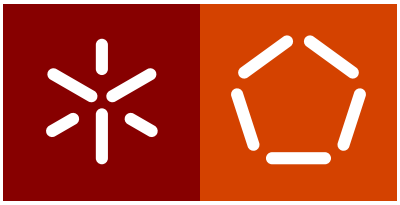
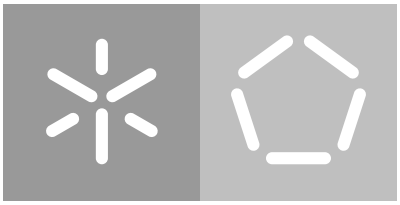


**Universidade do Minho**
Escola de Engenharia
Departamento de Informática

Ivan Alexandre Pereira Gomes

# Development of language modelling techniques for protein sequence analysis.

December 2022



**Universidade do Minho**
Escola de Engenharia
Departamento de Informática

Ivan Alexandre Pereira Gomes

# Development of language modelling techniques for protein sequence analysis.

Master dissertation
Master Degree in Bioinformatics

Dissertation supervised by
**Miguel Francisco Almeida Pereira da Rocha**

December 2022

## DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

# ACKNOWLEDGEMENTS

The realization of this master's thesis undoubtedly relied on important support and incentives, without which it would never have become a reality.

I would like to begin by expressing my gratitude to Prof. Dr. Miguel Rocha, for the guidance, the knowledge, the opinions, suggestions and criticisms, and for all the words of encouragement.

To my professors of the MSc in Bioinformatics, my thanks for the diversity and wealth of knowledge so enthusiastically transmitted, which greatly enhanced my training and strengthened the theoretical foundations essential to the development of this dissertation.

To Carolina, my colleague in the Master's program, I would like to thank for her friendship, solidarity, advice, exchange of opinions, and assiduous comments.

To Francisco, my friend Tuno, I thank for all the support, especially in the final stage, which was indispensable for the conclusion of this work. I also thank the Tuna Universitária do Minho for their understanding and solidarity.

To my mother, Manuela, I thank her for her affectionate and unconditional support, for her constant encouragement and for always believing in my work. Also to my grandmother Maria, my grandfather António and my godmother Fernanda.

A very special word of appreciation to Marta, who volunteered to accompany me in this challenge, being, without a doubt, the great driving force behind this thesis and, at the same time, the guarantor of its accomplishment. I cannot fail to underline her total availability, friendly advice, continuous support and encouragement, for which I am very grateful.

A final word of gratitude to all those not explicitly named here who, directly or indirectly, also contributed to the realization of this project.

To all my sincere thanks.

## 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.

## ABSTRACT

Nowadays, the ability to predict protein functions directly from amino-acid sequences alone remains a major biological challenge. The understanding of protein properties and functions is extremely important and can have a wide range of biotechnological and medical applications. Technological advances have led to an exponential growth of biological data challenging conventional analysis strategies. High-level representations from the field of deep learning can provide new alternatives to address these problems, particularly NLP methods, such as word embeddings, have shown particular success when applied for protein sequence analysis.

Here, a module that eases the implementation of word embedding models toward protein representation and classification is presented. Furthermore, this module was integrated in the ProPythia framework, allowing to straightforwardly integrate WE representations with the training and testing of ML and DL models.

This module was validated using two protein classification problems namely, identification of plant ubiquitylation sites and lysine crotonylation site prediction. This module was further used to explore enzyme functional annotation. Several WE were tested and fed to different ML and DL networks. Overall, WE achieved good results being even competitive with state-of-the-art models, reinforcing the idea that language based methods can be applied with success to a wide range of protein classification problems.

This work presents a freely available tool to perform word embedding techniques for protein classification. The case studies presented reinforce the usability and importance of using NLP and ML in protein classification problems.



**Keywords:** Protein Classification, Word embedding, Natural Language Processing, Machine Learning, Deep Learning

## RESUMO

Hoje em dia, a habilidade de prever a função de proteínas a partir apenas da sequências de amino-ácidos permanece um dos grandes desafios biológicos. A compreensão das propriedades e das funções das proteinas é de extrema importância e pode ter uma grande variedade de aplicações médicas e biotecnológicas. Os avanços nas tecnologia levaram a um crescimento exponencial de dados biológicos, desafiando as estratégias convencionais de análise. O campo do *Deep Learning* pode providenciar novas alternativas para atender à resolução destes problemas, em particular, os métodos de processamento de linguagem, como por exemplo *word embeddings*, mostraram especial sucesso quando aplicados para análise de sequências proteicas.

Aqui, é apresentado um módulo que facilita a implementação de modelos de "word embedding" para representação e classificação de proteínas.

Além disso, este módulo foi integrado na *framework* ProPythia, permitindo integrar diretamente as representações WE com o treino e teste de modelos ML e DL.

Este módulo foi validado usando dois problemas de classificação de proteínas, identificação de locais de ubiquitilação de plantas e previsão de locais de crotonilação de lisinas. Este módulo foi usado também para explorar a anotação funcional de enzimas. Vários WE foram testados e utilizados em diferentes redes ML e DL. No geral, as técnicas de WE obtiveram bons resultados sendo competitivas, mesmo com modelos descritos no estado da arte, reforçando a ideia de que métodos baseados em linguagem podem ser aplicados com sucesso a uma ampla gama de problemas de classificação de proteínas.

Este trabalho apresenta uma ferramenta para realizar técnicas de *word embedding* para classificação de proteínas. Os caso de estudo apresentados reforçam a usabilidade e importância do uso de NLP e ML em problemas de classificação de proteínas.



**Palavras-chave:** Classificação de Proteínas, Word embedding, Processamento de Linguagem Natural, Machine Learning, Deep Learning

CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ACRONYMS

| | |
|---|---|
| AI | Artificial Intelligence. |
| ANN | Artificial Neural Networks. |
| AUC | Area Under Curve. |
| CBOW | Continuous Bag-of-words. |
| CNN | Convolutional Neural Networks. |
| DL | Deep Learning. |
| DNN | Deep Neural Networks. |
| DR | Dropout. |
| EC | Enzyme Comission. |
| GNB | Gaussian Naive Bayes. |
| GPUs | Graphics Processing Units. |
| HMM | Hidden Markov Models. |
| KNN | K-Nearest Neighbors. |
| LR | Logistic Regression. |
| LSTM | Long short-term memory. |
| MCC | Matthews Correlation Coefficient. |
| ML | Machine Learning. |
| NLP | Natural Language Processing. |
| OVO | One vs One. |
| OVR | One vs Rest. |

RF Random Forest.
RNN Recurrent Neural Networks.
ROC Receiver Characteristic Operator.

SGD Stocastic Gradient Descendent.
SN Sensibility.
SP Specificity.
SVD Singular Value Decomposition.
SVM Support Vector Machine.

WE Word Embedding.

# 1

# INTRODUCTION

## 1.1 CONTEXT AND MOTIVATION

Proteins are large biomolecules, consisting of chains of amino acids present in all living beings. They have a wide set of crucial functions in biological processes, such as catalysis of metabolic reactions, DNA repair, transport of molecules and others. The understanding of protein properties and functions is extremely important and can have a wide range of biotechnological and medical applications [1].

To date, the ability to predict protein functions directly from amino-acid sequences alone remains a major biological challenge [2]. Besides, technological advances have led to an exponential growth of biological data challenging conventional analysis strategies. The advances in high-level representations from the field of deep learning can provide new alternatives to address these problems [3, 4].

Deep Learning (DL) algorithms can learn representations of data and extract abstract and complex patterns, thus emerging as suitable approaches for large-scale protein analysis. Natural Language Processing (NLP) is the branch of artificial intelligence focused on understanding text and spoken words [5]. Common NLP methods include Word Embedding (WE), such as Word2Vec [6], Embedding from Language Model (ELMo), that uses word tokens as input to recurrent neural networks [7] and more complex architectures like Transformers that use attention mechanisms encoders and decoders [8]. Deriving motivations from the success of NLP methods, several language models, such as ProtVec [9], SeqVec [10], and ProtBERT [11], have been adopted and applied for protein sequence analysis.

ProPythia is a framework for the classification of proteins using machine and deep learning developed within the Biosystems group at CEB/ U. Minho. One of the objectives of this thesis will be the construction of a tool that will allow us to build, train and validate language models for protein classification tasks and integrate it within ProPythia platform [12].

## 1.2 OBJECTIVES

The aim of this work is to build a module that will allow to build, train and validate techniques already used in NLP for sequence-based protein classification tasks.

In detail, the technological objectives are:

- Review the state of the art in deep learning and NLP techniques, focusing on those applied to biological sequences and protein classification.
- Develop a framework that will ease the implementation of Word embedding models toward protein representation and classification.
- Validate the platform with protein classification case studies.
- Apply the developed pipeline in combination with ML and DL methods to the enzyme classification problem.
- Integrate the developed module in ProPythia package.
- Write the thesis with the results of the previous topics.

## 1.3 STRUCTURE OF THE DOCUMENT

This document is organized in 7 chapters, each of which is briefly described as follows:

- Chapter 1 - Introduction: Overview of the subject of this work. It presents the motivation and context and main goals.
- Chapter 2 - Machine Learning and Deep Learning: Introduction to theoretical concepts of machine and deep learning, as well as their algorithms, workflows, and architectures.
- Chapter 3 - Deep Learning applied to proteins: Introduction to protein sequences, applications of deep learning in protein sequences and overview of relevant previous work.
- Chapter 4 - Development and implementation: Methods for implementing the proposed Word embedding module.
- Chapter 5 - Validation: Overview of the results of case studies used for the validation of the proposed Word embedding module.
- Chapter 6 - Enzyme classification case study: This chapter presents the enzyme case study. The methods and results will be reported followed by a discussion of the main results.
- Chapter 7 - Conclusions and future prospects: Finally, this chapter highlights the main conclusions of the thesis followed by a description of possible improvements and future work.

# 2

# MACHINE LEARNING AND DEEP LEARNING

## 2.1 MACHINE LEARNING FUNDAMENTALS

An algorithm is a sequence of instructions or steps, transforming the input into an output, to solve a problem or reach a goal, and it is needed to solve a problem in a computer. One can use specific algorithms to identify certain patterns or regularities on data and build useful and robust approximations. In the end all the data was not explained but the majority of the process was understood. This is the main point of the Machine Learning (ML)[13, 14].

ML its a part of Artificial Intelligence (AI), this is, a system that have the ability to learn. This learning and adaptation is done using example data or past experience and allows the system to provide solutions for all possible scenarios.

In a model of ML, we define some parameters, and execute a computer program to optimize this parameters that's what we call "learning". The objective may be to have a predictive model or a descriptive model or both, to make predictions for the future or to obtain knowledge from the data[14].

We can divide ML methods in four different categories: supervised learning, self-supervised learning, reinforcement learning and unsupervised learning.

### 2.1.1 SUPERVISED LEARNING

Supervised or predictive learning approach, is the most used ML method by far. Nowadays, big part of the applications of DL belong to this category, such as speech recognition, language translation, etc. It basically consists in learning through an input data or annotations with given examples annotated by humans most of the time[15, 16].

To solve a supervised learning problem, the first step is to decide what kind of data will be used as training set. In the case of proteins, the decision is between only domains, functional sites or the entire protein sequence. To gather the dataset, we have to consider that the data is a representation of the real world variables, and it is possible to generalize the predictions

from them. The next step is to transform the input into a feature vector. The number of attributes is important because it will interfere directly with the accuracy of the model, and if is too large, the training will be too "heavy" and slow. On the other hand, the input means to have enough information for the model to be accurate and as close as possible to the real function[17].

The next step is to determine the structure of the model and the supervised learning algorithm(for example, logistic regression or Artificial Neural Networks (ANN)). In some learning algorithms we may need to choose control parameters which will be adjusted via cross-validation, or optimizing the performance of the validation set, which is a subset of the training set[18].

After completing the design, running the algorithm on the training set and doing the parameter adjustment, we have to evaluate the accuracy of the learned function. Both the performance of the model and how accurate the learning is, are measured on a separate subset from the training set, called the test set[19].

*Decision Trees(DT)*

The fundamental learning approach is to recursively divide the training data into buckets of homogeneous members through the most discriminative dividing criteria. The measurement of "homogeneity" is based on the output label; when it is a numeric value, the measurement will be the variance of the bucket; when it is a category, the measurement will be the entropy or gini index of the bucket[14].

The good part of Tree is that it is very flexible in terms of the data type of input and output variables which can be categorical, binary and numeric value. The level of decision nodes also indicate the degree of influences of different input variables. The limitation is each decision boundary at each split point is a concrete binary decision. Also the decision criteria only considers one input attribute at a time but not a combination of multiple input variables. Another weakness of Tree is that once learned it cannot be updated incrementally. When new training data arrives, you have to throw away the old tree and retrain every data from scratch[20].

*Support Vector Machines(SVM)*

Support Vector Machine takes numeric input and binary output. It is based on finding a linear plane with maximum margin to separate two classes of output. Categorical input can be turned into numeric input as before and categorical output can be modeled as multiple binary output. The strength of SVM is that it can handle large number of dimensions. With the kernel function, it can handle non-linear relationships as well[14].

*Bayesian Networks(BN)*

It is basically a dependency graph where each node represents a binary variable and each edge (directional) represents the dependency relationship. If NodeA and NodeB has an edge to NodeC. This means the probably of C is true depends on different combinations of the boolean value of A and B. NodeC can point to NodeD and now NodeD depends on NodeA and NodeB as well. The learning is about finding at each node the join-probability distribution of all incoming edges. This is done by counting the observed values of A, B and C and then update the joint probability distribution table at NodeC. The strength of Bayesian network is that it is highly scalable and can learn incrementally because all we do is to count the observed variables and update the probability distribution table. Similar to Neural Network, Bayesian network expects all data to be binary, categorical variable will need to be transformed into multiple binary variable as described above. Numeric variable is generally not a good fit for Bayesian networks[20].

### 2.1.2 Self-supervised learning

Self-supervised learning is a method where there is no interference of humans in the loop, where the "autoencoders" -self-supervised learning's instances- targets are the raw input, eliminating the need of humans labelling the data. By extracting automatically from data, this method gets available correlations, embedded metadata, or domain knowledge present in the input. An example of self-supervised learning can be predicting the next word in a text through previous words. In this method of ML has use cases in regression and classification, like supervised learning[16, 21].

### 2.1.3 Reinforcement learning

Reinforcement learning, different from supervised learning, creates its own dataset. The model learns from its experience, so is trained with the answer depending of the situation. There is no correct answer, this method uses the data that performs better to the given task[18].

To the data, this ML method is only a research area because, in exception of games, it was not successful. Despite this, it should be expected that in the near future the reinforcement learning will be used in a large range of applications, like robotics, self-driving, etc[16].

### 2.1.4 Unsupervised Learning

In this method, different from supervised learning, only inputs are given, $D = (x_i)N\ i = 1$. The unsupervised learning or descriptive learning is the second main type of ML. Its main goal is to discover the interaction between the data, exploring its structure and forming patterns, also known as "knowledge discovery"[18, 22].

In this case the definition of the problem is much more ambiguous because it is not given to the model what kind of patterns we are looking for, and, unlike supervised learning, there is no error metric because there are no predictions. Because of that, and comparatively to supervised learning, there is no subdivision of training set and test set, leading to a smaller and simpler pipeline, where the objective is knowledge discovery, valid according to the case study[16, 17, 21].

#### *Clustering*

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). It is a task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics[19].

Cluster analysis itself is not one specific algorithm, but the general task to be solved. It can be achieved by various algorithms that differ significantly in their understanding of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with small distances between cluster members, dense areas of the data space, intervals or particular statistical distributions. Clustering can therefore be formulated as a multi-objective optimization problem[17].

The appropriate clustering algorithm and parameter settings (including parameters such as the distance function to use, a density threshold or the number of expected clusters) depend on the individual data set and intended use of the results. Cluster analysis is not an automatic task, but it is an iterative process of knowledge discovery or interactive multi-objective optimization that involves trial and failure. It is often necessary to modify data preprocessing and model parameters until the result achieves the desired properties[23].

#### *PCA*

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called

principal components. If there are $n$ observations with $p$ variables, then the number of distinct principal components is $\min(n-1, p)$[24]. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component, in turn, has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors (each being a linear combination of the variables and containing $n$ observations) are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables[25].

### 2.1.5 Feature Selection

Feature selection is the automatic selection of the most relevant features to use in the model predictions. This method can be use to remove irrelevant and redundant variables from the dataset. By using fewer variables on the input data be able to build simpler and more accurate models. When we use a large amount of features the probability of the model to learn from the "noise" of the data is higher, so we aim to use less redundant attributes, preventing overfitting[26].

In addition to making a more generalist model, a good feature selection leads to less time for model training, by reducing the algorithm complexity. Therefore we can say that the main objectives are to improve the accuracy and to reduce the overfitting and training time[26, 27].

There are three main feature selection methods: filter, wrapper and embedded methods.

Filter feature selection methods uses a statistical measure to attribute a score to each feature, and by this score are selected to be removed or not from the dataset. This methods are generally used as a preprocessing step. The selection of features is independent of any machine learning algorithms. Some examples of filter methods include Pearson's Correlation, ANOVA and Chi squared test[27].

Wrapper Methods consists in preparing different sets of features and training the model with the different sets. Based on accuracy of the models, the best combination of features are selected. These methods are usually computationally very expensive. The most common examples of this methods are forward selection and backward elimination[27]. Forward selection is a method that starts by having no features in the dataset and is added iteratively new features. For each iteration, the best features that improves the model's predictions are added.In the reverse way, the backward elimination start with all features and remove the least significant in each iteration. This is repeated until no improvement in predictions are observed[27].

While the model is train, Embedded Methods learn what attributes influence the prediction of the model and contribute to its accuracy. The most common type of embedded feature selection methods are regularization methods, also known as penalization methods[28]. By adding constraints into the optimization of the learning algorithm, they change the bias values

and lower the complexity of the model. Examples of regularization algorithms are the LASSO, and Ridge Regression[27].

### 2.1.6 Bias/Variance Tradeoff

The bias/variance tradeoff is one way of analyzing the generalization error from the model, by sum the bias, variance, and the irreducible error(value resulting from the noise of the data).

In supervised learning the bias/variance decomposition is a major problem. On the one hand, bias is directly associated to the relation between features and target outputs, and high bias can cause the model prediction to miss the relevance in this relation, causing underfitting. On the other hand, variance is associated to the sensitivity to noise present in the training data, where high variance can cause defice in generalization of the prediction of the model, also called overfitting. Typically, models with high bias in parameter estimation tend to have a low variance, and vice versa. Therefore, the main objective is to have a model that has less of these type of errors, preventing the algorithm from overfit or underfit[29].

When we have low bias in the models, the models are more complex and this enables them to be accurate when present with new data. This is due to the training process of the model learn from the large noise component present in the training set. In contrast, when we have higher bias the models are too simple, producing predictions with lower variance and predictions less accurate too[30].

### 2.1.7 Model Evaluation

After the training of the model we evaluate how well the prediction fit in real data that it has never seen before, and the model is low on bias and variance. By measuring how accurately the learning of the model predict in unseen data, we can estimate generalization error. The error estimation gives a measure of the quality of the model and is crucial to choose the learning method[31].

#### *Validation*

In supervised learning we need to know how accurately an algorithm is able to predict values from unseen data. This measure is called generalization error. This measure can be reduced by avoiding overfit of the model. Because learning algorithms are evaluated on finite samples, the evaluation of a learning algorithm may be sensitive to sampling error[21]. As a result, measurements of prediction error on the current data may not provide much information about predictive ability on new data[16].

The performance of the learning algorithm is measured through the learning curves that is plots of generalization error values by the learning process[32].

Therefore, validation is the process of deciding if the prediction results represent well relationships between variables and are capable to describe the data patterns. Most of the time, an error estimation is made after training the model. In this process, a numerical estimate of the difference in predicted and original responses is done, also called the training error[31]. Unfortunately, , this evaluation technique only gives us an idea about how well our model does on data used to train it and does not give an indication of how well the model will generalize to an unseen data. Getting this idea about our model is known as Cross-validation[33].

Cross-validation divide the dataset into complementary subsets, training set and validation set. The first one is used to perform the analysis and the other subset is use to validate the analysis. Using multiple rounds of this method with different partitions, cross-validation can reduce the variability. In the end of each round the results are combined measures of fitness to give an estimate of the predictive performance of the model[30, 34].

*Classification Accuracy*

Classification Accuracy it is the ration of correct predictions to the total of samples.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made} \tag{1}$$

This method is great and easy to apply but in some situations can gives us a false sense of achieving high accuracy[30].

*Confusion Matrix*

Confusion Matrix as the name suggests gives us a matrix as output and describes the complete performance of the model. Using a binary problem as an example, where the results of the predicts can be, "YES" or "No", we build an example confusion matrix[17].

| | Predicted: No | Predicted: YES |
|---|---|---|
| Actual: NO | *x* | y |
| Actual: YES | z | k |

Consulting the matrix we can know the True Positives cases, $k$, which we predicted YES and the actual output was also YES, the True Negatives, $x$, which we predicted NO and the actual output was also "YES", the False Negatives, $z$, which we predicted NO and the actual output

was YES and the False Positives cases, $y$, we predicted YES and the actual output was NO. The accuracy in this method is measured using the values of the main diagonal of the matrix.

$$Accuracy = \frac{TruePositives + FalseNegatives}{Total\ Number\ of\ Samples} \tag{2}$$

*AUC*

Area Under Curve (AUC) is one of the most common metrics used for model evaluation.AUC is the area under the receiver operatings characteristic curve, or ROC curve. A ROC curve is a graph showing the performance of a classification model at all classification thresholds. To construct the graph it is necessary to calculate the True Positive Rate, also known as Sensitivity, and False Positive Rate, also known as Specificity[35].

True Positive Rate corresponds to the proportion of positive data predictions that are correctly considered as positive, with respect to all positive data points, correct or incorrect classified.

$$TruePositivesRate = \frac{TruePositive}{FalseNegative + TruePositive} \tag{3}$$

False Positive Rate its equal to True Positive Rate, but corresponds to the negative data predictions that are wrongly considered as positive with respect to all negative data points, correct or incorrect classified.

$$FalsePositiveRate = \frac{FalsePositive}{FalsePositive + TrueNegative} \tag{4}$$

AUC is the area under the curve of plot Specificity versus Sensitivity at different points in range 0 to 1. AUC has a range of 0 to 1 and the closer the value is to 1, the better the model performance.

*F1 Score*

F1 score, also known as, F measure, measures the balance between the Precision and the Recall. Precision corresponds to the number of True Positives divided by all positive results predicted by the model. Recall is the number of True Positives divided by the number of all values that should have been identified as positive values, in another words, the number of True Positives and False Negatives[36].

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{5}$$

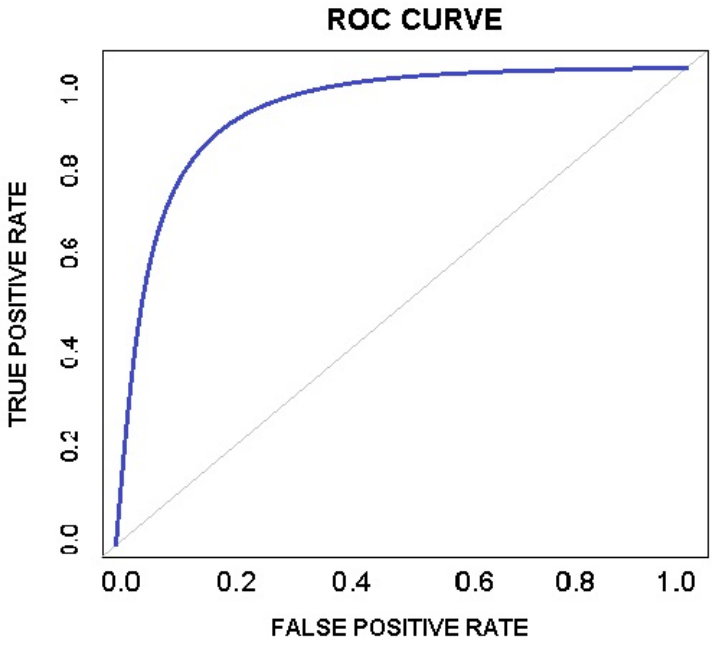



Figure 1 – AUC

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{6}$$

High precision but lower recall, gives you an extremely accurate, but it then misses a large number of instances that are difficult to classify. The greater the F1 Score, the better is the performance of our model.

$$F1 = 2 * \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{7}$$

## 2.2 Artificial Neural Networks

Inspired by neuroscience and the way human learn basic concepts are born ANN[16, 22]. The similarity between the machine learning neural networks and the biological neural networks is, on the one hand, the architecture and the learning process of the neural network, and, on the other hand, is interneuron connection, know as synaptic weights, responsible for storing the knowledge, acquired by the learning process[34].

ANN consists of a processor distributed for layers made up of simple processing units, called neurons, with the objective of storing knowledge in a hierarchical way. The strength of a connection between neurons is called weight and the weight between two neurons $i$ and $j$ is referred as $w_{i,j}$.

There are four main objects in ANN, and they interact in the following way: the layers, which combined we call them network, maps the input dataset into predictions. The loss function measure how far the predict output is to the expected, the robustness of model increases along with the decrease its value; The optimizer, which is an algorithm used to define the learning proceeds through weights' updates[15].

### 2.2.1 Neurons

A neuron, or perceptron, its nothing more than a mathematical function.

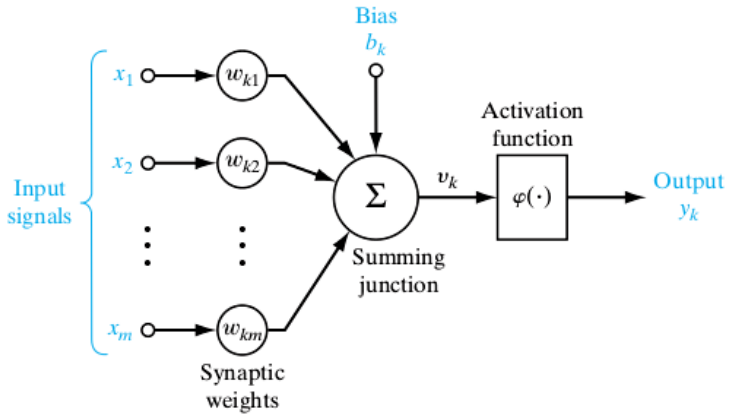



Figure 2 – Neuron Diagram

The $x_m$ values refer to inputs, either the original features or inputs from a previous hidden layer. At each layer, there is also a bias, $b_k$, which can help better fit the data, where the $k$ referees to the number of neuron connected. The $x_m$, are multiplied by the weight, $w_{km}$, and added together, then is added the bias value, $b_k$. This value pass through the activation function, $\varphi(\dot{})$, to become the neuron's output[34]. The neuron passes the value to all neurons it is connected to in the next layer, or returns it as the final value, and the calculation its done by equation 8.

$$z = \sum (x_n w_n) + b \; n \in \mathbb{N} \tag{8}$$

*Linear Activation Function*

In this activation function the $a$ value is a constant. The range goes from $-\inf$ to $inf$, and due to that the values can get very large. This linear activation function alone can not learn complex patterns.

$$f(z) = az \tag{9}$$

*Sigmoid Activation Function*

Sigmoid activation functions can capture more complex patterns because its a non-linear function. The range goes from $0$ to $1$, leading to a problem called "vanishing gradient", because the output values are bounded.

$$sigmoid(z) = \frac{1}{1 + e^{-z}} \tag{10}$$

*Hyperbolic Tangent Activation Function*

Like sigmoid activation functions, hyperbolic tangent, all known as "tanh", its a non linear function. In this case, instead of $0$ the lower bound its $-1$ can thus understand negative values, but also has vanishing gradient problem.

$$tanh(z) = \frac{e^{z} - e^{-}z}{e^{z} + e^{-z}} \tag{11}$$

*Rectified Linear Unit Activation Function*

Rectified Linear Unit, or ReLU, its a activation function with range $0$ to $\inf$. Due to does not allowing negative values, certain patterns may not be captured, leading to the "dying ReLU problem".

$$relu(z) = max(0, z) \tag{12}$$

*Leaky ReLU Activation Function*

Leaky ReLU Activation Function was construct in an attempt to solve "dying ReLU problem", with no bounded range.

$$leakrelu(z) = \begin{cases} 0.01z, & \text{for } z < 0. \\ \text{z}, & \text{for } z \geq 0. \end{cases} \tag{13}$$

Instead of using 0.01, we can replace this value for a parameter, $\alpha$, which will be changed during the training process, referred as Parametric ReLU (PReLU).

$$prelu(z) = \begin{cases} \text{az}, & \text{for } z < 0. \\ \text{z}, & \text{for } z \geq 0. \end{cases} \tag{14}$$

*Softmax Activation Function*

Softmax its a activation function that is only used in the output layer rather than throughout the network. This activation function divides output so the total sum is equal to 1, being a advantage in model probability distributions.

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{15}$$

### 2.2.2 Artificial Neural Networks Families

In ANN, there are two main families of neural networks: Feed-forward Neural Networks and Recurrent Neural Networks (RNN).

Feed-forward neural networks are the earliest form of ANN due to its simplicity compare to RNN. The main difference between the two families is that feed-forward neural network don't have cycles in the connections between neurons,instead the information moves through the layers in one direction[37].

In this type of neural network the first layer, known as input layer, is the layer that receives the input dataset use to the learning process. The last layer is the output layer, where the number of neurons is equal to the output attributes returning a numerical value. The layers between the input layer and output layer are called hidden layers, and may have several functions depending on the architecture or ultimate goal of the ANN[38].

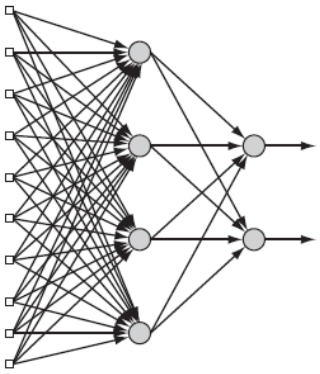


Figure 3 – Feed-forward Neural Network Diagram

In the case of RNN we have the phenomenon of recurrence in which the neurons are connected cyclically. These cycles give the ANN the capacity to have a temporal-like behaviour, and sometimes these neural networks do not have a defined input or output neurons[37, 39].

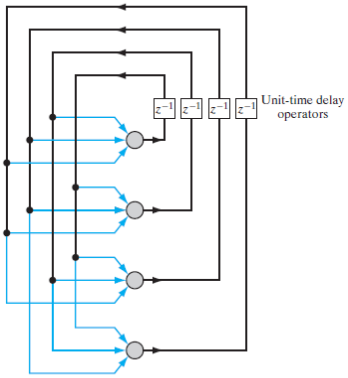



Figure 4 – Recurrent Neural Network Diagram

### 2.2.3 Training Algorithms

The neural networks are usually trained by using iterative gradient descent optimizers. The main objective of the this algorithms is to reduce the cost function to a very low value. By a differentiable function we can find the minimum analitytically. The minimum point of a function is the point where the derivative is 0. Therefore, finding all points and check which have the lowest value we achieve the absolute minimum. Translated to the ANN that means find analytically the combination of weights values that lead to a lowest lost function value possible [15, 20].

Stochastic gradient descent is a stochastic approximation of gradient descent optimization. Also called incremental gradient descent is a method with the main objective of optimize interactively an differentiable objective function. The gradient of $Q(w)$ is given for the equation:

$$w := w - \eta \nabla Q_i(w) \tag{16}$$

The algorithm can pass several times through the training set until converge to a minimum. For each loop to the training set the above update is applied to each training example.

**Algorithm 1** SGD

Choose an initial vector of parameters, $w$ and learning rate, $n$.
**while** approximate minimum is not obtained **do**
  Randomly shuffle examples in the training set.
  **for** $i$ in range $n$ **do**
    $w := w - \eta \nabla Q_i(w)$
  **end for**
**end while**

There are different variants of SGD that differ in the way that the algorithm computes the weight updates. They are called "optimizers" and the most common are "SGD with momentum", "ADAM" and "RMSprop", but there are several others[15].

### 2.2.4 Backpropagation Algorithm

The backpropagation algorithm is a genezalization of Delta rule and is used to update interatively the weights of the inputs in a layer. First the loss function have to calculate the difference between the predicted value from the output and the expected value, after a training iteration[39]. By using the chain rule to know the contribution of each parameter to the loss value backpropagation algorithm works backwards from the top layer to bottom layer. At each iteration the algorithm adjusts the weights so that its influence on the final result of the model is more precise[15, 39].

*Mean squared error (MSE)*

MSE is the most commonly used regression loss function. MSE is the sum of squared distances between our target variable and predicted values.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{17}$$

*Cross Entropy*

Cross entropy quantifies the difference between two probability distribution. Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.

$$\text{Cross entropy} = -\sum_{i}^{M} y_i \log(\hat{y}_i) \tag{18}$$

### 2.2.5 Regularization

Regularization, in the context of ML, make some changes in the learning algorithm with the main objective to prevent overfitting and underfitting, by imposing constraint on the learned method[28, 40].

*Lasso and Ridge Regularization*

The regularization is a a set of techniques that limits and prevents the model to fits too closely to the input data, allowing it to be more general predictions. This is possible, for a given learned relation, $Y$, by constraining or regularizating the coefficient estimates,$\beta$, to zero, for different predictions, $x_p$.

$$Y \approx \beta_0 + \beta_1 x_1 + \beta_2 x_2 + (...) + \beta_p x_p \tag{19}$$

The coefficients are chosen with the objective of minimize a loss function, known as "residual sum of squares" or RSS.

$$RSS = \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 \tag{20}$$

There are two types of techniques that the regression model can use: L1 regression, know as Lasso regression, and L2 regression, know as Ridge regression, and the difference between is the penalty term.

In one hand, L1 regression adds to the loss function the "absolute value of magnitude" as penalty term, in other hand L2 regression adds "squared magnitude".

$$L1\ regression = RSS + \lambda \sum_{j=1}^{p} |\beta_j| \tag{21}$$

$$L2\ regression = RSS + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{22}$$

These methods have a great impact on the model, reducing the variance without substantial increase the bias' values. The value of $\lambda$ is the tuning parameter in this techniques, controlling the change in values on bias and variance. When $\lambda = 0$, the penalty term has no effect. But if the value rises, it reduces the values of $\beta$, and consequently reduce the variance. So the goal here is to increase the value of $\lambda$ to the point that only reducing the variance, without loosing any important properties in the data, avoiding overfitting. But from a certain value the data begins to have a very high value of bias leading to underfiting[33].

*Dropout*

Dropout are a regularization technique used in ANN. Applied to layers, consists of randomly shot down some neurons present in them, during the training, by setting to zero. As result, hidden layers become more generally useful, producing more robust models. This is one of the most effective and common used regularization techniques in ANN[15, 28].

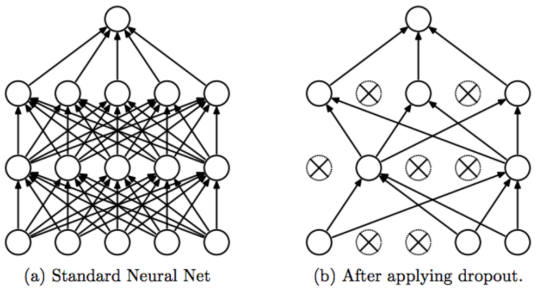



Figure 5 – Dropout Regularization Diagram

*Early Stopping*

A major problem with training ANN model is chosing the number of iteration of the training algorithm to use, known as epochs. High number of epochs can train a overfited model, and lower values of epochs can lead to underfitting. Early Stopping is a recursive method that stops training the model once the performance in the validation set stops to improve. The model chosen is the model with the lowest number of epochs, with better performance before the training begins to overfit.

*Batch Normalization*

To increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. The main objective of batch normalization technique is improve the performance and stability of ANN,leading to faster learning and a higher accuracy in the model by reducing internal covariate shift.

In the case of ANN, the input to each layer is affected by parameters in all the input layers. In fact, even small changes to the network get amplified down the network. This leads to change in the input distribution to internal layers of the deep network and is known as internal covariate shift.

Batch normalization regularizes the model and reduces the need for dropout, photometric distortions, local response normalization and other regularization techniques, also allowing the use of saturating nonlinearities and higher learning rates[41].

## 2.3 Deep Learning Concept

The idea of using Deep Neural Networks (DNN) goes back to 1950s but the lack of an efficient way to train large neural networks led to this approach taking decades to be used. In the

mid-1980s, with the rediscovered of the backpropagation algorithm and its application on neural networks, this changed[15, 20].

DL, also known as "layered representations learning" or "hierarchical representations learning", is a subfield of ML[22, 42]. While the other approaches of ML, sometimes called "shallow learning", tend to use on or two layers in the architecture of the model, modern DL uses a new insight into learning representations by using of a larger successive number of layers. This fact leads to the use of "deep" in "deep learning"[15].

Comparatively to thirty years ago, the hardware is much faster and this, allied to the introduction of Graphics Processing Units (GPUs) as processing units, helped the field of DL to grow. The use of new hardware and new training techniques led to being possible to train larger DNN in much less time , comparatively to be beginning of DL[37].

The way that we present the data to the model is crucial to the success of the learning. Nonetheless, by combining simpler features learned from data DL is capable of constructing hierarchical representation of data, learn useful complex features by themselves, through the use of ANN of multiple nonlinear layers, increasing levels of abstraction of learning[39].

### 2.3.1 Architectures

There are different architectures designed in order to have a better performance for specific tasks and here we present the most used architectures in DL.

#### *DNN*

A DNN is an ANN with several layers between the output layer and the input layer. The DNN finds the mathematically correct manipulation to turn the input into output, that can be either a non-linear or a linear relationship, and calculates the probability of each output by moving through the layers. DNNs have the ability to model non-linear relationships that seem to be complex. Its architecture generates models of composition where the expression of the object is a composition of primitives. The extra layers enable composition of features from lower layers, potentially modeling complex data with fewer units than a similarly performing shallow network[28].

#### *Convolutional Neural Networks (CNN)*

A convolutional neural network (Convolutional Neural Networks (CNN), or ConvNet) is a part of deep neural networks, mostly applied to analyzing visual imagery[39].

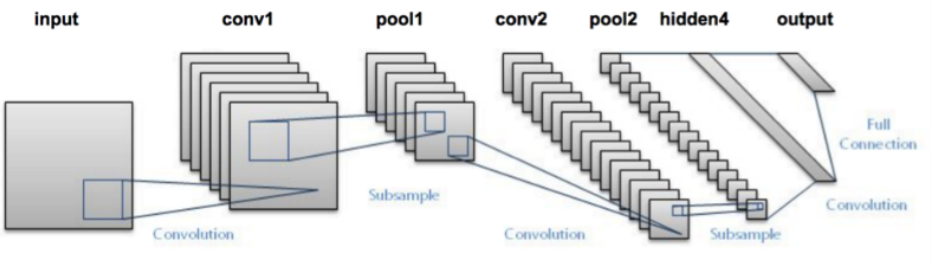



Figure 6 – Convolutional Neural Networks Diagram

Biological processes were the inspiration of convolutional networks[42]. Those processes were related to the resemblence of connectivity pattern between neurons and the organization of the animal's visual cortex. Cortical neurons respond to stimuli restrictly in the region of the visual field called receptive field. Receptive fields of various neurons overlap so that they can cover the totally of the visual field[28].

This type of network uses fewer pre-processing compared to other image classification algorithms, which means that it learns the filters that were hand-engineered in traditional algorithms. The major advantage is the independence of prior knowledge and the human effort in feature design. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, and natural language processing.

*Long short-term memory (LSTM)*

There are several architectures of Long short-term memory (LSTM)s units. The memory part of the unit is one model made of a cell and three "regulators" called gates: an input gate, an output gate and a forget gate. Some variations do not have one or more of these regulators or maybe have other types, of which is an example the gated recurrent units (GRUs) that do not have an output gate[28].

The cell's responsiblity is to keep track of the dependencies between elements in the sequence of input. The input gate controls the extent to which a new value flows into the cell, the forget gate controls the extent to which a value remains in the cell and the output gate controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit. The activation function of the LSTM gates is often the logistic function.

Connections can be into and out of the LSTM gates, and a few are recurrent. The weights of the connections need to be learned during training and they determine the gates' operation.

*Deep Autoencoders*

A special type of DNN is the deep autoencoder and its input vectors have the same dimensionality as the output vectors. It is often used to learn a representation or effective encoding of the original data, in the form of input vectors, at hidden layers[39]. The autoencoder is a nonlinear feature extraction method that does not use class labels. In fact, the extracted features pretend to conserve and better represent information other than performing classification tasks, but sometimes these two objetives are related[38].

Typically, an autoencoder has an input layer representing the original data or input feature vectors (e.g., pixels in image or spectra in speech), one or more hidden layers that represent the transformed feature, and an output layer which matches the input layer for reconstruction. If there is a greater than one number of hidden layers, the autoencoder is considered deeper. The dimension of the hidden layers can be either smaller (when the goal is feature compression) or larger (when the goal is mapping the feature to a higher-dimensional space) than the input dimension. This type of DNN is many times trained by using one of backpropagation variants, specially the stochastic gradient descent method[28, 43].

### 2.3.2 Deep Learning Frameworks and Tools

There are some frameworks built to deal with the problems DL present to us at the computational level, such as building different ANN deep architectures.

*Theano*

*Theano* is an open source project developed by the *LISA* group at the University of Montreal, Quebec. Libraries such as *Keras*, *Lasagne* and *Blocks* have been built on top of *Theano* and its name is inspired in a greek mathematician. Bying considered an standard for DL research and development, *Theano* is designed to work with large neural networks algorithms used in DL[44].

*TensorFlow*

*TensorFlow* is written in *C++*, offering an interface to *Python*. *TensonrFlow* offer some advantages such as, supports to reinforcement learning and other algorithms, offers computational graph abstraction and a faster compile time than *Theano*. It is possible to visualise learning curves or parameter updates, using *TensorBoard* and it can be deployed on multiple CPUs and GPUs[45].

*Keras*

*Keras* is a DL framework that provides convenient ways to define and train different types of DL models in any *Python*'s version, from 2.7 to 3.6. *Keras* has different features such as allowing to run the code on CPU or GPU. Had an user-friendly API to prototype DL models and a support for CNN, RNN, and any hybrid neural networks. With *Keras* it is possible to build arbitrary networks architectures, being essential to build any type of DL model. *Keras* supports several different bakend engines, such as *TensorFlow* backend, *Theano* backend and *CNTK* backend[15].

## 2.4 Word embedding models

NLP is the branch of artificial intelligence focused on understanding text and spoken words [5] intersecting computer science, linguistics, and machine learning.

Common NLP methods include word embeddings, such as Word2Vec [46], Embedding from Language Model (ELMo), that uses word tokens as input to recurrent neural networks [7] and more complex architectures like the Transformer Models that use attention mechanisms encoders and decoders [8].

Here, a more detailed explanation on Word embedding models, the focus of this thesis, is presented. The idea behind word embeddings is to represent words by vectors. These vectors capture hidden information about a language, like word analogies or semantics, and can also be used to improve the performance of text classifiers [47].

Published in 2013 by Google engineers, Word2vec was the first model focusing on producing word embeddings and reconstructing linguistic contexts of words [46].

Word2Vec is a shallow, two-layer neural network that takes as input a large corpus of text and produces a vector space typically of several hundred dimensions. Each unique word in the corpus is assigned a corresponding vector in the space, thus, depending on the corpus, the word vectors will capture different information [46, 47].

Word embeddings assume that the meaning of a word can be inferred by the company it keeps. In this way, words that often occur in similar contextual locations in the corpus also have a similar location in Word2vec space. In the same way, more dissimilar words are located farther from one another in the space. As words that have similar neighbor words are likely semantically similar, this means that the Word2vec approach can retain semantic relationships [46].

During training, Word2vec considers both individual words and a sliding window of context words surrounding individual words as it iterates over the entire corpus. To produce these distributed representations of words can use two different model architectures: Continuous

Bag-of-words (CBOW) and continuous skip-gram. A schematic example of both architectures can be seen in Figure 7.

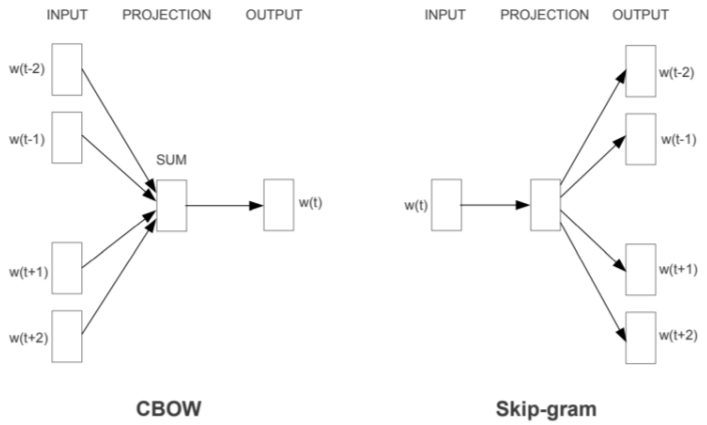



Figure 7 – A)The CBOW architecture predicts the current word based on the context. B) Skip-gram predicts surrounding words given the current word. Adapted from [46]

The CBOW model predicts the current word from the window of surrounding context words. The context is represented as a bag of words contained in a fixed-size window around the target word. The order of context words does not influence prediction (bag-of-words assumption). The skip-gram model uses the current word to predict the surrounding window of context words.

Thus, whereas the CBOW model predicts the target word according to its context, the skipgram model learns to predict a target word according to a nearby word. While CBOW is described as faster, skip-gram is better with infrequent words [46].

In 2016, FastText was introduced by T. Mikolov et al. from Facebook [48] in a tentative to improve the Word2vec model. the main difference is that FastText operates at a character level whereas Word2Vec operates at a word level.

FastText learns vectors for the n-grams that are found within each word, as well as each complete word. Thus, words are a bag of character n-grams. The authors claim that these vectors can be more accurate than Word2Vec vectors. Another advantage is that it can generate embedding vectors for words that are not part of the training texts, using its character embeddings. However, by creating word embedding vectors from its characters, FastText can lead to a high memory requirement [48]. Fastext is available as an open-source framework.

*Important WE model parameters*

In order to achieve good word representations the models should be fined tuned according to the problem in question and the amount of data available. The most important parameters of the WE model are:

- Dimension. Dimension controls the size of the vectors produced. The larger they are, the more information they can capture. Also, they will require more data to be learned and will be harder and slower to train. A common value is 100.
- Window context.The window context size is the maximum context location at which the words need to be predicted.
- Epoch. this parameter controls the number of epochs for model training and refers to how many times the model will loop over the data. The default value is set as 5, but for big datasets, this value can be reduced.
- Min count. Minimum occurrences of a gram to be considered. By default is set to 5. This means that only words with more than 5 occurrences in the corpus dataset are considered.
- Learning rate. The higher the value the faster the model converge to a solution but higher the risk of overfitting. the default is set to 0.05.
- Range of size of subwords. Only applied to fastext, it determines the size of all substrings contained in a word between the minimum and the maximum, 3 and 6 by default respectively [47].

Another advantage of these models is their interpretability. It is possible to get the nearest neighbors of a word vector and check the semantic information captured by the model [47].

One of the most popular and widely used libraries to implement word embedding approaches is gensim. Gensim is an NLP open source Python library for topic modeling, document indexing and similarity retrieval with large corpora. It can apply both word2vec and Fastext models with both CBOW and Skip-gram architectures [49].

# 3

# DEEP LEARNING APPLIED TO PROTEIN

## 3.1 PROTEIN SEQUENCES

Proteins are large biomolecules essential in every domain of life. They participate in almost every process within cells. Their functions range from catalyzing metabolic reactions, DNA replication, responding to stimuli, providing structure to cells and organisms, and transporting molecules from one location to another.

The protein building blocks are amino acids, which are small organic molecules that consist of an alpha (central) carbon atom linked to an amino group, a carboxyl group, a hydrogen atom, and a variable component called a side chain.

A protein sequence is composed of one or more long chains of amino acid residues. Amino acid residues are bonded together by peptide bonds and adjacent amino acid residues. There are twenty-six different amino acid residues, although only twenty are commonly found in proteins.

Each of the amino acids has a unique side chain with different chemistries, namely: nonpolar side chains, side chains with positive or negative charges and polar but uncharged side chains. The chemistry of amino acid side chains is critical to protein structure because these side chains can bond with one another to hold a length of protein in a certain shape or conformation.

Charged amino acid side chains can form ionic bonds, and polar amino acids are capable of forming hydrogen bonds. Hydrophobic side chains interact with each other via weak van der Waals interactions. The vast majority of bonds formed by these side chains are noncovalent. In fact, cysteines are the only amino acids capable of forming covalent bonds, which they do with their particular side chains. Because of side chain interactions, the sequence and location of amino acids in a particular protein guides where the bends and folds occur in that protein [50].

Proteins have four different levels of structure:

- Primary structure. The linear sequence of amino acids.

- Secondary structure. Interaction of non-consecutive amino acids within a chain can cause certain patterns of folding to occur, namely alpha helices and beta sheets.
- Tertiary structure. Ensemble of formations and folds in a single linear chain of amino acids. It describes 3-Dimensional folding.
- Quaternary structure. It refers to protein complexes resulting of interaction or assembling of multiple protein chains.

Proteins differ from one another primarily in their sequence of amino acids. The sequence of amino acid residues determines the protein folding into a specific 3D structure determining its activity and function, hence being very informative.

### 3.1.1 Enzymes

In this thesis, one of the main goals will be the classification of Enzyme protein sequences. Enzymes catalyze chemical reactions, regulating biological processes. They are usually specific and accelerate only one or few chemical reactions. They are responsible for the majority of the reactions involved in metabolism, as well as manipulating DNA in several processes. Because of their essential role in so many biological processes, enzyme prediction and function annotation have a broad range of applications and have been a challenging problem in bioinformatics.

The Nomenclature Committee of the International Union of Biochemistry classifies enzymes into an Enzyme Comission (EC) number, a four digit representation separated by periods (e.g. 2.4.2.17) that are based on the chemical reactions catalyzed. The four levels are related to each other in a functional hierarchy. The first number describes one of the 7 main enzymatic classes, the second describes the subclass, the third represents the sub-subclass and the fourth refers to the substrate of the enzyme [51].

### 3.1.2 Protein Data and Databases

Over the last years, the decrease of the cost of DNA sequencing has led to an exponential growth of available biological data such as protein sequences. Despite that, the experimental characterization of proteins is slow and requires sophisticated and expensive techniques, for example, X-ray crystallography, or functional assays. Thus, computational methods to predict protein function prediction gain special relevance [52].

The availability of data is one of the most important resources needed for ML and DL algorithms to perform well. Also, for accurate functional annotation, a correct protein sequence is a prerequisite. Widely accepted databases by the scientific community that incorporate accurate Protein information include:

- UniProt. The Universal Protein Resource is a database of protein sequences grouping together Swiss-Prot, TrEMBL and Protein Information Resource. It was Developed by the European Bioinformatics Institute (EMBL-EBI), the SIB Swiss Institute of Bioinformatics and the Protein Information Resource (PIR).[53]
- SWISS-PROT. SWISS-PROT is a database within UniProt, where the main objective is to provide curated protein sequences with a high level of information about the protein function, domains structure, post-translation modifications, and other features, with minimum redundancy level. [54]
- TrEMBLE is a computer-annotated protein sequence database from UniProt enriched with automated classification and annotation. This database is used to complement Swiss-Prot.[55]
- PROSITE database consists of entries describing the protein families, domains and functional sites, as well as amino-acid patterns and profiles in them. These are manually curated by a team of the Swiss Institute of Bioinformatics and tightly integrated into Swiss-Prot protein annotation. [56]
- The Pfam database provides a complete and accurate classification of protein families and domains. This database uses protein annotations and multiple sequence alignments generated using hidden Markov models [57].

## 3.2 Protein classification methods

Finding the function of a protein is important, not only to understand its role in its context, but also as a method to obtain valuable information useful in protein engineering and synthetic biology. Understanding the relationship between a protein sequence and its function could be used to induce mutations creating similar compounds with optimized or different functions, or even to create new pathways for production of valuable compounds.

The characterization of protein sequences and prediction of its function based on the amino acid sequence is a long-standing problem with important applications in several areas such as biomedical, industrial, and environmental.

Traditionally, proteins are classified according to their sequence or structure similarity with other proteins. Based on the family (group of proteins with similarities in sequence or structure) to which it belongs, the domains (functional and/or structural units in a protein) and sequence features (groups of amino acids that confer certain characteristics upon a protein) that comprise it, a protein can be compared to others and, additionally its function and properties can be predicted. [58]

The most common approach to classify a protein is to use alignment algorithms to search for homologous sequences against a large database, using tools like BLAST.

### *BLAST*

BLAST (basic local alignment search tool) is a set of tools that allows the comparison of primary biological sequence information with a library or sequence databases. The heuristic algorithm it uses is much faster than other approaches, such as for example calculating an optimal alignment. Using an amino-acid sequence of proteins or nucleotides we can identify library sequences that resemble the query sequence above a certain threshold. Using a BLAST program called BLASTp is possible to search protein databases using a protein query. The BLAST search can also be limited to a specific organism or taxonomic group.

However, to annotate a protein function based only in the protein sequence is limiting and error prone. For example, eliminating a single amino acid can change the function of a protein, divergent natural sequences occasionally have similar functions, and, beyond that, there is a large set of proteins without known homologous.

To answer this problem complex statistical models such as Hidden Markov Models (HMM) have been developed. Unfortunately in addition to being computationally intensive to run at scale, such approaches often lack generalizability.

### *HMMER*

HMMER is an example of algorithm that allows to make sequence alignments using probabilistic hidden Markov Models. HMMER is used to search databases for homologous sequences and is often used together with a profile database, such as Pfam.

Single sequences or database of sequences are compared to a profile-HMM to detect homology. Sequences that score significantly better to the profile-HMM compared to a null model are considered to be homologous to the sequences that were used to construct the profile-HMM. [59]

Compared to BLAST, HMMER is designed to detect remote homologs as sensitively as possible, and, therefore, are better in studies with evolutionary significance. Besides that, with HMMER, it is possible to use sequence queries, not just profiles.

### *Machine learning*

Traditional methods have been successful in many cases, but still fail when sequences have low homology with available characterized sequences. Besides this, alignment algorithms can have

a considerable computational cost. Therefore, the ability to predict the function of a protein from only its amino acid sequence remains one of the current challenges of biology [60].

Recently, DL has reached a level of attention and investment never seen before in the history of AI. ML and DL algorithms apply algorithms to uncover the underlying relationships within data and extract abstract and complex patterns, thus emerging as suitable approaches for large-scale protein analysis.

Nonetheless, due to the increase in the amount of biological data and new protein sequences, DL algorithms can be a solution to make the full connection between sequence and protein's function, by associating a large range of functional properties and not only the protein sequence [15].

In the last years, Deep learning models have been applied to complex protein classification problems with very good results, even though using solely the protein sequence. Examples include D-SPACE[61], DeepSeq[62], DeepFAm[52] and DeepGO[63] all applying convolutional neural networks, but also other approaches using RNNs [60] and DNNs [64].

In the next section, we will focus on the application of word embedding models to protein classification problems.

## 3.3 Word embeddings applied to proteins

This section intends to briefly describe the methodology to apply word embeddings to protein sequence data and make an overview of the research papers that implement word embeddings to protein classification tasks.

### 3.3.1 Apply word embeddings to protein sequence data

In 2015, Asgari *et al* [9], presented Protvec, the first paper describing the application of methods in NLP to gain a deeper understanding of the "language of life", specifically proteins.

The basic idea behind this concept is that protein sequences are a special language with an alphabet with only 20 symbols, corresponding to the 20 amino acids. Protein sequences can be segmented into constituent biological words, n-grams. For NLP, a meaning of a word is characterized by its context, i.e. neighboring words. In this way, word vectors with biological meaning are constructed. Word embedding techniques are feature extraction methods that can be then fed to ML and DL models to predict the protein function.

Applying Word Embedding methods for protein classification implies K-mer segmentation, training or downloading a word embedding model and obtaining sequence representations that can be fed to ML and DL models.

*K-mer segmentation*

The first step is to decompose a protein sequence into protein words. To do this, one must choose the size of the 'word' to be considered - k - and segment sequences into all the possible words. A schematic example can be seen in Figure 8.

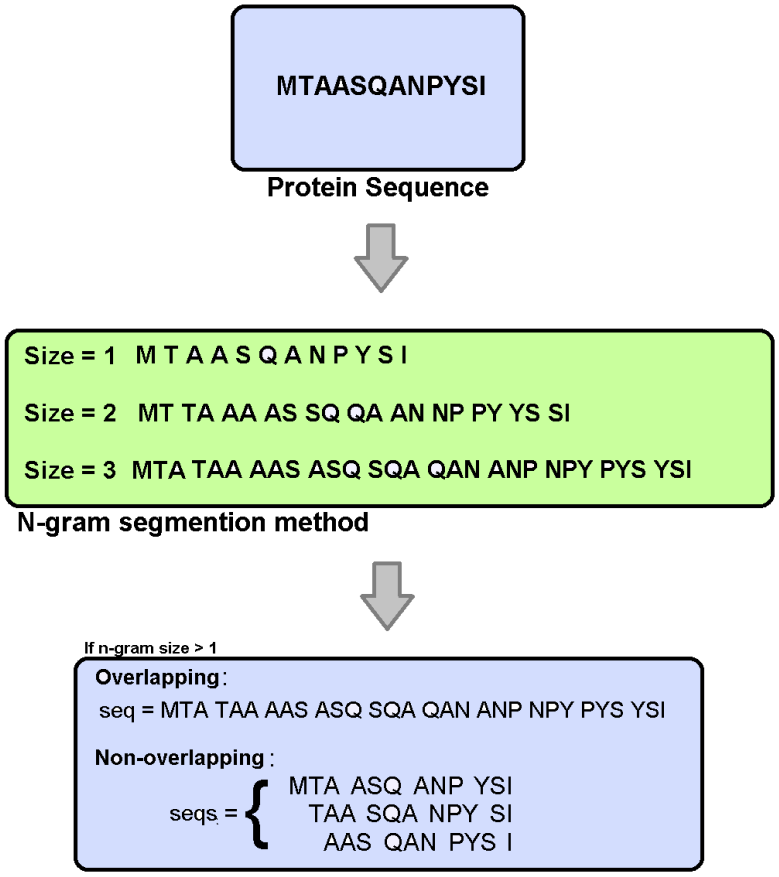



Figure 8 – Schematic representation of the process of segmentation of a protein sequence in biological words.

Each sequence is scanned from beginning to the end, one k-mer word at time. In this step, two parameters are crucial: the size of the protein word to consider and the overlapping. The size of the protein word is usually between 1 and 3. Larger words will produce embeddings with a higher complexity and will capture more relationships between aminoacids. However, those will also make heavier models which require lots of memory. The number of possible k-mers is obtained by the number of combinations between the aminoacids. If the k-mer is 1, there will have distinct 20 words in total, if the k-mer equals to 2 there will be 400 possible words, and if the kmer = 3, there will be 8000 possible words. With overlapping, the sequence is represented

by consecutive and overlapping windows of k size. In non-overlapping mode, a protein sequence is composed of k lists of shifted non-overlapping words.

All possible words constitute the vocabulary. All the protein sequences segmented in words constitute the corpus.

*Obtaining the word embedding model*

After the protein segmentation into $n$ grams, one can train a word embedding model to understand and capture meaningful representations of the words. Word2vec or Fasttex are usually trained using Skip-gram or CBOW algorithms. After this step, each word is represented by a vector of dimension $N$. The bigger the $N$, the more information the vector can capture, but more computational memory and processing are required [46–48]. A schematic representation can be viewed on Figure 9.

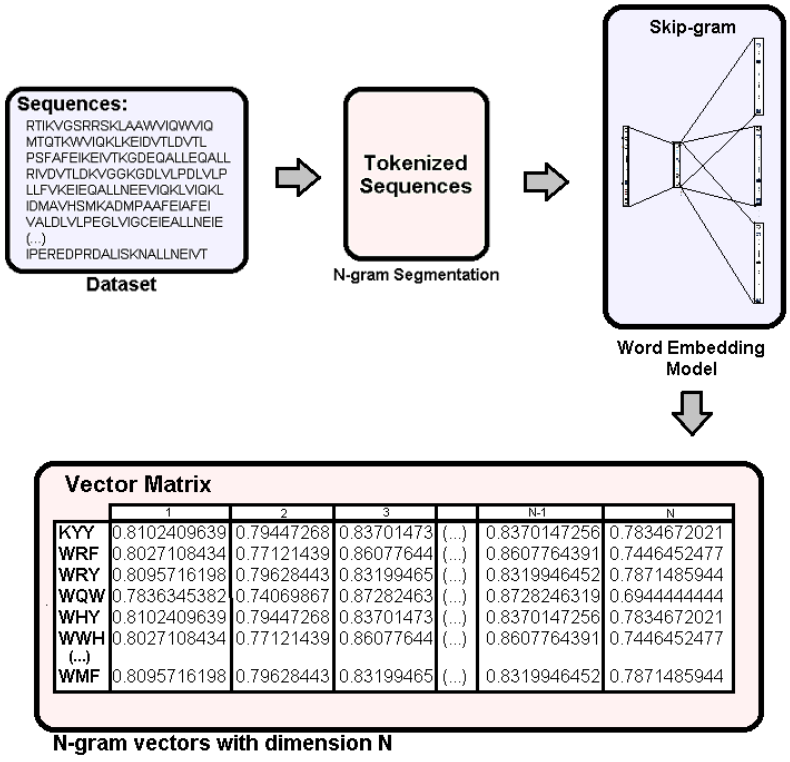



Figure 9 – Steps to train a Word embedding model and obtain a vector matrix of biological words.

Usually, word embedding models benefit from large corpora of sequences. In this way, it is common to train Word embedding models using large datasets, such as all the sequences available in databases like the Swiss-Prot [9]. These models are, however, very expensive to compute and, when a scientific group does it, the model is made available for the scientific

community. Therefore, as a alternative to training a model for a specific dataset, one can choose to download and use a pre trained model, such as Protvec, known as a process of transfer learning.

On the other hand, training in-house Word Embedding models gives more freedom in the choice of parameters such as the dimension $K$ of the words and the dimension $N$ of the vectors representing those words. Besides, language models trained on different datasets will capture different relationships between datasets, that can be useful in some biological problems.

*Obtaining protein representations based on word embeddings*

Lastly, the protein sequences are represented by the vectors learned in the previous steps. There are several ways to do this:

- Method 1

  Substitute directly the $n$-grams presented in the sequence by the WE vector. Being $K$ the dimension of the word and $N$ the dimension of the WE vector, a sequence of size $L$ will be represented by a final vector of $(L - k - 1) * N$ elements. This method preserves the spatial information of the location of biological words.

- Method 2

  In this method, $k$-mer word frequencies are calculated and multiplied by the corresponding WE vectors. A sequence, independent of the size, will be represented by a matrix of dimensions $Number\_of\_words * N$.

- Method 3

  All the vectors of Method 2 are summed to reproduce a single vector of dimension $N$.

Figure 10 displays a flowchart demonstrating the different feature extraction methods based on word embedding vectors. For simplicity, it is only represented the case of 2 sequences and biological word lengths equal to 3.

*Functional classification of proteins using ML and DL methods*

After the feature extraction is completed, the protein sequences represented as numerical features can be fed to ML models and DL networks to train models for protein classification. One can use the vectors as described above, or give the vectors as weights in embedding layers. These embeddings layers can be, if desirable, trained along the DL models for more fine tuned results. Figure 11 represents the whole pipeline ranging from the acquisition of WE vectors and of sequence vectors that are then fed to a DL model, for example a RNN.

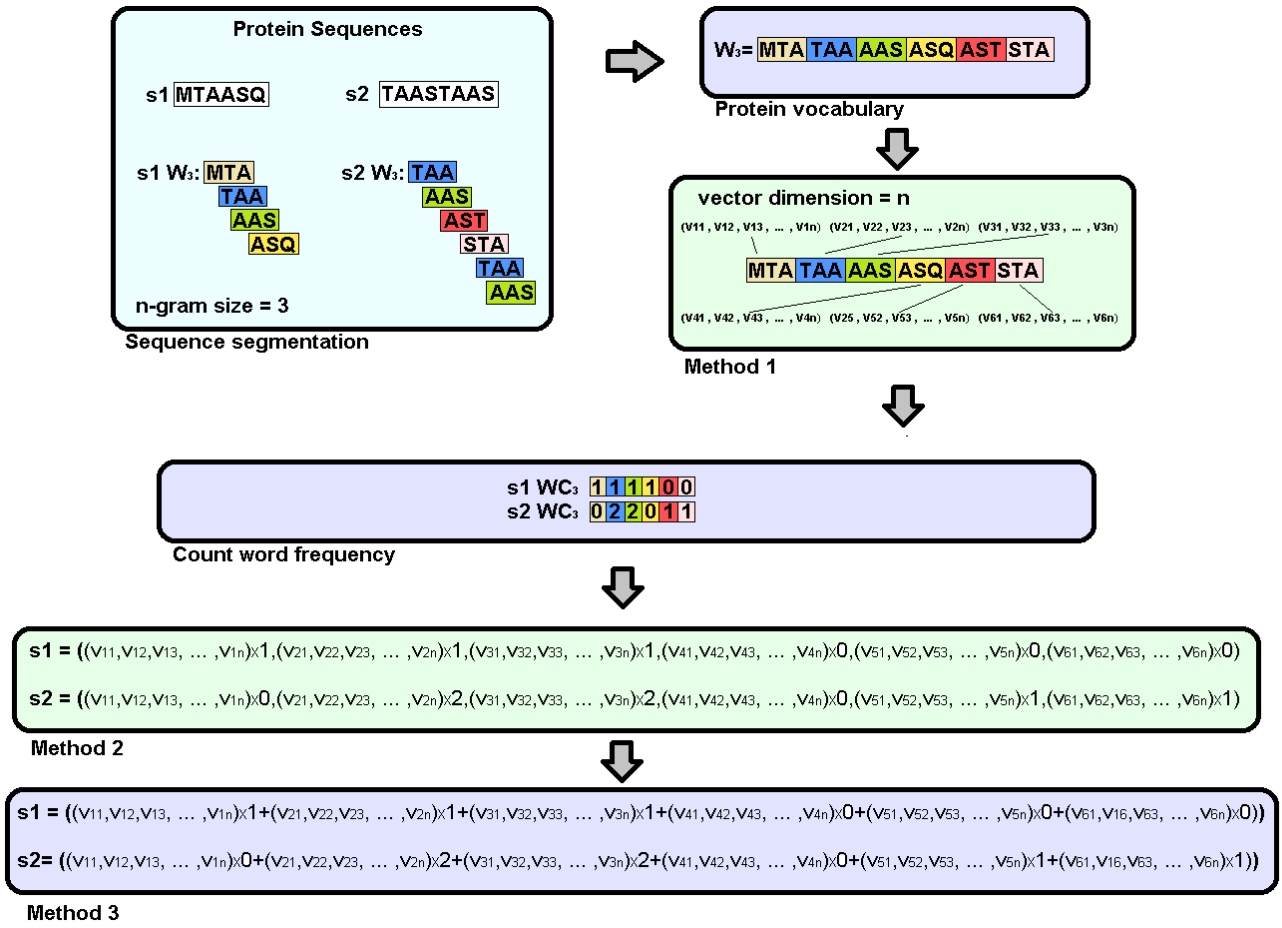



Figure 10 – Flowchart demonstrating the possible different methods for using word embedding vectors as protein features with 2 sequences and word length3. Adapted from [65].

### 3.3.2 Word embedding research applied to protein classification

In recent years, several tools using word embeddings have been created to assist the analysis of biological sequences. Many of them focus on proteins, but there are also studies on DNA/RNA classification. An overview of the recent papers that used WE to protein classification is presented in Table 1.

In a first instance, it is possible to notice that the WE complexity level is normally correlated with the type of architecture used for classification. When the WE vector has high dimension, the use of deep learning architectures, such as CNN and LSTM, becomes more frequent. This is the case of DeepGly [70] and the paper by Lee et al.[71]. On the other hand, when the complexity of the WE is low, the use of shallow machine learning architectures, like Support Vector Machine (SVM), Random Forest (RF) and K-Nearest Neighbors (KNN), is more common, as it is possible to see in the papers by Yang et al. [66] and Nguyen et al[65].

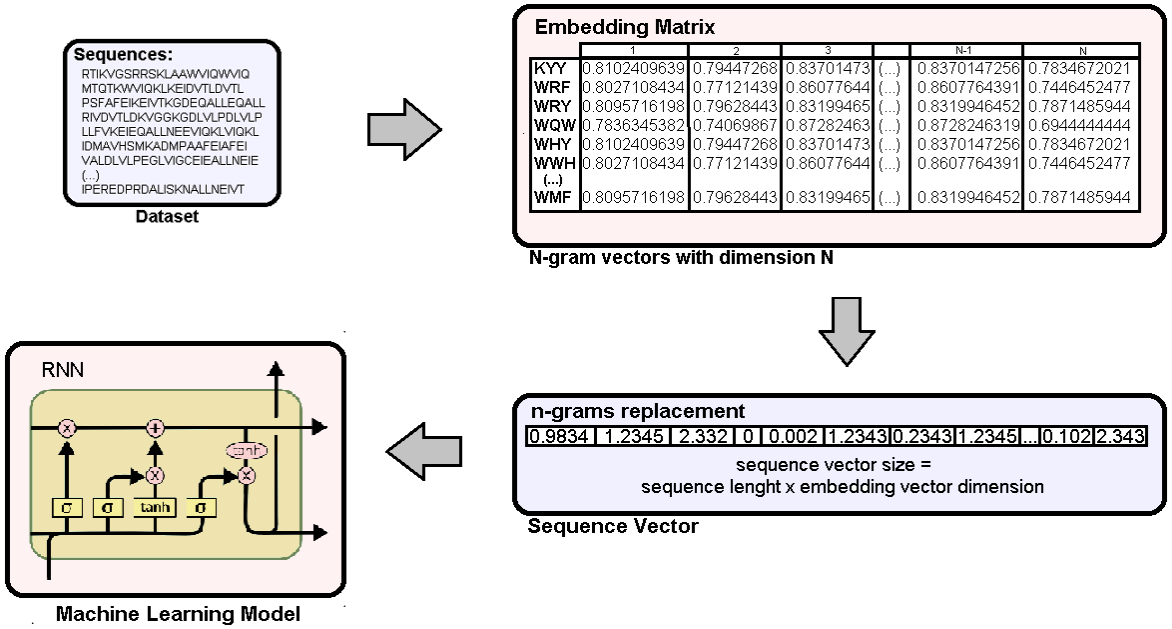



Figure 11 – Pipeline for classification using WE. An embedding matrix is obtained to represent Word vectors. The protein sequences are transformed into vectors and fed to DL models.

Table 1 – Word embedding research applied to biological sequence classification

| Paper | Seq Type | WE Model | WE Architecture | gram size | vector dimension | ML model | Ref | Data |
|---|---|---|---|---|---|---|---|---|
| ProtVec | AA / DNA | word2vec | Skip-Gram | 3 | 100 | SVM | [9] | 2015 |
| Yang k. *et al.* | AA | doc2vec | DBOW | 3 | 64 | GP regression | [66] | 2018 |
| NeuBI | AA | word2vec | Skip-Gram | 3 | 200 | GRU | [67] | 2018 |
| Sarker *et al.* | AA/Domains | FastText | CBow | 3 and Domains | NS | SVM kNN PNN | [68] | 2019 |
| fastSNARE | AA | FastText | Skip-Gram | 5 | 100 | CNN | [69] | 2019 |
| DeepGly | AA | word2vec | Skip-Gram | 3 | 100 | LSTM CNN | [70] | 2019 |
| Nguyen T. *et al.* | AA | FastText | Skip-Gram | 3 | 1 | SVM RF Naïve Bayes | [65] | 2019 |
| ubisites | AA | word2vec | Skip-Gram | 2 | 20 | CNN | [71] | 2020 |
| TNFPred | AA | FastText | NS | 2 & 3 gram | 1 | SVM | [72] | 2020 |
| INSP | AA | word2vec | CBOW | 3 | 100 | SVM | [73] | 2020 |
| RNA2vec pro2vec | RNA/AA | word2vec | Skip-Gram | 4-gram RNA 3-gram AA | 300 | LPI-Pred LR RF SVM | [74] | 2020 |
| DeepKcrot | AA | word2vec | Skip-Gram | 1 | 5 | RF CNN LSTM | [75] | 2021 |

The paper describing the INSP tool [73] uses 3-gram with 100 dimensional feature vector coupled with a SVM model. Furthermore, the authors stated that the size of the vector was not reduced in order not to lose important information. The use of an SVM model can be justified with the size of the dataset constituted by 290 proteins in total. The same applies to the paper by Yi et al. [74] also using a dataset with less than 5000 entries.

CNNs and RNNs are powerful networks to deal with sequential data due to its ability to capture patterns in sequential data. This makes these architectures suitable to deal with WE [76,

77]. Based on this feature, we can infer that, not only these architectures have good performance with great complexity vector representation, but also with WE with smaller vector dimension representations. Research articles that present DL architectures coupled with WE usually opted for CNNs and RNNs. These architectures, when directly compared to shallow algorithms, tend to yield better results. An example of this behaviour is demonstrated by Wei et al. [75].

WE vectors can become a burden due to excessive size and limit the suitable training. Usually, the solution goes by the use of lower $n$ gram values and vector dimensions. However, this can heavily impact the classification performance. Hamid et al [67] proposed using truncated Singular Value Decomposition (SVD) to cluster the $n$ grams and reducing the size of the vocabulary from 8000 trigrams to a vector of size 200. Sadly, the methods are poorly described, making them harder to replicate.

Another way to reduce the dimensionality is applying the method 3 of protein vectorization described above. The method 3 produces a protein vector of the same size as the word vector. This method was applied in the paper of Protvec [9]. As the final vector obtained has a dimension of 100, the papers using this method can use traditional shallow ML models.

We would like to highlight two papers. The paper developed by Chen et al [72] and the one by Sarker et al [68] as they present different $n$-gram segmentation strategies. Chen et al. [72] focused their work on finding a different methodology for identifying tumor necrosis factors given cytokine sequences. They employed a word embedding technique to create hybrid features. This method tested $n$-gram segmentation with length from 1 to 5 and combinations between the split grams. On the other hand, Sarker et al [68], instead of looking for different divisions of the size of words, made a comparison between sequence-based representations, with overlapping and non-overlapping modes, and domain-based representations.

Le et al [71], is the only one proposing to upload the weights of the WE into an embedding layer, instead of replacing the sequences with the generated vectors. This layer can be trained with the rest of the compiled model, and can have their weights readjusted.

# 4

# DEVELOPMENT

This section will present the description of the methods used to develop the module to perform Word embedding methods in protein classification problems. In addition, it will describe the integration of this module on the ProPythia package [12].

This Python module was developed for processing biological sequences aiming to search for semantic meaning in sequence "words" (such as nucleotides and amino acids). Although it can be used for both protein and DNA classification problems, it was only validated in proteins.

All the code developed in this thesis was written in python 3.9 using Pycharm professional 2021.1.1 as IDE. The environment was constructed using Anaconda platform and it uses the gensim 4.0.0 library as base, to manipulate the different NLP models and architectures. The other imported libraries used were pandas and numpy.

This module is available on GitHub https://github.com/ivanapg96/bumblebee.git. In addition, the GitHub page also contains the files and the code used to validate the module and the enzyme case study, that will be explained further ahead.

## 4.1 DEVELOPMENT OF THE PYTHON PACKAGE

To use the module, it is necessary to create a class object and call the desired methods. The user can set specific values for the majority of the parameters, but default values are established. Although it is built in only one module, it is organised in sub-modules so that the user can use them in different specific tasks and adapted to fit the problem that is working on. The submodules are:

1. **Read sequence sub-module**: To read and/or change sequences;
2. **Sequence processing sub-module**: To generate subsequences;
3. **Create vocabulary list sub-module**: To get all the vocabulary in the dataset;
4. **Training word embedding models sub-module**: To train the word embedding models;

5. **Load models list sub-module**: To get a pre-trained embedding model;
6. **Vectors sub-module**: To get a sequence vector or the matrix of vectors;
7. **Interpretability sub-module**: TO visualise word embeddings in space and get similarities between vectors.

### 4.1.1 Read sequence sub-module

The read sequence submodule contains methods to read and/or change sequences allowing to remove errant sequences and change/replace unwanted amino acids with their analogs. For example, the user can put asparagine (N) in the place of aspartic acid (D). This is specially important to replace non-relavant/not-common aminoacids and can simplify the vocabulary.

### 4.1.2 Sequence processing sub-module

This sub-module implements the segmentation of sequences by grams of size $n$. After tokenizing the sequence in $n$-grams, it is necessary to organise the sequence using a lapping method. This section contains two different lapping methods: the overlapping method, generating a sequence with $n$ times its size and the non-overlapping method, generating $n$ different sequences starting in the different indices of the sequence, as described above.

### 4.1.3 Creating vocabulary list sub-module

In order to train the different word embeddings and create the word embeddings matrix, we need to gather all the grams (words) present in all sequences. The methods present in this sub-module create a list of all grams of the dataset. As the search algorithm is slow, this module provides a method to fetch the list of $n$-grams taking into account their size, from a pre-created file. Also, if the user does not have the file with all the $n$-grams, she can use an implemented method with the purpose of creating the file.

### 4.1.4 Training word embedding models sub-module

The training sub-module is responsible for training the different word embedding models, using as a base the gensim library. It is possible to train Word2Vec and FastText models with both CBOW or skipgram algorithms. The training can be done with user-specified values for the parameters, but defaults are established for the majority of them. The default parameters defined

in the class are vector dimension, number of epochs, learning rate, min_count, and window of context. The min_count was defined as 1, in contrast with the value of 5 defined by gensim default. This was made to take into consideration all the possible protein words and to capture the maximum interactions between them.

Besides, human languages have significantly more words. For example, the English language contains 470 000 words. In this case, considering words with less than 5 occurrences would create noise, and a very sparse matrix and would increase significantly the computational time and resources needed. In contrast, a trigram-based protein language will have around 8000 words which makes it possible to account for all the possibilities. On the other hand, the protein datasets are limited and important words may be less frequent. These infrequently used words can also be very important for the protein classification as protein function is very affected by changes in aminoacids, for example, in the case of mutations. The sub-module always saves the embedding models in the ".model" format used by gensim, but can also save an embedding matrix in the CSV format if the user chooses to.

### 4.1.5 Loading models list sub-module

In this sub-module, it is possible to load a pre-trained word embedding model, either in '.model' or '.csv' format. After loading the model, the vector weights and the vector-matrix vocabulary are updated in the *init* definitions of the class.

### 4.1.6 Vectors sub-module

Whether training a model or using a pre-trained model, one of the main objectives of the presented module is to create a vector representation of a sequence. With this sub-module, we are able to obtain a vector for a given $n$-gram and the number of occurrences of that $n$-gram. Three methods of representing sequences as vectors are implemented as described in section 3.3.1.

- The first method is the direct replacement of n-grams by vectors present in the vector matrix. This method returns a vector with a size equal to the length of the sequence times the dimension of the vector of the embedding matrix.
- The second method uses the method of counting $n$-grams to later multiply the number of occurrences of $n$-grams by the matching vectors present in the matrix and return the matrix.

- The third method is a method similar to the second but in the end sums all the vectors multiplied by the occurrences of the n-grams. This method returns a vector of the size of the matrix vectors.

In the case of using the first or the second method, the user can choose if the returned vector is displayed in a flat array or in a matrix array.

### 4.1.7 Interpretability sub-module

Finally, this sub-module intends to apply interpretability techniques that can shed a light on understanding the model behavior and separation of the $n$-grams in space. This way, the user can obtain important information in the search for learning patterns of the models or biological meaning captured in the word embeddings.

The sub-module allows training the dimensional reduction algorithm t-SNE creating different plots related to different physicochemical properties.

The biological features available are charge, volume, mass, Van der Waals Volume, polarity and hydrophobicity. These characteristics are defined in the literature for individual amino acids. Following the approach of Asgari et al. [9], in the case of $n$-grams larger than 1, a mean of these characteristics will be presented. However, a simple average of physicochemical characteristics may not be enough to distinguish different $n$-grams. For example, a trigram 'TAA' would have the same values as 'AAT', even if they behave differently. For that reason, an additional characteristic, the binding free energy was added for trigrams, the most usual case. The binding free energy values were defined experimentally by Chen et al. [78]. If the user wants, she can also define other characteristics.

Additionally, the models can also retrieve scores of similarity and neighborhood of the $n$-grams also helping in understanding possible reasons for vector similarity.

Figure 12 provides an overview of all the functionalities and pipelines of the proposed module.

## 4.2 Integration with ProPythia

ProPythia [12] is a generic Python-based platform to apply ML and DL pipelines to protein classification problems developed within the Biosystems group at CEB/ U. Minho. It has several modules to do different parts of the protein classification pipeline including modules to generate and calculate protein features and aminoacid encodings; modules to run unsupervised methods, dimensionality reduction and visualization techniques such as PCA or TSNE algorithms, and models to run supervised methods that allow to train, validate, test and interpret ML and DL

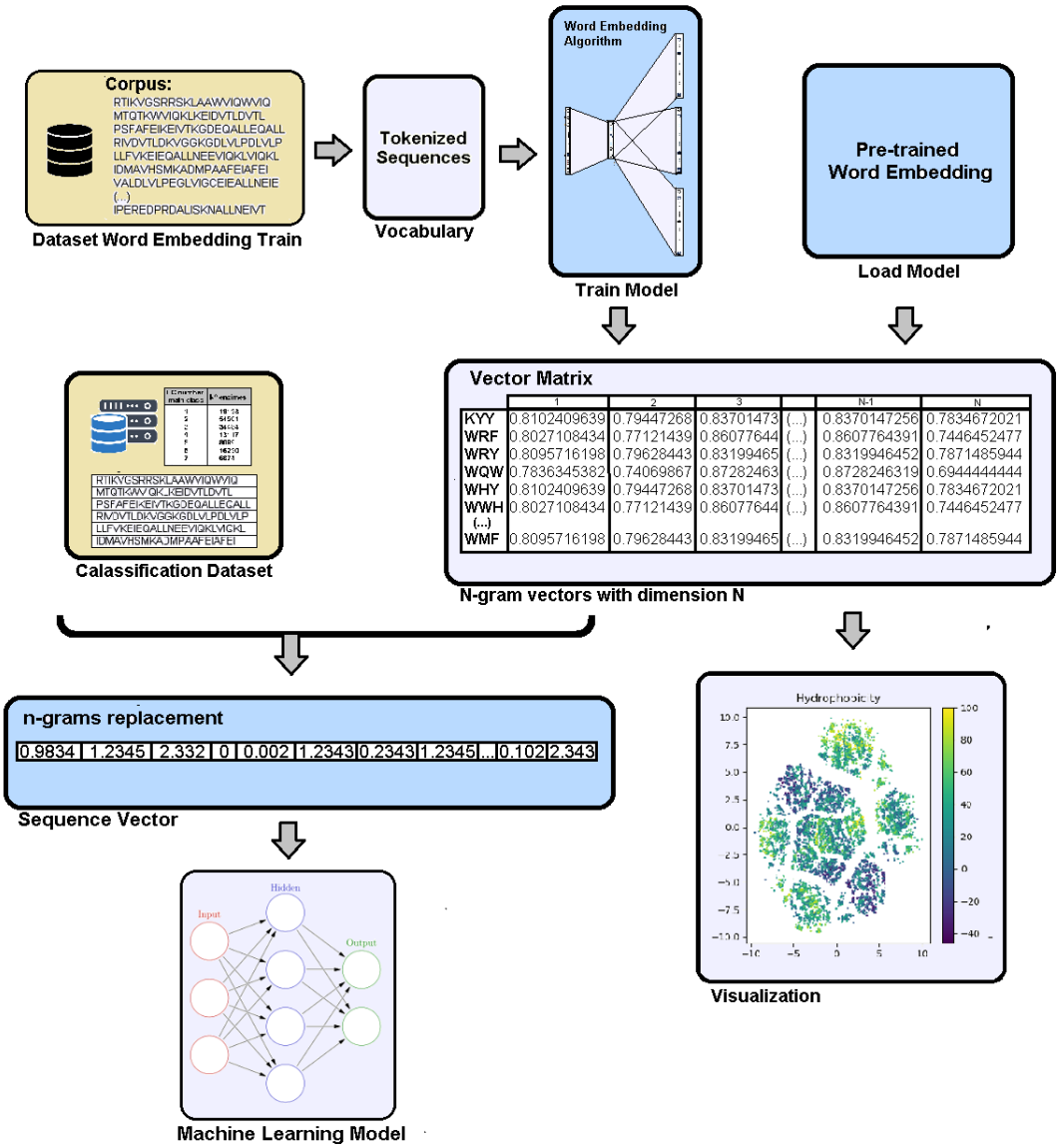



Figure 12 – Overview of all the functionalities and pipelines of the proposed Word embedding module

models. For protein representation, ProPythia allows calculating several protein descriptors and encoding the protein sequence using one-hot encoding or other biological matrices such as the BLOSUM substitution matrix or Z-scale values [12].

As mentioned in section 1.1, one of the objectives was to extend the protein representation methods available to include WE techniques capable of efficiently representing a protein sequence. To accomplish this, a module was developed that can be imported by other modules to create protein sequence vectors.

As ProPythia is built in a modular way, the integration of this WE model allows someone to get WE protein representations and train ML and DL models already developed in the package.

Alternatively, a ProPythia user is now also able to get WE representations independently of the rest of the package. Moreover, methods for protein preprocessing were also added to the already existing preprocessing sequence ProPythia module.

Figure 13 illustrates how the WE module developed was integrated into ProPythia. The colored modules were the new ones implemented and the gray ones were already developed in ProPythia.

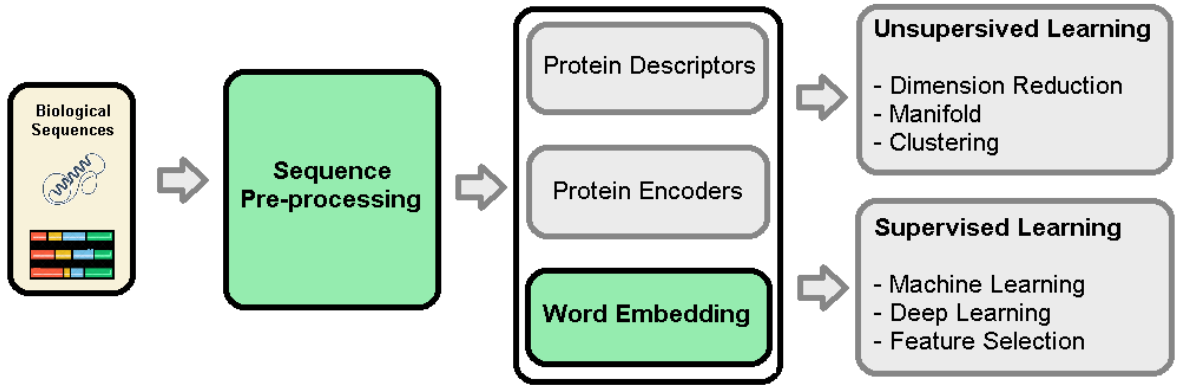



Figure 13 – Schematic overview of the integration of the developed WE module in ProPythia

ProPythia package, with the WE module already available, is available on GitHub https://github.com/BioSystemsUM/propythia.

# 5

# VALIDATION

This section will present a comparative analysis to demonstrate the application and performance of the developed module for addressing sequence-based prediction problems. The comparative analysis will be made with two test cases: plant ubiquitylation sites [71] and lysine crotonylation sites [75, 79]. The case studies were chosen based on the availability of datasets and capability of replication. The DL models were constructed using the Tensorflow library.

## 5.1 IDENTIFICATION OF PLANT UBIQUITYLATION SITES

Protein ubiquitylation is one of the most important posttranslational modifications (PTM) and plays an essential role in the regulation of physiological mechanisms involved in diverse biological processes in plants. Aiming to find the differences in ubiquitylated patterns of different species annotated in the Protein Lysine Modifications Database (PLMD), Le et al [71] proposed a sequence-based characterization of ubiquitylated substrate sites.

To assess the predictive ability of our package to address sequence-based prediction problems, we replicated the paper's experiments which are described to highly perform on ubiquitylation sites prediction.

We used the plant ubiquisites dataset provided by the paper [71], constituted of 7000 entries, 3500 positive entries, and 3500 negative. All the sequences have 31 aminoacids. As described in the paper, a word2vec model embedding model with Skip-Gram architecture was trained. The sequences were tokenized to 2-gram words and the resulting vectors of the WE were of dimension 20. Due to the lack of information on the parameters used in word2vec training, the default parameters were used whenever parameters were not described. For example, the Skipgram model was trained for 5 epochs. To build the DL model, the matrix values were given as weights to an embedding layer with the trainable parameter set as True. The DL model was composed of 3 consecutive CNN units with 128 units and kernel size 3. Each of these units was followed by a max pooling layer with kernel size 2 and a dropout layer of 0.5. Finally, a dense layer with 128 units is followed by the final classification layer with 1 unit and a sigmoid

activation function. All the previous layers have ReLu activation. The model was trained using the RMSprop optimizer, 120 epochs, and 500 batch size.

The dataset was divided into train and independent test sets with a division of 80% - 20% of the sequences, respectively. For training, we further divided the training set into training and validation sets. The validation set is used to evaluate model performance during training. the division was also 0.8/0.2. The scores presented are over the independent test set. Accuracy, Matthews Correlation Coefficient (MCC), log loss, F1 score, Receiver Characteristic Operator (ROC)-AUC, precision and recall were calculated. However, the paper only calculates Accuracy, F1 score, precision, and recall.

As shown in table 2, mimicking the published model (second row) with our software tool obtained 0.753 of accuracy and 0.754 of F1 score, values very similar to the ones described in the original paper (first row).

An alternative model was also built to try to improve the obtained scores. The embedding layer was set as non-trainable and the more common Adam optimizer was used. Callbacks of early stopping (patience of 50) and Reduced learning rate (patience of 20 epochs, reduce factor of 0.2 and initial learning rate of 0.01) were also added. The maximum number of epochs available to train was set to 300 and the batch size was 32. This model obtained 0.778 of accuracy, 0.556 of MCC, and 0.783 F1 score (third row of table 2), higher scores than the ones reported by the paper.

Furthermore, we tested representing the protein sequences with a simple one-hot encoding instead of WE. The one-hot encoded protein representations were fed to a model similar to the proposed one. The one hot encoded model (fourth row of table 2) achieved 0.787 accuracy, 0.577 MCC, and an F1 score of 0.798, yielding slightly better results.

Table 2 – Summary of the scores of the models produced with the package compared to the ones described in [71]

| Model/Scores | Acc | MCC | log loss | f1 score | ROCAUC | precision | recall |
|---|---|---|---|---|---|---|---|
| Paper [71] | 0.756 | - | - | 0.749 | - | 0.733 | 0.767 |
| [71] Replica | 0.753 | 0.506 | 0.513 | 0.754 | 0.753 | 0.751 | 0.758 |
| Proposed alteration | 0.778 | 0.556 | 0.532 | 0.783 | 0.778 | 0.765 | 0.801 |
| One-hot encoding | 0.787 | 0.577 | 0.537 | 0.798 | 0.787 | 0.759 | 0.845 |

The model mimicking the one described in [71] yielded slightly different results, achieving a higher f1 score but less accuracy. This difference, however, is marginal and can be due to the random initialization of the models or the use of default parameters for the ones that were not specified. This result shows that the package can be used to build models, as it performs similarly to the ones described in the literature.

Changing the embedding layer to trainable False and adding callbacks improved the scores of the model. The best model was achieved using one-hot encoding, which, for this particular problem may be more suitable than using WE. However, the differences were in fact very small and a more in-depth analysis with more models would be needed to assess which type of encoding would be the best.

## 5.2 Lysine Crotonylation Site Prediction - DeepKcrot

Protein lysine crotonylation (Kcr) is a PTM, originally identified at histone proteins, that regulates various activities. Identification of Kcr sites through wet lab experiments is expensive and time consuming and, therefore, the development of efficient computational prediction approaches is necessary.

To have a second comparative analysis with a different dataset, the module built was used to mimic the models developed by the group of Lei Li firstly in 2020 [79] and then improved in 2021 [75]. The papers gather Kcr sites available for human proteins and use a convolutional neural network with WE to distinguish Kcr sites from non-Kcr sites.

The datasets used were the ones provided by the paper on the GitHub page (https://github.com/linDing-group/Deep-Kcr). The training dataset contains 6975 positive sequences and the same number of negative sequences, while and the test dataset includes 2989 positive sequences and an equal number of negative ones. Note that the datasets provided do not coincide with the ones described in the paper, with a severe discrepancy relative to the negative case studies reported as 67000 on the training and 16000 on the test set. We opted to use the datasets provided on the GitHub page as no other data was available.

As described in the paper, a word2vec model with an n-gram of size 1 (each AA is considered a word) and vector dimension of 5 was trained. As the authors do not provide further information, we used the skip-gram as the WE algorithm trained for 5 epochs. The protein representation is obtained by a direct replacement of the AA by vectors present in the vector-matrix. The word embedding layer is then fed to a DL model composed of 3 sequential convolutional units with 128 units each, kernel size of 8, and Relu activation. Each CNN is also followed by a max pooling layer with kernel size 2. Finally, a dense layer with 128 units and Relu activation and a final dense classification layer with 1 unit and sigmoid activation perform the binary classification.

The network was trained with Adam optimizer with a learning rate of 0.001 and binary cross-entropy as the loss function. An Early stop callback was defined with a patience of 50 epochs and the network was trained for a maximum of 500 epochs.

For evaluation, a ten-fold cross-validation test was performed using the training data. Furthermore, MCC, ROC-AUC, Sensibility (SN) and Specificity (SP) were also calculated for

the independent test set. Table 3 describes the scores for both CV and independent test set for KCR paper and the replica.

Table 3 – Summary of the scores of the WE and a CNN model produced with the package compared to the ones described in [75]

| | Model | Accuracy | MCC | ROCAUC | sn | sp |
|---|---|---|---|---|---|---|
| CV | CNN Kcr | 0.871 | 0.342 | 0.864 | 0.537 | 0.9 |
| | CNN replica | 0.813 | 0.627 | 0.813 | 0.842 | 0.784 |
| Test | CNN Kcr | 0.869 | 0.338 | 0.861 | 0.524 | 0.9 |
| | CNN replica | 0.825 | 0.650 | 0.825 | 0.850 | 0.799 |

As shown in Table 3, the models constructed with the package yield slightly less accuracy, specificity, and ROC-AUC and significantly more MCC and sensitivity. This can be explained by the differences in the dataset as the ones described in the paper has significantly more negative examples.

Additionally, the authors developed a classifier based on LSTM with word embedding. The word embedding layer remains the same, but it is fed to a network composed of an LSTM layer with 32 units with Relu activation followed by a dense with 16 units also with Relu activation and a final classification dense with one unit and sigmoid activation. The training and evaluation were the same as described above. Similarly to the CNN model, the replica of the LSTM yields slightly less accuracy, ROC AUC, and specificity and significantly more MCC and sensibility. As above, we envision that the discrepancy is related to the differences in the datasets. A detailed description can be found in Table 4.

Table 4 – Summary of the scores of the WE and an LSTM model produced with the package compared to the ones described in [75]

| | Model | Accuracy | MCC | ROCAUC | sn | sp |
|---|---|---|---|---|---|---|
| CV | LSTM Kcr | $0.863 \pm 0.001$ | $0.294 \pm 0.009$ | $0.839 \pm 0.015$ | $0.458 \pm 0.010$ | $0.9$ |
| | LSTM | $0.788 \pm 0.010$ | $0.789 \pm 0.009$ | $0.789 \pm 0.025$ | $0.803 \pm 0.026$ | $0.77 \pm 0.032$ |
| | GRU | $0.803 \pm 0.005$ | $0.607 \pm 0.011$ | $0.804 \pm 0.005$ | $0.825 \pm 0.005$ | $0.782 \pm 0.014$ |
| | biLSTM | $0.802 \pm 0.008$ | $0.605 \pm 0.016$ | $0.802 \pm 0.008$ | $0.816 \pm 0.015$ | $0.789 \pm 0.021$ |
| | biLSTM+att | $0.780 \pm 0.008$ | $0.569 \pm 0.013$ | $0.781 \pm 0.008$ | $0.826 \pm 0.08$ | $0.736 \pm 0.077$ |
| Test | LSTM Kcr | 0.86 | 0.306 | 0.839 | 0.524 | 0.9 |
| | LSTM | 0.810 | 0.621 | 0.810 | 0.837 | 0.783 |
| | GRU | 0.803 | 0.610 | 0.803 | 0.861 | 0.745 |
| | biLSTM | 0.810 | 0.620 | 0.810 | 0.832 | 0.787 |
| | biLSTM+att | 0.784 | 0.577 | 0.784 | 0.873 | 0.694 |

To test if altering the LSTM to another recurrent layer would improve the performance, models with GRU, bidirectional LSTM, and bidirectional LSTM were tested. In each case, the original LSTM layer was replaced and all the other parameters remained the same. Results

for both CV and independent test set can be found in Table 4. The scores achieved are very similar and none of the alternatives tested yield better results than an LSTM layer. Besides, as described in the paper, the model with CNN perform better than the one using RNN.

Overall, this comparative analysis evidences the performance and validates the module here described.

# 6

# ENZYME CLASSIFICATION CASE STUDY

After validating the module on Ubiquitylation Sites and Lysine Crotonylation Site prediction, the module was applied to the classification of enzymes. This chapter will present the analysis pipeline and the description of the methods used to create the vectors of the enzyme sequences, the results obtained and the analysis and discussion of results both from a computational and biological point of view.

## 6.1 METHODS

The methods used for dataset construction and preprocessing, Word embedding, ML and DL model development, optimization and evaluation, and the application of these models to predict the main class of EC number, are described below. The tensorflow, numpy and system seed were set to have reproducible results.

### 6.1.1 DATASET

The dataset used was retrieved from [80]. This dataset contains 152048 enzyme sequences and 22708 non-enzyme sequences belonging to the Uniref90 dataset, this is, only contains sequences with less than 90 % similarity. The dataset contains the EC numbers belonging to the 7 main classes. Enzymes with multiple EC numbers were only maintained if the main class coincides. The distribution of sequences across the 7 enzyme classes can be seen in table 5. Although EC numbers have 4 different levels, in this work only the first class is considered. Although for the training of the WE models, enzymes (EC class 1-7 in table 5) and non enzymes were considered, for the DL classification models, only enzyme sequences are considered. This is, all of the classification models described below are considering EC main classes from 1 to 7.

The enzyme sequences have different lengths. However, ML and DL models required the same shape input. Therefore, a length threshold is defined, and sequences longer than

Table 5 – Distribution of sequences across the 7 main enzyme classes

| EC class | Counts |
|---|---|
| non-enzymes | 22708 |
| 1 | 19066 |
| 2 | 54343 |
| 3 | 34484 |
| 4 | 12924 |
| 5 | 8081 |
| 6 | 16290 |
| 7 | 6860 |

the threshold were truncated at the end of the sequence. For the sequences shorter than the threshold, a dummy amino acid (X) is added. The length threshold defined in this work for the enzyme sequences was 500. Accordingly to Sequeira and Rocha [80], this length is capable of capturing the majority of enzyme information.

### 6.1.2 Word Embedding

To evaluate the use of WE, pretrained and trained WE models were tested. Two pre-trained WE models were used. The first one, Protvec [9] uses W2V with SG to train 3-grams with vector dimension of 100 and it was trained on the SwissProt database that contains all the curated proteins. The second pre-trained model applied was the TransformerCPI [81], a model similar to the Protvec but trained on all the human proteins present on the Uniprot. Both matrices were downloaded with the link provided in the papers.

The in-house model was trained using the module developed that is based on the gensim library. The base WE was trained using a W2V and a SG algorithm on the enzyme dataset, considering 3-grams and generating vector of size 100. A minimum count of words of 1 and, window size of 5 and 5 epochs were considered. All the parameters not specified were kept as the default ones. Throughout the development of this case study, several different combinations of parameters to try to improve scores were tested: number of epochs, size of n-grams, size of the reproduced vector dimension, model, algorithm and the dataset used to train the WE. The details of these models are described below along the results obtained.

With the different WE matrices, three different protein representation methods were tested. The methods were applied using the developed module and were as described in section 4.1.

The resulting protein vectors were fed to Machine and Deep learning models. Method 1 was tested using RNN and DNN, method 2 and 3 were tested with DNN and shallow ML. The parameters for the model training and evaluation are described below.

### 6.1.3 Machine Learning models

For training ML models the dataset was divided into training (67%) and test (33%). The division was stratified to have the same proportions of classes on both datasets.

The training dataset was used to perform a grid search cross validation with 3 folds and the best model was retrieved. To evaluate the model, Accuraccy, MCC, F1 score weighted, ROC AUC One vs Rest (OVR), ROC AUC One vs One (OVO), precision, recall, sensitivity and specificity of the predictions made by the model on the test dataset were calculated.

The models were used as defined in the scikit-learn and the Propythia package. The models SVM, RF, Stocastic Gradient Descendent (SGD), KNN, Logistic Regression (LR) and Gaussian Naive Bayes (GNB) were tested.

Table 6 describes the hyperparameter grid used for each of the seven machine learning models tested.

Table 6 – Hyperparameter values used in grid search for the models: SVM, RF, KNN, SGD and LR

| **Model** | **Parameter grid** | |
|---|---|---|
| SVM | C:<br>kernel:<br>gamma: | [0.01, 0.1, 1.0, 10]<br>['rbf']<br>['scale', 0.001, 0.0001] |
| RF | estimators:<br>features:<br>bootstrap':<br>criterion: | [10, 100, 500]<br>['sqrt', 'log2']<br>[True]<br>["gini"] |
| KNN | neighbors:<br>weights:<br>leaf size: | [2, 5, 10, 15]<br>['uniform', 'distance']<br>[15, 30, 60] |
| LR | C:<br>solver: | [0.01, 0.1, 1.0, 10.0]<br>['liblinear', 'lbfgs', 'sag'] |
| SGD | loss:<br><br>early_stopping:<br>alpha:<br>validation fraction:<br>n iter no change': | ['hinge', 'log', 'modified_huber', 'perceptron']<br>[True]<br>[0.00001, 0.0001, 0.001, 0.01]<br>[0.2]<br>[30] |
| GNB | var smoothing: | [1e-12, 1e-9, 1e-6] |

### 6.1.4 Deep Learning models

For training DL models, the dataset was divided into train and test with a split ratio of 0.1. The division was stratified to have the same proportions of classes on both datasets. Furthermore, to train DL models, the train dataset was further divided into 0.8 train and 0.2 validation.

Architectures composed of RNN followed by Dense layers, and architectures with only dense layers were built. RNN were tested using GRU, LSTM, bidirectional LSTM layers. Furthermore, attention mechanisms, as defined in [80], were also added to the networks using LSTM layers. The tensorflow library was used to construct the DL models.

Parameter optimization for the number of layers, units, Dropout (DR) and batch size was performed to achieve a better consensus model. However, due to computational limitations, this parameter optimization was not made using a grid search approach.

All the models were run with Adam optimizer, Relu activation (except the output layer with softmax activation), sparse categorical crossentropy loss function, reduce learning rate (patience of 20 epochs, reduce factor of 0.2 and initial learning rate of 0.01) and early stop (patience = 50) callbacks. The models were run for a maximum of 500 epochs. Accuracy, MCC, F1-score weighted, ROC AUC OVR, ROC AUC OVO, precision and recall were calculated for the test set.

## 6.2 Results and Discussion

The results and discussion are presented on this section. Training WE with different parameters, comparison of different vector methods and interpretability and visualization of WE are addressed.

### 6.2.1 Train of WE with different epochs

First, a Word2vec model with the skip-gram algorithm was trained considering 3 grams and generating a vector of size 100. These parameters were chosen to make it easy to compare with the Protvec model, the reference when considering WE for proteins. We trained different models to test the impact of increasing the epoch number. Models with 5 (default value), 10, 15, and 25 epochs were tested. A window size of 5 and min count of 1 was defined.

Furthermore, to assess if training the WE with just positive sequences would be better than training a joint dataset with positive sequences, the models with 5 and 10 epochs were tested using the full dataset described in table 5 or trained with all the 152048 positive sequences.

The WE models were then used to represent the positive proteins (each overlapping trigram was substituted by the WE vector) and fed to a neural network with 3 dense layers with 1024

units, with Relu activation and 0.3 dropout rate. The final layer was constituted of 7 units with softmax activation to predict the enzyme main class. The results of this comparison can be seen in table 7.

Table 7 – Comparison results of WE trained with different number of epochs and datasets containing only positive sequences and both negative and positive sequences.

| Dataset | epoch | Acc | MCC | F1 | ROCAUC OVR | ROCAUC OVO | precision | recall |
|---|---|---|---|---|---|---|---|---|
| w/ negative | 5 | 0.685 | 0.590 | 0.684 | 0.894 | 0.906 | 0.691 | 0.685 |
| positive | 5 | 0.758 | 0.688 | 0.757 | 0.934 | 0.944 | 0.760 | 0.758 |
| w/ negative | 10 | 0.757 | 0.688 | 0.757 | 0.932 | 0.940 | 0.759 | 0.757 |
| **positive** | **10** | **0.766** | **0.698** | **0.765** | **0.935** | **0.944** | **0.768** | **0.766** |
| positive | 15 | 0.755 | 0.687 | 0.754 | 0.933 | 0.942 | 0.756 | 0.755 |
| positive | 25 | 0.763 | 0.694 | 0.761 | 0.932 | 0.941 | 0.766 | 0.763 |

The model achieving better results is the one built with only enzyme sequences and run with 10 epochs. The second model, yielding close results is the one built with 25 epochs. The results with worst results are the ones with 5 epochs, especially the one that contains negative sequences. The epochs parameter controls the number of epochs for model training and refers to how many times the model will loop over the data. The results appeared to agree with the thesis that bigger datasets require 5 or fewer epochs. In this case, as the dataset is relatively small with few examples for the model to learn, it benefits from being trained for more epochs [47].

Furthermore, including only positive enzyme sequences appears to be better when compared to corpora with both positive and negative sequences. Different corpus capture different semantic relationships between words. For this particular case, using only positive sequences can be advantageous to improve the enzyme classification task.

Throughout this thesis, this model with 3-gram word2vec and 100 dimension was used.

### 6.2.2 Comparison of different vector methods

In this subsection, the three different vector-producing methods were tested. As the methods return different types of data, they were trained with suitable DL and ML models. When possible, in-house WE trained models were compared to the Protvec, the reference protein WE model.

*Protein representation - Vector method 1*

As previously referred, the first method of protein representation is a direct substitution of the n-grams presented in the sequence by the WE vector. This method preserves spatial information

regarding the location of biological words and therefore is usually coupled with RNNs, as these networks excel at extracting information of sequential data.

Models with one layer of RNN with 0.3 DR followed by 3 dense layers with 128, 64, and 7 (output layer) with 0.1 DR each were built. GRU, LSTM, biLSTM were tested. BiLSTM and LSTM were further tested with the addition of an attention mechanism. Parameter optimization for the number of layers, units, DR and batch size was performed to achieve a better consensus model. GRU, LSTM, LSTM with attention and biLSTM with attention had 256 units whereas the biLSTM models were run with 512 units. All the models were run with Adam optimizer, Relu activation (except the output layer with softmax activation), sparse categorical crossentropy loss, reduce learning rate (patience of 20 epochs, reduce factor of 0.2 and initial learning rate of 0.01) and early stop (patience = 50) callbacks. The models were run for a maximum of 500 epochs. Accuracy, MCC, f1 weighted, ROC AUC OVR, ROC AUC OVO, precision and recall were calculated for the test set.

The best architectures defined for each RNN type are as follows:

- 1 layer GRU with 256 units and DR 0.3 followed by 2 DNN with 128 and 64 and DR 0.1. Batch size=512
- 1 layer LSTM with 256 units and DR 0.3 followed by 2 DNN with 128 and 64 and DR 0.1. Batch size=128
- 1 layer BiLSTM with 512 units and DR 0.3 followed by 2 DNN with 128 and 64 and DR 0.1. Batch size=128
- 1 layer LSTM with 256 units and DR 0.3 with attention followed by 2 DNN with 128 and 64 and DR 0.1. Batch size=256
- 1 layer BiLSTM with 256 units and DR 0.3 with attention followed by 2 DNN with 128 and 64 and DR 0.1. Batch size=512

For each of the 5 architectures described, we tested to represent the proteins using the Protvec matrix or the WE model defined in the previous subsection. Both Protvec and the trained model use Word2vec skip-gram with 3 gram and vectors of dimension 100. The main difference is that the trained model is trained solely on the positive enzyme dataset, while the Protvec is trained using all of the Uniprot/SWiss-Prot [9]. This way, 10 models were tested. The detailed results of the best models obtained can be found in table 8.

In addition of the in-house WE model and the Protvec, the TransformerCPI was tested. TransformerCPI [81] is also an embedding trained with Word2vec with 3 grams and 100 dimension. The main difference to the Protvec is that is trained using human proteins of Uniprot. However, the results obtained with this technique were significantly lower from the ones obtained with the Protvec and the trained model. One possible explanation for the bad performance of TransformerCPI may lay on the fact that this model is only trained in human proteins, a very

Table 8 – Comparison results of trained WE and Protvec with different RNN layers

| | Acc | MCC | F1 | roc_auc ovr | roc_auc ovo | Precision | Recall |
|---|---|---|---|---|---|---|---|
| GRU ProtVec | 0.880 | 0.849 | 0.882 | 0.984 | 0.986 | 0.892 | 0.880 |
| GRU | 0.891 | 0.865 | 0.893 | 0.989 | 0.990 | 0.908 | 0.891 |
| LSTM ProtVec | 0.885 | 0.852 | 0.885 | 0.983 | 0.985 | 0.890 | 0.885 |
| LSTM | 0.883 | 0.850 | 0.883 | 0.983 | 0.985 | 0.889 | 0.883 |
| biLSTM ProtVec | 0.897 | 0.870 | 0.898 | 0.987 | 0.989 | 0.902 | 0.897 |
| biLSTM | 0.863 | 0.825 | 0.862 | 0.978 | 0.981 | 0.868 | 0.863 |
| LSTM att ProtVec | 0.900 | 0.874 | 0.900 | 0.990 | 0.992 | 0.908 | 0.900 |
| LSTM att | 0.874 | 0.839 | 0.874 | 0.982 | 0.985 | 0.875 | 0.874 |
| biLSTM att ProtVec | 0.923 | 0.902 | 0.923 | 0.992 | 0.993 | 0.924 | 0.923 |
| **biLSTM att** | **0.931** | **0.912** | **0.931** | **0.994** | **0.994** | **0.933** | **0.933** |

limited universe when considering enzymes. For simplicity reasons we opted not to show it in the table.

The best models were achieved using BiLSTM (1 layer 256 units) with an attention mechanism. In this particular case, using the WE trained performed better than using the Protvec. LSTMs perform overall better than GRU and are outperformed by the bidirectional LSTM. However, adding an attention mechanism improves both the results of LSTM and biLSTM. Protvec and the trained model achieved similar results with few differences. Whereas in the LSTM, LSTM with attention and biLSTM, the Protvec achieved better results; in GRU and biLSTM architecture, the trained model outperformed the Protvec. This strengthens the hypothesis that using different corpus would gather different information. In this case, using a model trained with all the Uniprot-Swiss Prot database was not a major advantage.

Furthermore, both Protvec and the trained model were fed to a DNN model composed of 3 dense units with 1024 units and 0.1 DR and batch size 1024 (the remaining parameters were the same as above). As expected, this architecture did not return better results. DNN are not as good as RNN in handling sequential data. Using the Protvec matrix, an Accuracy of 0.76 and MCC of 0.69 was obtained. For the trained model, accuracy of 0.77 and 0.7 MCC was obtained.

*Protein representation - Vector method 2*

In this method, k-mer word frequencies were calculated and multiplied by the corresponding WE vectors. A sequence, independent of the size, will be represented by a matrix of the size of vocabulary times the dimension vector. This method does not take into account spatial information or sequence of aminoacids, only the counts of the words found. Because of this, usually in the literature [65] it is fed to shallow networks, DNNs or CNNs.

Another aspect to take into consideration when using this method is the sparsity of the result returned as the majority of the words will not have counts. For example, considering a

3-gram, a sequence with 500 AAs will have 498 different words maximum (considering all the words are different) within a universe of 9000 words. This would produce a matrix of 9000 size of the vector, where 8500 lines were 0. To avoid this problem, the few papers that use this method tend to use lower values of $n$ or lower sizes of the vector.

To test this method, a Word2vec skipgram 3-gram with size 1 was trained for 10 epochs. The resulting matrix was fed to both shallow ML models and to a DNN network.

For the shallow ML models, a grid search 3-fold cross-validation was performed with the parameters defined by default in the ProPythia package. The shallow algorithms were defined using the scikit learn library.

For the DNN, the number of layers and units of the layer, batch size, and DR were also tested in search for the best performing model. The DNN model was trained using the parameters described above.

The best architectures resulting of the grid search were as follows:

- SVM Parameters: C: 0.1, kernel: rbf
- KNN Parameters: leaf size: 15, n neighbors: 2, weights:'distance'
- RF Parameters: max features: 'sqrt', n estimators: 500
- DNN Parameters: 3 layers DNN 1024, DR: 0.4, batch size:1024

Results of models on the test set can be viewed in Table 9.

Table 9 – Comparison results of SVM, KNN, RF and DNN models using protein representation method 2

| | Acc | MCC | F1 | roc_auc ovr | roc_auc ovo | Precision | Recall |
|---|---|---|---|---|---|---|---|
| SVM | 0.471 | 0.278 | 0.397 | | | 0.539 | 0.471 |
| kNN | 0.573 | 0.483 | 0.590 | 0.771 | 0.775 | 0.660 | 0.573 |
| RF | 0.674 | 0.605 | 0.665 | 0.956 | 0.963 | 0.817 | 0.674 |
| **DNN** | **0.874** | **0.838** | **0.874** | **0.979** | **0.983** | **0.875** | **0.874** |

The model achieving better performance was the DNN with an accuracy of 0.874 and MCC of 0.838. From the traditional ML models, RF was the one performing better achieving an accuracy of 0.674 and an MCC of 0.605. Direct comparisons with the protein representation method 1 cannot be made. However, as expected this method yields worst results than method 1.

*Protein representation - Vector method 3*

The third method has as base the method 2. The n-grams are counted and the vectors are multiplied by the frequency of the word. However, instead of returning the sparse matrix, it sums

the values of all the vectors generating a vector with the dimension of the word vector. This method is the one described in the Protvec original paper [9].

As in the 6.2.2, Protvec and the trained WE model were tested. This time, the sequences were represented by a vector of dimension 100. Although this method is independent from the sequence length (frequency counts and not substitutions), the enzyme sequences were kept with 500 aminoacids for a fair comparison. As described in [80], this length is capable of capturing the majority of enzyme information.

Similarly to the method 2, this method does not take into account the order of the words in the protein. Therefore, RNNs are not the most suitable networks. In this particular method, the final vectors produced have small dimensions, the final vector of a protein is the same size as the vector of a word, which makes it suitable to test shallow Machine Learning algorithms. Therefore, SVM, RF, KNN, GNB, SGD, LR and DNN models were tested.

For the shallow ML models, a grid search 3-fold cross-validation was performed with the parameters described in table 6. The models GNB, SGD, LR yield very poor results and were therefore exclude of the analysis for simplicity reasons. For the DNN, the number of layers and units of the layer, batch size, and DR were also tested in search for the best performing model. The DNN model was trained using the parameters described above.

The best architectures resulting of the grid search were as follows:

- SVM Parameters: C: 10, kernel: rbf, gamma: 0.0001
- KNN Parameters: leaf size: 15, n neighbors: 2, weights:'distance'
- RF Parameters: max features: 'sqrt', n estimators: 500
- DNN Parameters: 3 layers DNN 1048, DR: 0.3, batch size:1024

Results of models on the test set can be viewed in Table 10.

Table 10 – Comparison results of SVM, KNN, RF and DNN models using protein representation method 3

| | Acc | MCC | F1 | roc_auc ovr | roc_auc ovo | precision | recall |
|---|---|---|---|---|---|---|---|
| SVM ProtVec | 0.838 | 0.792 | 0.839 | 0.845 | 0.849 | 0.842 | 0.839 |
| SVM | 0.850 | 0.807 | 0.851 | 0.851 | 0.861 | 0.854 | 0.851 |
| kNN ProtVec | 0.857 | 0.817 | 0.857 | 0.937 | 0.941 | 0.857 | 0.857 |
| kNN | 0.872 | 0.837 | 0.872 | 0.946 | 0.948 | 0.872 | 0.872 |
| RF ProtVec | 0.654 | 0.551 | 0.635 | 0.908 | 0.919 | 0.731 | 0.654 |
| RF | 0.657 | 0.556 | 0.641 | 0.911 | 0.924 | 0.911 | 0.657 |
| DNN Protvec | 0.813 | 0.758 | 0.813 | 0.961 | 0.968 | 0.821 | 0.813 |
| DNN | 0.745 | 0.669 | 0.743 | 0.935 | 0.947 | 0.768 | 0.745 |

This method, however simple, produced good results. The best scores were achieved by the SVM and the KNN. DNNs produced close results and RF was the worst performing model. Overall, the in-house trained WE model achieve slightly better results than Protvec. The scores of this method are overall higher than the ones resulting from the method 2, however lower than the best method2 model that achieved 0.874. They are still lower than the ones produced by the method 1. This method has an advantage of being simple, faster and not requiring so many computational resources. The fact that the scores are close to the ones obtained from the method 1 may indicate that, in some situations, this method should be considered.

### 6.2.3 Comparison of different WE models parameters

With the best model performing so far: 1 BiLSTM layer with 256 units and 0.3 DR followed by an attention mechanism and 2 dense layers of 128 and 64 with 0.1 DR, we tested altering the parameters of the WE in a tentative to improve the results obtained.

The starting WE was also the best performing one, a W2V SG for 3-gram with 100 vectors trained for 10 epochs, window size of 5 and min count of 1.

The protein representation method used was the substitution (method 1). This combination of parameters, as described above, yield an accuracy of 0.931 and an MCC of 0.912 (first row of table 11).

The detailed results of all the comparisons made can be seen below in the table 11.

Table 11 – Comparison of different WE models parameters

| n gram | vector dim | WE Algor | WE Model | Acc | MCC | F1 | roc auc ovr | precision | recall |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 100 | SG | W2V | 0.931 | 0.912 | 0.931 | 0.994 | 0.933 | 0.933 |
| 3 | 100 | SG | FastT | 0.920 | 0.898 | 0.920 | 0.992 | 0.924 | 0.920 |
| 3 | 100 | CBow | W2V | 0.897 | 0.869 | 0.898 | 0.988 | 0.901 | 0.897 |
| 3 | 50 | SG | W2V | 0.929 | 0.909 | 0.929 | 0.993 | 0.931 | 0.929 |
| 3 | 20 | SG | W2V | 0.928 | 0.907 | 0.928 | 0.993 | 0.931 | 0.928 |
| 2 | 100 | SG | W2V | 0.844 | 0.810 | 0.849 | 0.986 | 0.883 | 0.844 |
| 1 | 100 | SG | W2V | 0.865 | 0.828 | 0.865 | 0.982 | 0.877 | 0.865 |

To evaluate different WE configurations, we started to test if the same configuration but training a FastText model instead of a W2V would improve the classification. This model achieved an accuracy of 0.92 and MCC of 0.898 (second row in table 11), slightly lower than the baseline established. As the Fastext model did not improve the classification, we kept the baseline with W2V.

Next, we assessed the impact of altering the algorithm from a SG to a CBow (third row of table 11). This model achieved 0.897 accuracy and 0.869 MCC, lower than the SG baseline and even lower than the SG using FastText. Therefore, we kept the baseline with SG and W2V.

To check the consequences of lowering the dimension of the WE vector, we tested reducing the vector dimension from 100 to 50 (fourth row of table 11) and to 20 (fifth row of table 11). The model with words represented by a vector of 50 achieved a score of 0.929 and an MCC of 0.909, slightly lower than the threshold, but the second best model achieved so far. The vector dimension to 20 also yielded slightly lower results than both the baseline and the vector with dimension 50 achieving 0.928 accuracy and 0.907 MCC. Both of these models achieved scores lower than the baseline and, therefore, the baseline was kept. However, one should pay attention to the fact that the results were very close to the best performing model, while producing a vector significantly lower and that consumes less computational resources like memory and takes less time to run. This also indicates that vectors of size 50 and 20 are capable to capture meaningful information sufficient to classify enzymes.

Lastly, the influence of lowering the n-gram value was estimated. Words of size 1 and 2 were tested. Words with size 4 were not tested due to limitations in computational resources. Results are displayed in sixth and seventh rows of table 11. 2-grams return results of 0.844 accuracy and 0.810 MCC being the worst performing model. Words constituted by 1 AA achieved 0.865 acc and 0.828 MCC. The one gram vector performed better than the 2-gram. However, both models were the less performative of this test. This brings us to the conclusion that diminishing the $n$ gram size does not appear to be good for the enzyme classification problem. This models may, however, benefit from different networks not tested here.

### 6.2.4 Interpretability and visualization

Finally, the visualization techniques implemented and described in the Module development section were applied. This allows to understand better how the WE algorithms represent and distribute the protein words.

As mentioned above, the best WE performing model was the one trained with the enzyme dataset using a W2V model with a SG algorithm for 3-gram words represented by 100 dimensional vectors trained for 10 epochs, window size of 5 and min count of 1.

Following the approach made by [9] in the Protvec paper, a TSNE with the embedding was run. The scikit-learn TSNE was used with the parameters of perplexity 500, number of iterations 1000, learning rate 1000, initialization 'pca' and 2 components.

The TSNE produced was labeled according to the average charge, volume, mass, Van der Waals Volume, polarity and hydrophobicity of individual AAs constituting each trigram. The results can be seen in Figure 14. Clusters are easily identified and correlated with biological features indicating that the WE are capturing meaningful biological information.

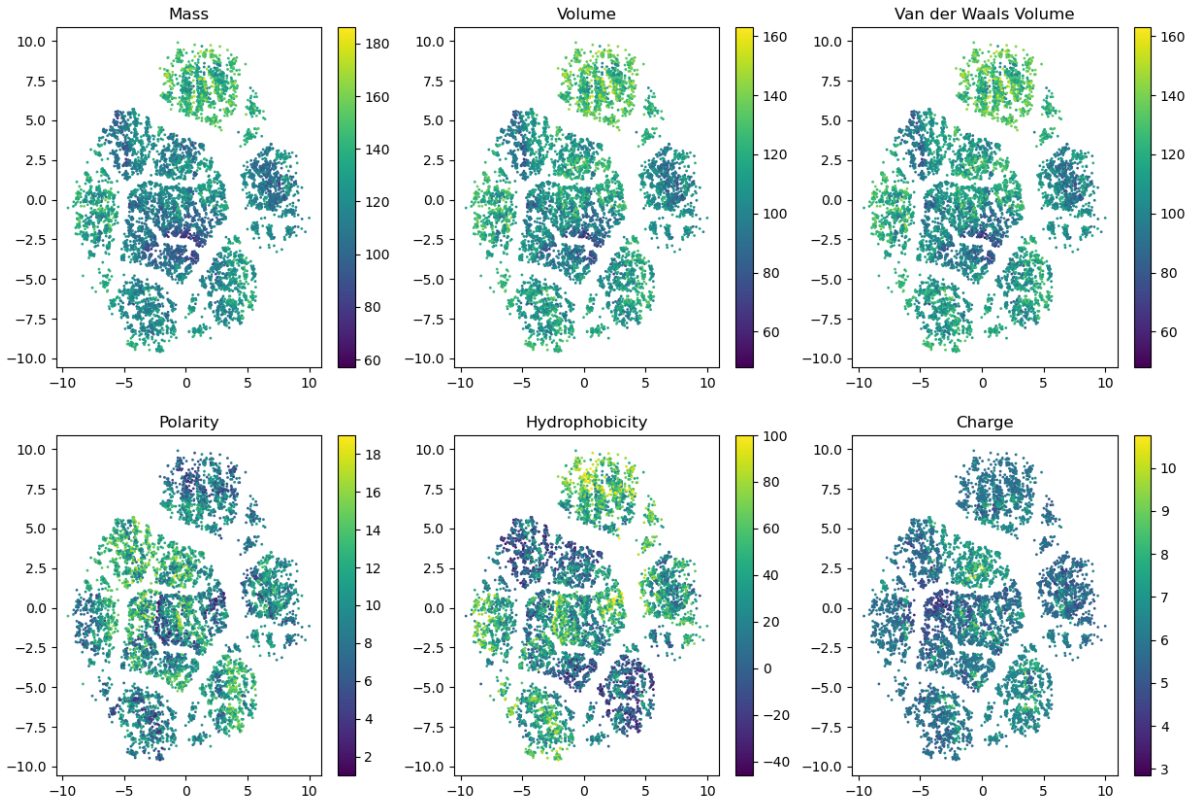



Figure 14 – TSNE mapping of the WE referent to the average charge, volume, mass, Van der Waals Volume, polarity and hydrophobicity of trigrams.

As described in the development section, a simple average of physicochemical characteristics may not be enough to distinguish different n-grams as it gives the same property values to trigrams that may behave differently. For this reason, and explained above, an additional characteristic, the binding free energy for each trigram was used to label the clusters produced by the TSNE.

To also visualize how the different WE modules behave, we performed TSNE on the WE tested on 6.2.3, namely: W2V SG 3gram and 100 dimensions (the described above), W2V with CBOW with 3gram and 100 dimensions, FastText with SG with 3gram and 100 dimensions, W2V SG 3gram and 50 dimensions,W2V SG 3gram and 20 dimensions. Additionally, we represented the WE obtained with W2V SG 3gram and 100 dimensions, but using both positive and negative sequences. The graphics were labeled accordingly to the binding free energy of each trigram. The results can be seen in Figure 15.

Considering the baseline WE of this thesis, it is possible to observe a separation of clusters related with binding free energy, an important biological factor that may be important in enzyme function.

Furthermore, it is possible to observe that the WE trained only on positive sequences yield more explicit clusters than the one trained with both positive and negative sequences. The clusters produced with the CBow algorithm are the ones less visible. Fastext and the model with W2V with 50 dimensions produced results more similar with the baseline WE. This is in accordance with the results obtained in section 6.2.3.

### 6.2.5 General discussion

Overall, word embeddings achieved good results and are suitable to apply in enzyme classification problems. The enzyme classification is a complex problem, which makes one envisioning that it will be also a good approach for other protein classification challenges.

Our best model achieved an accuracy and F1 score of 0.931, MCC of 0.912 and precision of 0.933. Compared to other enzyme prediction tools (that do not use homology but only the sequence itself) this model shows to be competitive. For example, Amidi et al propose EnzyNet a 3D CNN classifier that predicts the EC number based only on their voxel-based spatial structure achieving 0.784 of accuracy [82], DeepEC also using CNN but with one hot encoding shows results of 0.920 precision [83], DEEPre [84] achieved an accuracy of 0.85, using RNN and hot encoding, Sequeira et al. showed results of 0.88 MCC [80] and ProPythia [12] had results of 0.868 MCC using physicochemical features. More recent analysis, such as UDSMPROT, also based on language models and RNNs showed performances of 0.97 accuraccy [85]. A fair comparison cannot be made as the datasets, protein representation and models used are very

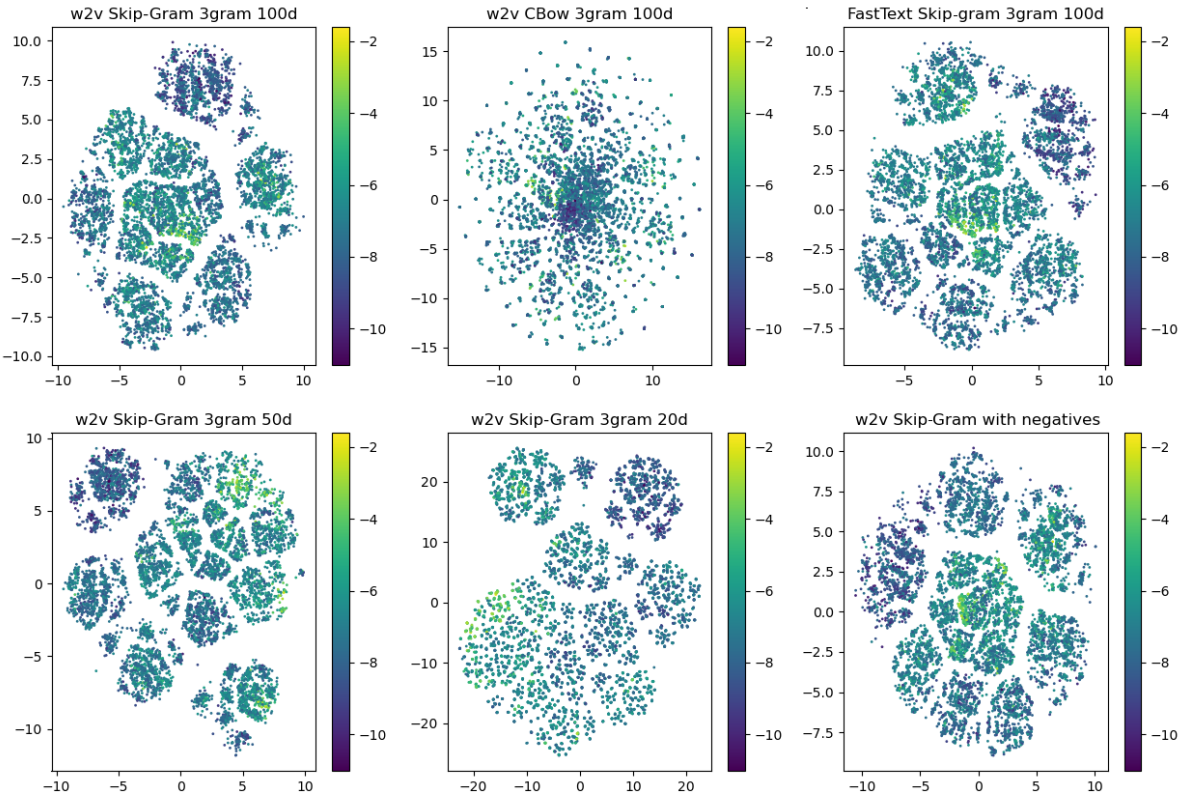



Figure 15 – TSNE mapping of different WE labeled accordingly to the free binding energy of trigrams

different. However, these results show that training and using WE architectures coupled with DL models yield competitive models for challenging protein classification tasks.

More than trying to achieve the best enzyme classification model, the goal was to asses the efficacy of training WE models for protein classification and check different approaches of ML and DL. Furthermore, the developed module enables easy and quick testing of several conformations of WE.

An important aspect when using WE is the choice of training a WE with a specific dataset or use a pre-trained WE. Usually the pre-trained WE are trained using large datasets and are suitable for a broad range of problems. In this work, the Protvec, trained on the curated protein database Swiss-Prot was used. Other WE are available, such as the TransformerCPI [81]. As shown in the results, the use of different WE significantly impact the results, therefore, one should choose carefully the most suitable WE.

The other option is to train a in-house WE model. In this thesis, however with results very close to the ones obtained with Protvec, this choice was the one yielding the best results. Training a WE model also gives more freedom of choice of crucial parameters, such as n-grams and the size of vector. These parameters are not only crucial for the results obtained, but also for the strategy used for the classification models.

Furthermore, when training a WE model, the results obtained showed that the sequences used to train the WE model impact the results, as the WE models capture different relationships between the words. To train a WE model, this is a crucial step that one should take into consideration.

The parameters used to train these models, specially the number of epochs (smaller datasets should be trained for more than 5 epochs) also impact the quality of the relationships between words captured by the model. Vectors with larger dimensions, such as 100, capture more patterns, but are more computationally expensive. On the other hand, vectors with lower dimensions are fast to train, but can loose some of the complexity and relationships between vectors. The results showed that the best vector was the one with 100 dimension, while the results achieved with vectors of 50 are also competitive and require less computational resources.

Three different techniques of representing a protein sequence with the WE vectors were tested. From the results described above, substituting the $n$-gram of the sequence for the WE vector is the best approach. This method is the only one that preserves the position of the words and, as it does not perform any change to the vectors, maintains all the relationships learnt by the model. This method, depending on the sequence length can result in very large vectors and become a computational burden. Moreover, this method, makes more sense using RNNs that are very good capturing patterns and sequential data. These networks are also computational heavy.

Counting based methods (method 2 and 3) are good to reduce complexity and allow to use different networks and even shallow ML. As the method 2 generates very sparse matrices, usually it is used with very low vector dimensions or $n$ grams and is the worst performing method. Method 3, that returns a final vector for the protein with the same size as the vector of a word, yields also competitive results while being very fast to train. Both these methods loose the spatial information while classifying, but can be more suitable for other simpler protein problems or when the computational resources are a limiting factor.

Finally, the module created allowed to plot the produced n-gram vectors in a 2-Dimensional space and correlate them with physicochemical features. One of the advantages of this type of encoding is that we can derive interpretability methods and understand what type of meaningful biological relationships the model is capturing.

# 7

# CONCLUSIONS AND FUTURE PROSPECTS

The first aim of this project was to develop a framework that would ease the implementation of word embedding models toward protein representation and classification. The module developed facilitates the tokenization of sequences, training of WE models using different algorithms, downloading pre-trained state-of-art WE models, vectorization of protein sequences and visualization and interpretability of WE. Accordingly, the module is able to process biological sequences aiming to search for semantic meaning in sequence "words". Although it can be used for both protein and DNA sequences, it was only tested in proteins. The module made it easy, quick and in an intuitive way, to test several conformations of WE.

Furthermore, this module was integrated in the ProPythia framework, allowing to straightforwardly integrate WE representations with the training and testing of ML and DL models.

This module was validated using two studies namely, identification of plant ubiquitylation sites and lysine crotonylation site prediction. The comparative analysis made evident the performance and the usability and validated the WE module presented.

Another major goal was to apply the developed pipeline in combination with ML and DL methods to the enzyme classification problem. Several WE were tested and fed to different ML and DL networks. Overall, WE achieved good results being even competitive with state-of-the-art models. This way, we can conclude that WE can be applied with success to a wide range of protein classification problems. Moreover, visualization techniques can be applied to the WE vectors and shed a light to the complex interaction between biological words.

To further improve this study, more advanced language models such as Elmo and transformers should be implemented in the module and tested. This would be a major improvement to the developed code and with great usability.

In addition, study different types of ngram segmentation to include the segmentation for domains [68] and combinations of variable sized grams [72] would be very interesting and could improve the scores here obtained.

Finally, to expand the case studies to test WE for DNA sequences and explore more challenging protein classification tasks could be future prospects of this work.

# BIBLIOGRAPHY


[1] Creighton, T. (1993). *Proteins: Structures and molecular properties*. W. H. Freeman.

[2] Kulmanov, M., & Hoehndorf, R. (2019). DeepGOPlus: improved protein function prediction from sequence. *Bioinformatics*, *36*(2), 422–429.

[3] Lemley, J., Bazrafkan, S., & Corcoran, P. (2017). Deep learning for consumer devices and services: Pushing the limits for machine learning, artificial intelligence, and computer vision. *IEEE Consumer Electronics Magazine*, *6*, 48–56.

[4] Larsen, P. O., & von Ins, M. (2010). The rate of growth in scientific publication and the decline in coverage provided by science citation index. *Scientometrics*, *84*, 575–603.

[5] Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). *Natural language processing: An introduction*.

[6] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013a). Efficient estimation of word representations in vector space.

[7] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations.

[8] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *CoRR*, *abs/1706.03762*.

[9] Asgari, E., & Mofrad, M. R. K. (2015). Continuous distributed representation of biological sequences for deep proteomics and genomics. *PLoS ONE*, *10*, 141287.

[10] Elnaggar, A., Heinzinger, M., Dallago, C., Rihawi, G., Wang, Y., Jones, L., Gibbs, T., Feher, T., Angerer, C., Steinegger, M., Bhowmik, D., & Rost, B. (2020). Prottrans: Towards cracking the language of life's code through self-supervised deep learning and high performance computing.

[11] Heinzinger, M., Elnaggar, A., Wang, Y., Dallago, C., Nechaev, D., Matthes, F., & Rost, B. (2019). Modeling aspects of the language of life through transfer-learning protein sequences. *BMC Bioinformatics*, *20*.

[12] Sequeira, A. M., Lousa, D., & Rocha, M. (2022). Propythia: A python package for protein classification based on machine and deep learning. *Neurocomputing*, *484*, 172–182.

[13] Clark, W. T., & Radivojac, P. (2011). Analysis of protein function and its prediction from amino acid sequence, 2086–2096.

[14] Schwaighofer, A., Schroeter, T., Mika, S., & Blanchard, G. (2009). How wrong can we get? a review of machine learning approaches and error bars. *Combinatorial Chemistry & High Throughput Screening*, *12*(5), 453–468.

[15] Chollet, F. (2017). *Deep learning with python*. Manning.

[16] Alpaydin, E. (2014). *Introduction to machine learning* (3rd ed.). MIT Press.

[17] Tarca, A. L., Carey, V. J., Chen, X.-w., Romero, R., & Drăghici, S. (2007). Machine learning and its applications to biology. *PLOS Computational Biology*, *3*(6), 1–11.

[18] Schmidhuber, J. (2015). Deep learning in neural networks : An overview. *Neural Networks*, *61*, 85–117.

[19] Raschka, S., & Mirjalili, V. (2017). *Python Machine Learning, 2nd Ed.* (2nd ed.). Packt Publishing.

[20] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning* (M. Press, Ed.).

[21] Ben-david, S. (2014). *Understanding Machine Learning : From Theory to Algorithms*.

[22] Angermueller, C., Pärnamaa, T., Parts, L., & Stegle, O. (2016). Deep learning for computational biology, 1–16.

[23] Bishop, C. M. (2006). *Pattern recognition and machine learning (information science and statistics)*. Springer-Verlag.

[24] Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis [Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists]. *Chemometrics and Intelligent Laboratory Systems*, *2*(1), 37–52.

[25] Jolliffe, I. (2011). Principal component analysis. In M. Lovric (Ed.), *International encyclopedia of statistical science* (pp. 1094–1096). Springer Berlin Heidelberg.

[26] Dash, M., & Liu, H. (1997). Feature selection for classification. *Intelligent Data Analysis*, *1*(1), 131–156.

[27] Bluma, A. L., & Langley, P. (1997). Artificial Intelligence Selection of relevant features and examples in machine. *97*(97), 245–271.

[28] Min, S., Lee, B., & Yoon, S. (2016). Deep learning in bioinformatics. *Briefings in Bioinformatics*, *18*(5), 851–869.

[29] Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, *4*(1), 1–58.

[30] Kohavi, R. (2013). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. (June).

[31] Bartlett, P. L. (2002). Model Selection and Error Estimation, 85–113.

[32] Yelle, L. E. (1979). The learning curve: Historical review and comprehensive survey. *Decision Sciences*, *10*(2), 302–328.

[33] Dangeti, P. (2017). *Statistics for machine learning: Techniques for exploring supervised, unsupervised, and reinforcement learning models with python and r*. Packt Publishing.

[34] Haykin, S. S. (2009). *Neural networks and learning machines* (Third). Pearson Education.

[35] Bradley, A. P. (1997). The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, *30*(7), 1145–1159.

[36] Goutte, C., & Gaussier, E. (2005). A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In D. E. Losada & J. M. Fernández-Luna (Eds.), *Advances in information retrieval* (pp. 345–359). Springer Berlin Heidelberg.

[37] Schwartz, A. S., Hannum, G. J., Dwiel, Z. R., Smoot, M. E., Grant, A. R., Knight, M., Becker, S. A., Eads, J. R., Lafave, M. C., & Eavani, H. (2018). Deep Semantic Protein Representation for Annotation, Discovery, and Engineering.

[38] Kim, K. G. (2016). Deep Learning. *22*(4), 351–354.

[39] Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning.

[40] Girosi, F. (1995). Regularization Theory and Neural Networks Architectures. *269*(1), 219–269.

[41] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, *abs/1502.03167*.

[42] Mamoshina, P., Vieira, A., Putin, E., & Zhavoronkov, A. (2016). Applications of Deep Learning in Biomedicine.

[43] Deng, L., Way, O. M., Yu, D., & Way, O. M. (2014). Deep Learning : Methods and Applications. *7*(2013), 197–387.

[44] Al-Rfou, R., Alain, G., Almahairi, A., Angermüller, C., Bahdanau, D., Ballas, N., Bastien, F., Bayer, J., Belikov, A., Belopolsky, A., Bengio, Y., Bergeron, A., Bergstra, J., Bisson, V., Snyder, J. B., Bouchard, N., Boulanger-Lewandowski, N., Bouthillier, X., de Brébisson, A., . . . Zhang, Y. (2016). Theano: A python framework for fast computation of mathematical expressions. *CoRR*, *abs/1605.02688*.

[45] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I. J., Harp, A., Irving, G., Isard, M., Jia, Y., Józefowicz, R., Kaiser, L., Kudlur, M., . . . Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, *abs/1603.04467*.

[46] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013b). Efficient estimation of word representations in vector space.

[47] Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). Enriching word vectors with subword information. *CoRR*, *abs/1607.04606*.

[48] Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of tricks for efficient text classification. *CoRR*, *abs/1607.01759*.

[49] Řehůřek, R., & Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 45–50.

[50] O'Connor, J. U., C. M. Adams. (2010). *Essentials of cell biology*. Cambridge, MA: Nature Publishing Group Education.

[51] McDonald, A. G., Boyce, S., & Tipton, K. F. (2008). ExplorEnz: the primary source of the IUBMB enzyme list. *Nucleic Acids Research*, *37*(suppl$_1$), D593–D597.

[52] Seo, S., Oh, M., Park, Y., & Kim, S. (2018). DeepFam: deep learning based alignment-free method for protein family modeling and prediction. *Bioinformatics*, *34*(13), i254–i262.

[53] Consortium, T. U. (2016). UniProt: the universal protein knowledgebase. *Nucleic Acids Research*, *45*(D1), D158–D169.

[54] Bairoch, A., & Apweiler, R. (1996). The SWISS-PROT Protein Sequence Data Bank and Its New Supplement TREMBL. *Nucleic Acids Research*, *24*(1), 21–25.

[55] Bairoch, A., & Apweiler, R. (2000). The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000. *Nucleic Acids Research*, *28*(1), 45–48.

[56] Sigrist, C. J. A., Cerutti, L., de Castro, E., Langendijk-Genevaux, P. S., Bulliard, V., Bairoch, A., & Hulo, N. (2009). PROSITE, a protein domain database for functional characterization and annotation. *Nucleic Acids Research*, *38*(suppl$_1$), D161–D166.

[57] Mistry, J., Chuguransky, S., Williams, L., Qureshi, M., Salazar, G. A., Sonnhammer, E. L. L., Tosatto, S. C. E., Paladin, L., Raj, S., Richardson, L. J., Finn, R. D., & Bateman, A. (2020). Pfam: The protein families database in 2021. *Nucleic Acids Research*, *49*(D1), D412–D419.

[58] Cantelli, G., Bateman, A., Brooksbank, C., Petrov, A. I., Malik-Sheriff, R. S., Ide-Smith, M., Hermjakob, H., Flicek, P., Apweiler, R., Birney, E., & McEntyre, J. (2021). The European Bioinformatics Institute (EMBL-EBI) in 2021. *Nucleic Acids Research*, *50*(D1), D11–D19.

[59] Krogh, A., Brown, M., Mian, I., Sjölander, K., & Haussler, D. (1994). Hidden markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, *235*(5), 1501–1531.

[60] Liu, X. (2017). Deep recurrent neural network for protein function prediction from sequence.

[61] Schwartz, A. S., Hannum, G. J., Dwiel, Z. R., Smoot, M. E., Grant, A. R., Knight, J. M., Becker, S. A., Eads, J. R., LaFave, M. C., Eavani, H., Liu, Y., Bansal, A. K., & Richardson, T. H. (2018). Deep semantic protein representation for annotation, discovery, and engineering. *bioRxiv*.

[62] Politano, G., & Benso, A. (2018). Beyond Homology Transfer : Deep Learning for Automated.

[63] Kulmanov, M., Khan, M. A., & Hoehndorf, R. (2018). Deepgo: Predicting protein functions from sequence and interactions using a deep ontology-aware classifier. *Bioinformatics*, *34*, 660–668.

[64] Fa, R., Cozzetto, D., Wan, C., & Jones, D. T. (2018). Predicting human protein function with multi- task deep neural networks, 1–16.

[65] Nguyen, T. T. D., Le, N. Q. K., Ho, Q. T., Phan, D. V., & Ou, Y. Y. (2019). Using word embedding technique to efficiently represent protein sequences for identifying substrate specificities of transporters. *Analytical Biochemistry*, *577*, 73–81.

[66] Yang, K. K., Wu, Z., Bedbrook, C. N., & Arnold, F. H. (2018). Learned protein embeddings for machine learning. *Bioinformatics*, *34*, 2642–2648.

[67] Hamid, M. N., & Friedberg, I. (2019). Identifying antimicrobial peptides using word embedding with deep recurrent neural networks. *Bioinformatics*, *35*, 2009–2016.

[68] Sarker, B., Ritchie, D. W., & Aridhi, S. (2019). Functional Annotation of Proteins using Domain Embedding based Sequence Classification. *KDIR 2019 - 11th International Conference on Knowledge Discovery and Information Retrieval*, 163–170.

[69] Le, N. Q. K., & Huynh, T.-T. (2019). Identifying snares by incorporating deep learning architecture and amino acid embedding representation. *Frontiers in Physiology*, *10*.

[70] Guo, Y., Yang, Y., Huang, Y., & Shen, H. B. (2020). Discovering nuclear targeting signal sequence through protein language learning and multivariate analysis. *Analytical Biochemistry*, *591*.

[71] Wang, H., Wang, Z., Li, Z., & Lee, T.-Y. (2020). Incorporating deep learning with word embedding to identify plant ubiquitylation sites. *Frontiers in Cell and Developmental Biology*, *8*.

[72] Chen, J., Yang, R., Zhang, C., Zhang, L., & Zhang, Q. (2019). Deepgly: A deep learning framework with recurrent and convolutional neural networks to identify protein glycation sites from imbalanced data. *IEEE Access*, *7*, 142368–142378.

[73] Nguyen, T. T. D., Le, N. Q. K., Ho, Q. T., Phan, D. V., & Ou, Y. Y. (2020). Tnfpred: Identifying tumor necrosis factors using hybrid features based on word embeddings. *BMC Medical Genomics*, *13*.

[74] Yi, H. C., You, Z. H., Cheng, L., Zhou, X., Jiang, T. H., Li, X., & Wang, Y. B. (2020). Learning distributed representations of rna and protein sequences and its application for predicting lncrna-protein interactions. *Computational and Structural Biotechnology Journal*, *18*, 20–26.

[75] Wei, X., Sha, Y., Zhao, Y., He, N., & Li, L. (2021). Deepkcrot: A deep-learning architecture for general and species-specific lysine crotonylation site prediction. *IEEE Access*, *9*, 49504–49513.

[76] Ibtehaz, N., & Kihara, D. (2021). Application of sequence embedding in protein sequence-based predictions.

[77] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780.

[78] Chen, R., Miao, Y., Hao, X., Gao, B., Ma, M., Zhang, J. Z., Wang, R., Li, S., He, X., & Zhang, L. (2021). Investigation on the characteristics and mechanisms of ace inhibitory peptides by a thorough analysis of all 8000 tripeptides via binding free energy calculation. *Food Science & Nutrition*, *9*(6), 2943–2953.

[79] Zhao, Y., He, N., Chen, Z., & Li, L. (2020). Identification of protein lysine crotonylation sites by a deep learning framework with convolutional neural networks. *IEEE Access*, *8*, 14244–14252.

[80] Sequeira, A. M., & Rocha, M. (2022). Recurrent deep neural networks for*enzyme functional annotation. In M. Rocha, F. Fdez-Riverola, M. S. Mohamad, & R. Casado-Vara (Eds.), *Practical applications of computational biology & bioinformatics, 15th international conference (pacbb 2021)* (pp. 62–73). Springer International Publishing.

[81] Chen, L., Tan, X., Wang, D., Zhong, F., Liu, X., Yang, T., Luo, X., Chen, K., Jiang, H., & Zheng, M. (2020). TransformerCPI: improving compound–protein interaction predic-

tion by sequence-based deep learning with self-attention mechanism and label reversal experiments. *Bioinformatics*, *36*(16), 4406–4414.

[82] Amidi, A., Amidi, S., Vlachakis, D., Megalooikonomou, V., Paragios, N., & Zacharaki, E. I. (2017). Enzynet: Enzyme classification using 3d convolutional neural networks on spatial representation.

[83] Ryu, J. Y., Kim, H. U., & Lee, S. Y. (2019). Deep learning enables high-quality and high-throughput prediction of enzyme commission numbers. *Proceedings of the National Academy of Sciences*, *116*(28), 13996–14001.

[84] Li, Y., Wang, S., Umarov, R., Xie, B., Fan, M., Li, L., & Gao, X. (2017). DEEPre: sequence-based enzyme EC number prediction by deep learning. *Bioinformatics*, *34*(5), 760–769.

[85] Strodthoff, N., Wagner, P., Wenzel, M., & Samek, W. (2020). Udsmprot: Universal deep sequence models for protein classification. *Bioinformatics*, *36*, 2401–2409.