia Palarés, Francisco J Barba, Sneh Punia Bangar, Vandana Chaudhary, Salam Ibrahim, and Gioacchino Bono. Exploring the role of green and Industry 4.0 technologies in achieving sustainable development goals in food sectors. *Food Research International*, 162:112068, 2022.

- [17] Izza Faiz Ul Rasool, Afifa Aziz, Waseem Khalid, Hyrije Koraqi, Shahida Anusha Siddiqui, Ammar Al-Farga, Wing-Fu Lai, and Anwar Ali. Industrial Application and Health Prospective of Fig (*Ficus carica*) By-Products. *Molecules (Basel, Switzerland)*, 28(3), jan 2023.
- [18] Tugce Akyazi, Aitor Goti, Aitor Oyarbide, Elisabete Alberdi, and Felix Bayon. A Guide for the Food Industry to Meet the Future Skills Requirements Emerging with Industry 4.0. *Foods (Basel, Switzerland)*, 9(4), apr 2020.
- [19] David L Nelson and Michael M Cox. *Lehninger Principles of Biochemistry, Eighth Edition*. Eighth edition edition, 2021.
- [20] J E Murray, N Laurieri, and R Delgoda. Chapter 24 - Proteins. In Simone Badal and Rupika Delgoda, editors, *Pharmacognosy*, pages 477–494. Academic Press, Boston, 2017.
- [21] Ricardo N Pereira and Rui M Rodrigues. Emergent Proteins-Based Structures—Prospects towards Sustainable Nutrition and Functionality. *Gels*, 7(4), 2021.
- [22] Miguel Lima, Rui Costa, Ivo Rodrigues, Jorge Lameiras, and Goreti Botelho. A Narrative Review of Alternative Protein Sources: Highlights on Meat, Fish, Egg and Dairy Analogues. *Foods (Basel, Switzerland)*, 11(14), jul 2022.
- [23] Paul Wood and Mahya Tavan. A review of the alternative protein industry. *Current Opinion in Food Science*, 47:100869, 2022.
- [24] R M Schultz. Proteins and Protein Structure. In *Reference Module in Life Sciences*. Elsevier, 2017.
- [25] L H Fasolin, R N Pereira, A C Pinheiro, J T Martins, C C P Andrade, O L Ramos, and A A Vicente. Emergent food proteins - Towards sustainability, health and innovation. *Food research international (Ottawa, Ont.)*, 125:108586, nov 2019.
- [26] R. Heiner Schirmer Georg E. Schulz. *Principles of Protein Structure*. Springer New York, NY, first edition, 2013.
- [27] Jan Małeck, Siemowit Muszyński, and Bartosz G Sołowiej. Proteins in Food Systems- Bionanomaterials, Conventional and Unconventional Sources, Functional Properties, and Development Opportunities. *Polymers*, 13(15), jul 2021.
- [28] Rhiannon Morris, Katrina A Black, and Elliott J Stollar. Uncovering protein function: from classification to complexes. *Essays in Biochemistry*, 66(3):255–285, 2022.

- [29] Bhupender Singh, Arun Karnwal, Anurag Tripathi, and Atul Kumar Upadhyay. *Food Allergens and Related Computational Biology Approaches: A Requisite for a Healthy Life*, pages 145–160. Springer Singapore, Singapore, 2021.
- [30] Anai León Madrazo and Maira Rubi Segura Campos. In silico prediction of peptide variants from chia (*S. hispanica* L.) with antimicrobial, antibiofilm, and antioxidant potential. *Computational Biology and Chemistry*, 98:107695, 2022.
- [31] I. D. Nwachukwu and R. E. Aluko. Structural and functional properties of food protein-derived antioxidant peptides. *Journal of Food Biochemistry*, 43:e12761, 2019.
- [32] Chaoting Wen, Jixian Zhang, Haihui Zhang, Yuqing Duan, and Haile Ma. Plant protein-derived antioxidant peptides: Isolation, identification, mechanism of action and application in food systems: A review. *Trends in Food Science & Technology*, 105:308–322, 2020.
- [33] Naijin Xu, Guanqun Chen, and Hui Liu. Antioxidative Categorization of Twenty Amino Acids Based on Experimental Evaluation. *Molecules (Basel, Switzerland)*, 22(12), nov 2017.
- [34] Yongsheng Zhu, Fei Lao, Xin Pan, and Jihong Wu. Food Protein-Derived Antioxidant Peptides: Molecular Mechanism, Stability and Bioavailability. *Biomolecules*, 12(11), 2022.
- [35] M Prado, I Ortea, S Vial, J Rivas, P Calo-Mata, and J Barros-Velázquez. Advanced DNA- and Protein-based Methods for the Detection and Investigation of Food Allergens. *Critical reviews in food science and nutrition*, 56(15):2511–2542, nov 2016.
- [36] Shridhar K Sathe, Suzanne S Teuber, and Kenneth H Roux. Effects of food processing on the stability of food allergens. *Biotechnology advances*, 23(6):423–429, sep 2005.
- [37] Chhavi Arya and Chetna Jantwal. A Review on Identified Major Food Allergens: Characteristics and Role in Food Allergy. *The Indian Journal of Nutrition and Dietetics*, 54:346, 2017.
- [38] Liyang Wang, Dantong Niu, Xinjie Zhao, Xiaoya Wang, Mengzhen Hao, and Huilian Che. A Comparative Analysis of Novel Deep Learning and Ensemble Learning Models to Predict the Allergenicity of Food Proteins. *Foods*, 10(4), 2021.
- [39] Kento Goto, Norimasa Tamehiro, Takumi Yoshida, Hiroyuki Hanada, Takuto Sakuma, Reiko Adachi, Kazunari Kondo, and Ichiro Takeuchi. Novel machine learning method allerStat identifies statistically

- significant allergen-specific patterns in protein sequences. *Journal of Biological Chemistry*, 299(6), jun 2023.
- [40] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass. [u.a.], 2013.
- [41] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.
- [42] François Chollet. *DEEP LEARNING with PYTHON*. Manning Publications Co, New York, 2017.
- [43] Rutvij H Jhaveri, A Revathi, Kadiyala Ramana, Roshani Raut, and Rajesh Kumar Dhanaraj. A Review on Machine Learning Strategies for Real-World Engineering Applications. *Mobile Information Systems*, 2022:1833507, 2022.
- [44] Rene Y Choi, Aaron S Coyner, Jayashree Kalpathy-Cramer, Michael F Chiang, and J Peter Campbell. Introduction to Machine Learning, Neural Networks, and Deep Learning. *Translational vision science & technology*, 9(2):14, feb 2020.
- [45] Emily A Kringle, Evan C Knutson, Collin Engstrom, and Lauren Terhorst. Iterative processes: a review of semi-supervised machine learning in rehabilitation science. *Disability and rehabilitation. Assistive technology*, 15(5):515–520, jul 2020.
- [46] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.
- [47] J Brownlee. *Data Preparation for Machine Learning: Data Cleaning, Feature Selection, and Data Transforms in Python*. Machine Learning Mastery, 2020.
- [48] Myeongsu Kang and Jing Tian. *Machine Learning: Data Pre-processing*, chapter 5, pages 111–130. John Wiley & Sons, Ltd, 2018.
- [49] Andreas C Müller and Sarah Guido. *Introduction to machine learning with Python : a guide for data scientists*. O'Reilly Media, Inc Sebastopol, CA, Sebastopol, CA, first edition, 2017.
- [50] Patricia Garcia-Canadilla, Sergio Sánchez Martínez, Fatima Crispi, and Bart Bijmens. Machine Learning in Fetal Cardiology: What to Expect. *Fetal Diagnosis and Therapy*, 47:1–10, 2020.

- [51] Yilin Wan, Qi Xiong, Zhiwei Qiu, and Yaohan Xie. K-Means Clustering Algorithm Based on Memristive Chaotic System and Sparrow Search Algorithm. *Symmetry*, 14(10), 2022.
- [52] Colleen M Farrelly, Seth J Schwartz, Anna Lisa Amodeo, Daniel J Feaster, Douglas L Steinley, Alan Meca, and Simona Picariello. The analysis of bridging constructs with hierarchical clustering methods: An application to identity. *Journal of Research in Personality*, 70:93–106, 2017.
- [53] Wan-Lei Zhao, Cheng-Hao Deng, and Chong-Wah Ngo. k -means: A revisit. *Neurocomputing*, 291, 2018.
- [54] Hervé Abdi and Lynne J Williams. Principal component analysis. *WIREs Computational Statistics*, 2(4):433–459, 2010.
- [55] Markus Ringnér. What is principal component analysis? *Nature Biotechnology*, 26(3):303–304, 2008.
- [56] Laurens Van Der Maaten. Accelerating T-SNE Using Tree-Based Algorithms. *J. Mach. Learn. Res.*, 15(1):3221–3245, jan 2014.
- [57] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.
- [58] A Burkov. *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019.
- [59] Dastan Maulud and Adnan M Abdulazeez. A Review on Linear Regression Comprehensive in Machine Learning. *Journal of Applied Science and Technology Trends*, 1(4):140–147, 2020.
- [60] Ernest Yeboah Boateng and Daniel A. Abaye. A Review of the Logistic Regression Model with Emphasis on Medical Research. *Journal of Data Analysis and Information Processing*, 07(04):190–207, 2019.
- [61] Hui Zou and Trevor Hastie. Regularization and Variable Selection Via the Elastic Net. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67(2):301–320, 2005.
- [62] Daniel T. Larose and Chantal D. Larose. *kx2010;Nearest Neighbor Algorithm*, pages 149–164. 2014.
- [63] Haiyi Zhang and Di Li. Naïve bayes text classifier. In *2007 IEEE International Conference on Granular Computing (GRC 2007)*, pages 708–708, 2007.

- [64] Jair Cervantes, Farid Garcia-Lamont, Lisbeth Rodríguez-Mazahua, and Asdrubal Lopez. A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing*, 408:189–215, 2020.
- [65] Alka Rani, Nirmal Kumar, Jitendra Kumar, Jitendra Kumar, and Nishant K Sinha. Chapter 6 - Machine learning for soil moisture assessment. In Ramesh Chandra Poonia, Vijander Singh, and Soumya Ranjan Nayak, editors, *Deep Learning for Sustainable Agriculture*, Cognitive Data Science in Sustainable Computing, pages 143–168. Academic Press, 2022.
- [66] Italo Zoppis, Giancarlo Mauri, and Riccardo Dondi. Kernel Methods: Support Vector Machines. In Shoba Ranganathan, Michael Gribskov, Kenta Nakai, and Christian Schönbach, editors, *Encyclopedia of Bioinformatics and Computational Biology*, pages 503–510. Academic Press, Oxford, 2019.
- [67] Raoof Gholami and Nikoo Fakhari. Chapter 27 - Support Vector Machine: Principles, Parameters, and Applications. In Pijush Samui, Sanjiban Sekhar, and Valentina E Balas, editors, *Handbook of Neural Computation*, pages 515–535. Academic Press, 2017.
- [68] Carl Kingsford and Steven L. Salzberg. What are decision trees? *Nature biotechnology*, 26(9):1011–1012, September 2008.
- [69] Yan-Yan Song and Ying Lu. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, 27(2):130–135, apr 2015.
- [70] Ibomoiye Domor Mienye, Yanxia Sun, and Zenghui Wang. Prediction performance of improved decision tree-based algorithms: a review. *Procedia Manufacturing*, 35:698–703, 2019.
- [71] Ammar Mohammed and Rania Kora. A comprehensive review on ensemble deep learning: Opportunities and challenges. *Journal of King Saud University - Computer and Information Sciences*, 35(2):757–774, 2023.
- [72] Yuzhen Zhang, Jingjing Liu, and Wenjuan Shen. A Review of Ensemble Learning Algorithms Used in Remote Sensing Applications. *Applied Sciences*, 12(17), 2022.
- [73] Adele Cutler, D Richard Cutler, and John R Stevens. *Random Forests*, pages 157–175. Springer New York, New York, NY, 2012.

- [74] Seonwoo Min, Byunghan Lee, and Sungroh Yoon. Deep learning in bioinformatics. *Briefings in bioinformatics*, 18(5):851–869, sep 2017.
- [75] Miguel Pérez-Enciso and Laura M Zingaretti. A Guide for Using Deep Learning for Complex Trait Genomic Prediction. *Genes*, 10(7), jul 2019.
- [76] Yann LeCun, Y Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521:436–444, 2015.
- [77] Ajay Shrestha and Ausif Mahmood. Review of Deep Learning Algorithms and Architectures. *IEEE Access*, PP:1, 2019.
- [78] Debasish Kalita. Basics of CNN in Deep Learning, 2023.
- [79] R Mu and X Zeng. A review of deep learning research. *KSII Transactions on Internet and Information Systems*, 13:1738–1764, 2019.
- [80] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.
- [81] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural computation*, 31(7):1235–1270, jul 2019.
- [82] Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. A Review on the Long Short-Term Memory Model. *Artificial Intelligence Review*, 53, 2020.
- [83] Xuan Hien Le, Duc Hai Nguyen, Sungho Jung, and Giha Lee. Deep neural network-based discharge prediction for upstream hydrological stations: a comparative study. *Earth Science Informatics*, 16:1–12, 2023.
- [84] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.
- [85] Yusen Wang, Wenlong Liao, and Yuqing Chang. Gated Recurrent Unit Network-Based Short-Term Photovoltaic Forecasting. *Energies*, 11:2163, 2018.
- [86] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In I Guyon, U Von Luxburg, S Bengio,

- H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
- [87] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Scao, Sylvain Gugger, and Alexander Rush. Transformers: State-of-the-Art Natural Language Processing. pages 38–45, 2020.
- [88] L Tunstall, L von Werra, and T Wolf. *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*. O'Reilly Media, 2022.
- [89] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Stroudsburg, PA, USA, 2019. Association for Computational Linguistics.
- [90] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.
- [91] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics.
- [92] Yusen Wang, Wenlong Liao, and Yuqing Chang. Gated Recurrent Unit Network-Based Short-Term Photovoltaic Forecasting. *Energies*, 11:2163, 2018.
- [93] Jie Cai, Jiawei Luo, Shulin Wang, and Sheng Yang. Feature selection in machine learning: A new perspective. *Neurocomputing*, 300:70–79, 2018.
- [94] M Z Naser and Amir Alavi. Error Metrics and Performance Fitness Indicators for Artificial Intelligence and Machine Learning in Engineering and Sciences. 1, 2021.
- [95] Xin He, Kaiyong Zhao, and Xiaowen Chu. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

- [96] Jonathan Waring, Charlotta Lindvall, and Renato Umeton. Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artificial Intelligence in Medicine*, 104:101822, 2020.
- [97] Radwa Elshawy and Sherif Sakr. Automated Machine Learning: Techniques and Frameworks. In Ralf-Detlef Kutsche and Esteban Zimányi, editors, *Big Data Management and Analytics*, pages 40–69, Cham, 2020. Springer International Publishing.
- [98] Matthias Feurer, Aaron Klein, Katharina Eggenberger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and Robust Automated Machine Learning. In C Cortes, N Lawrence, D Lee, M Sugiyama, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, pages 2962–2970. Curran Associates, Inc., 2015.
- [99] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, 2016.
- [100] Erin LeDell and Sebastien Poirier. H2O AutoML: Scalable Automatic Machine Learning. 7th ICML Workshop on Automated Machine Learning, 2020.
- [101] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 1946–1956, New York, NY, USA, 2019. Association for Computing Machinery.
- [102] Lucas Zimmer, Marius Lindauer, and Frank Hutter. Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):3079–3090, 2021.
- [103] Nick Erickson, Jonas W Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alex Smola. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. 2020.
- [104] Microsoft. Neural Network Intelligence v2.0, 2021.
- [105] Rosalin Bonetta and Gianluca Valentino. Machine learning techniques for protein function prediction. *Proteins: Structure, Function, and Bioinformatics*, 88(3):397–413, 2020.

- [106] Ana Marta Sequeira, Diana Lousa, and Miguel Rocha. ProPythia: A Python package for protein classification based on machine and deep learning. *Neurocomputing*, 484:172–182, 2022.
- [107] Marcelo Boareto, Michel E B Yamagishi, Nestor Caticha, and Vitor B P Leite. Relationship between global structural parameters and Enzyme Commission hierarchy: implications for function prediction. *Computational biology and chemistry*, 40:15–19, oct 2012.
- [108] Afshine Amidi, Shervine Amidi, Dimitrios Vlachakis, and Nikos Paragios. A Machine Learning Methodology for Enzyme Functional Classification Combining Structural and Protein Sequence Descriptors. In *Bioinformatics and Biomedical Engineering*, volume 9656, pages 728–738, 2016.
- [109] Vanessa Isabell Jurtz, Alexander Rosenberg Johansen, Morten Nielsen, Jose Juan Almagro Armenteros, Henrik Nielsen, Casper Kaae Sønderby, Ole Winther, and Søren Kaae Sønderby. An introduction to deep learning on biological sequence data: examples and solutions. *Bioinformatics (Oxford, England)*, 33(22):3685–3690, nov 2017.
- [110] Annalisa Franco, Alessandra Lumini, Dario Maio, and Loris Nanni. An enhanced subspace method for face recognition. *Pattern Recognition Letters*, 27(1):76–84, 2006.
- [111] M Sandberg, L Eriksson, J Jonsson, M Sjöström, and S Wold. New chemical descriptors relevant for the design of biologically active peptides. A multivariate characterization of 87 amino acids. *Journal of medicinal chemistry*, 41(14):2481–2491, jul 1998.
- [112] Beakcheol Jang, Inhwan Kim, and Jong Wook Kim. Word2vec convolutional neural networks for classification of news articles and tweets. *PLOS ONE*, 14(8):1–20, 2019.
- [113] Beakcheol Jang, Inhwan Kim, and Jong Wook Kim. Word2vec convolutional neural networks for classification of news articles and tweets. *PLOS ONE*, 14(8):1–20, 2019.
- [114] Shuyan Ding, Yan Li, Zhuoxing Shi, and Shoujiang Yan. A protein structural classes prediction method based on predicted secondary structure and PSI-BLAST profile. *Biochimie*, 97:60–65, feb 2014.
- [115] Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C Lawrence Zitnick, Jerry Ma, and Rob Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences of the United States of America*, 118(15), apr 2021.

- [116] Chaolu Meng, Yue Pei, Yongbo Bu, Quan Zou, and Ying Ju. Machine learning-based antioxidant protein identification model: Progress and evaluation. *Journal of Cellular Biochemistry*, 124, 2023.
- [117] Pengmian Feng, Wei Chen, and Hao Lin. Identifying Antioxidant Proteins by Using Optimal Dipeptide Compositions. *Interdisciplinary sciences, computational life sciences*, 8(2):186–191, jun 2016.
- [118] Chaolu Meng, Shunshan Jin, Lei Wang, Fei Guo, and Quan Zou. Aops-svm: A sequence-based classifier of antioxidant proteins using a support vector machine. *Frontiers in Bioengineering and Biotechnology*, 7, 09 2019.
- [119] Saeed Ahmed, Muhammad Arif, Muhammad Kabir, Khaistah Khan, and Yaser Daanial Khan. Predaodp: Accurate identification of antioxidant proteins by fusing different descriptors based on evolutionary information with support vector machine. *Chemometrics and Intelligent Laboratory Systems*, 228:104623, 2022.
- [120] Chaolu Meng, Yue Pei, Quan Zou, and Lei Yuan. DP-AOP: A novel SVM-based antioxidant proteins identifier. *International Journal of Biological Macromolecules*, 247:125499, 2023.
- [121] Ivan Dimitrov, Ivan Bangov, Darren R Flower, and Irini Doytchinova. AllerTOP v.2—a server for in silico prediction of allergens. *Journal of molecular modeling*, 20(6):2278, jun 2014.
- [122] Neelam Sharma, Sumeet Patiyl, Anjali Dhall, Akshara Pande, Chakit Arora, and Gajendra P S Raghava. AlgPred 2.0: an improved method for predicting allergenic proteins and mapping of IgE epitopes. *Briefings in bioinformatics*, 22(4), jul 2021.
- [123] Pallavi M Shanthappa and Rakshitha Kumar. ProAll-D: protein allergen detection using long short term memory - a deep learning approach. *ADMET & DMPK*, 10(3):231–240, 2022.
- [124] Chun He, Xinhai Ye, Yi Yang, Liya Hu, Yuxuan Si, Xianxin Zhao, Longfei Chen, Qi Fang, Ying Wei, Fei Wu, and Gongyin Ye. DeepAlgPro: an interpretable deep neural network model for predicting allergenic proteins. *Briefings in Bioinformatics*, 24(4):bbad246, 06 2023.
- [125] Peng-Mian Feng, Hao Lin, and Wei Chen. Identification of antioxidants from sequence information using naïve Bayes. *Computational and mathematical methods in medicine*, 2013:567529, 2013.
- [126] Muhammad Usman, Shujaat Khan, Seongyong Park, and Jeong-A Lee. AoP-LSE: Antioxidant Proteins Classification Using Deep Latent Space Encoding of Sequence Features. *Current issues in molecular biology*, 43(3):1489–1501, oct 2021.

- [127] Guoli Wang and Jr Dunbrack, Roland L. PISCES: a protein sequence culling server. *Bioinformatics*, 19(12):1589–1591, 08 2003.
- [128] Weizhong Li and Adam Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659, 05 2006.
- [129] Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rehawi, Yu Wang, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Martin Steinegger, DEBSINDHU BHOWMIK, and Burkhard Rost. Prottrans: Towards cracking the language of life's code through self-supervised deep learning and high performance computing. *bioRxiv*, 2020.