

**Universidade do Minho**

Escola de Engenharia

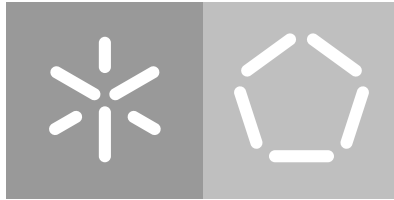
Departamento de Informática

Nuno Miguel Caetano Alves

**Development of a Tool based  
on Deep Learning able to classify  
Biomedical Literature**

October 2020





**Universidade do Minho**

Escola de Engenharia

Departamento de Informática

Nuno Miguel Caetano Alves

**Development of a Tool based  
on Deep Learning able to classify  
Biomedical Literature**

Master dissertation

Master Degree in Bioinformatics

Dissertation supervised by

**Miguel Francisco Almeida Pereira da Rocha**

October 2020

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

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada. Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

### **Licença concedida aos utilizadores deste trabalho**



**Atribuição  
CC BY**

<https://creativecommons.org/licenses/by/4.0/>

---

## ACKNOWLEDGEMENTS

---

Firstly, I would like to thank Dr. Miguel Rocha and Ruben Rodrigues for the great opportunity of working with them. All the support, availability and expertise were highly motivating and the main force of this entire work. This thesis would not have been possible without their patient guidance and useful corrections. Also, thank you for integrating me in the BISBII group, where all members were really welcoming and cooperative.

Secondly, I would like to thank the opportunity given me to work as a grantee on the project DeepBio- Machine learning and deep learning approaches to industrial biotechnology applications - ref. NORTE-01-0247-FEDER-039831, funded by Lisboa 2020, Norte 2020, Portugal 2020 and FEDER - Fundo Europeu de Desenvolvimento Regional. This project allowed me to extend my knowledge on Deep Learning and increase my devotion for the field.

In addition, I must express my sincere gratitude to my family for giving continuous support and encouragement throughout all my years of study. A special thanks also to my girlfriend Cátia Mendes for all the emotional support and friendship not only on this challenging year but throughout these 6 years together.

Finally, I wish to thank all my friends for caring about me and for cheering me up throughout specially on this difficult pandemic year that we are all going through, with a special thanks to Ana Andrade for all the support.

## **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

---

In the last decades, the scientific community has produced huge amounts of publications about the most varied biomedical topics, making the search for relevant information a really difficult task for every researcher. Some approaches have been followed to develop tools that can facilitate this process. For instance, PubMed implemented in 2017 a Machine Learning model to sort documents by their relevance. Nevertheless, even the authors consider that their system would benefit from the implementation of a Deep Learning model, which for now needs more studies.

In this context, a package called BioTMPy<sup>1</sup> was developed in this work, to perform document classification of biomedical literature using the Python programming language. The package is divided into different modules to provide to the user functions to read documents in different formats, perform preprocessing and data analysis and to train, optimize and evaluate Machine and Deep learning models. Our package also provides intuitive pipelines that can be easily adapted for the user needs, illustrating how to implement complex deep learning models.

The developed package was applied to a dataset from a challenge of the BioCreative forum, from 2019, about protein-protein interactions altered by mutations, an important topic for the advances related to precision medicine. Using this dataset, it was possible to observe a slightly better performance of BioWordVec pre-trained embeddings over GloVe, "pubmed\_pmc" and "pubmed\_ncbi" embeddings. Also, with the evaluation of the developed models on the test set, we managed to overcome the challenge's best submission, by using a model with BioBERT and a bidirectional LSTM on top, resulting in a difference of 7.25% for average precision, 3.22% for precision, 2.99% for recall and 3.15% for the f1-score.

Also, a web server was developed to provide access to the best Deep Learning model trained in this work. The overall pipeline here developed can be applied to other case studies in different topics, provided there is a set of documents annotated as relevant and non-relevant, allowing to train the models.

**Keywords:** Deep Learning; Machine Learning; Document Classification; Text Mining.

---

<sup>1</sup> <https://gitlab.bio.di.uminho.pt/biotextminingpy/biotmpy>

---

## RESUMO

---

Nas últimas décadas, a comunidade científica tem produzido uma enorme quantidade de publicações sobre os mais variados tópicos biomédicos, tornando a procura de informação relevante num processo complicado para qualquer investigador. Algumas abordagens têm sido seguidas para desenvolver ferramentas que possam facilitar este processo. Por exemplo, o *PubMed* implementou em 2017 um modelo de aprendizagem máquina para ordenar documentos pela sua relevância. Contudo, os autores consideram que o seu sistema pode beneficiar com a implementação de um modelo de *Deep Learning*, o que para já necessita de mais estudos.

Neste projeto, foi desenvolvida um *package* chamado BioTMPy para classificar documentos da literatura biomédica através da linguagem de programação *Python*. Este *package* é dividido em diferentes módulos para fornecer ao utilizador funções para ler documentos de formatos diferentes, realizar pré-processamento e análise de dados, e para treinar, otimizar e avaliar modelos de aprendizagem máquina. A plataforma também fornece *pipelines* intuitivas que podem ser facilmente adaptadas de acordo com as necessidades do utilizador, demonstrando como implementar modelos complexos de *Deep Learning*.

O *package* desenvolvido foi aplicado a um conjunto de dados de um desafio do fórum *BioCreative*, de 2019, acerca de interações proteína-proteína alteradas por mutações, um tópico importante para a área da medicina de precisão. Usando este conjunto de dados, consegue-se observar um melhor desempenho dos BioWordVec *embeddings* pré-treinados em relação a *embeddings* como GloVe, "pubmed\_pmc" e "pubmed\_ncbi". Com os modelos desenvolvidos, foi possível ultrapassar a melhor submissão do *challenge*, usando um modelo com BioBERT e uma LSTM bidirecional acima, obtendo-se diferenças de 7.25% na precisão média, 3.22% na precisão, 2.99% no recall e 3.15% para o *f1-score*.

Foi ainda desenvolvido um servidor *web* de forma a fornecer acesso ao nosso melhor modelo. A plataforma desenvolvida neste trabalho poderá ser aplicável a outros casos de estudo em diferentes tópicos, desde que exista um conjunto de documentos anotado como relevante ou não relevante, que permita treinar os modelos.

**Palavras-Chave:** *Deep Learning*; Aprendizagem Máquina, Classificação de Documentos; Mineração de Texto



---

## CONTENTS

---

1	INTRODUCTION	1
1.1	Context and Motivation	1
1.2	Objectives	2
1.3	Thesis Organization	3
2	STATE OF THE ART	4
2.1	Biomedical Text Mining	4
2.1.1	Document Classification	6
2.1.2	PubMed	7
2.1.3	Corpora and Imbalanced Data Sets	8
2.1.4	Preprocessing	9
2.1.5	Embeddings	10
2.1.6	Named Entity Recognition	13
2.2	Machine Learning	14
2.2.1	Unsupervised Learning	15
2.2.2	Supervised Learning	17
2.2.3	Evaluation of Machine Learning Models	17
2.3	Deep Learning	19
2.3.1	Neural Networks	20
2.3.2	Training Process	23
2.3.3	Regularization	26
2.3.4	Convolutional Neural Networks	28
2.3.5	Recurrent Neural Networks	30
2.3.6	Long Short-Term Memory	32
2.3.7	Gated Recurrent Units	34
2.3.8	Attention Mechanism	35
2.3.9	BERT	36
2.4	Transfer Learning	38
2.5	Hyperparameter Optimization	39
2.6	Related Work	40
3	DEVELOPMENT	43
3.1	Python Packages	43
3.2	BioTMPy - Package Description	44
3.3	Wrappers and Data Structures	45
3.4	Preprocessing	47

3.5	Machine Learning	50
3.6	Pipelines	51
3.7	Web	53
3.8	Discussion	55
4	RESULTS AND DISCUSSION	58
4.1	Dataset Description and Challenge Overview	58
4.2	Preprocessing	60
4.3	Data Analysis	61
4.4	Deep Learning Results	64
4.4.1	Computational Resources	64
4.4.2	Embeddings	64
4.4.3	Hyperparameter Optimization	68
4.4.4	Models	71
4.4.5	Evaluation	76
5	CONCLUSION	81
A	SUPPLEMENTARY MATERIAL	95

---

## ACRONYMS

---

<b>AdaGrad</b>	Adaptive Gradient Algorithm
<b>Adam</b>	Adaptive Momentum Estimation
<b>AI</b>	Artificial Intelligence
<b>ANN</b>	Artificial Neural Network
<b>AUC</b>	Area Under the Curve
<b>BERT</b>	Bidirectional Encoder Representations from Transformers
<b>Bi-GRU</b>	Bidirectional Gated Recurrent Unit
<b>Bi-LSTM</b>	Bidirectional Long Short-Term Memory
<b>Bi-RNN</b>	Bidirectional Recurrent Neural Network
<b>BioTM</b>	Biomedical Text Mining
<b>CBOW</b>	Continuous Bag-of-words
<b>CNN</b>	Convolutional Neural Network
<b>DL</b>	Deep Learning
<b>DNN</b>	Deep Neural Network
<b>FP</b>	False Positives
<b>FPR</b>	False Positive Rate
<b>FN</b>	False Negatives
<b>GRU</b>	Gated Recurrent Unit
<b>GPU</b>	Graphics Processing Unit
<b>LDA</b>	Linear Discriminant Analysis
<b>LSTM</b>	Long Short-Term Memory
<b>MCC</b>	Matthews correlation coefficient

<b>MeSH</b>	Medical Subject Headings
<b>ML</b>	Machine Learning
<b>MLM</b>	Masked Language Model
<b>MLP</b>	Multilayer Perceptron
<b>MSE</b>	Mean Squared Error
<b>NER</b>	Named Entity Recognition
<b>NLP</b>	Natural Language Processing
<b>NLTK</b>	Natural Language Processing Toolkit
<b>PPI</b>	Protein-protein Interactions
<b>PCA</b>	Principal Component Analysis
<b>PR</b>	Precision-Recall
<b>ReLU</b>	Rectified Linear Unit
<b>ROC</b>	Receiver Operating Characteristic
<b>RMSprop</b>	Root Mean Square Propagation
<b>RNN</b>	Recurrent Neural Network
<b>SGD</b>	Stochastic Gradient Descent
<b>SVM</b>	Support Vector Machine
<b>TF-IDF</b>	Term Frequency - Inverse Document Frequency
<b>TN</b>	True Negatives
<b>TP</b>	True Positives
<b>TPR</b>	True Positive Rate
<b>t-SNE</b>	t-Distributed Stochastic Neighbor Embedding

---

## LIST OF FIGURES

---

Fig. 1	Document Classification Pipeline	7
Fig. 2	One-hot Encoding process	10
Fig. 3	Word2vec models' representation	11
Fig. 4	Deep Learning's Location	19
Fig. 5	Perceptron architecture	21
Fig. 6	Activation Functions	22
Fig. 7	Graph representation of <i>underfitting</i> vs. <i>overfitting</i>	26
Fig. 8	Dropout Representation	27
Fig. 9	CNN operations on a Word Embedding matrix	30
Fig. 10	RNN architecture and its unfolded version	31
Fig. 11	LSTM's memory block representation	32
Fig. 12	Package structure	45
Fig. 13	Wrappers Inputs	46
Fig. 14	Wrappers Pipeline	47
Fig. 15	Preprocessing Pipeline	49
Fig. 16	BioTMPy - Home Page	54
Fig. 17	BioTMPy - Results Page	55
Fig. 18	Top Number of Words	61
Fig. 19	Total number of words per document	62
Fig. 20	t-SNE results for both sets	63
Fig. 21	Embeddings t-SNE 2D plots	66
Fig. 22	Results using different Pre-trained Embeddings	67
Fig. 23	Cross-validation results on Training Set	74
Fig. 24	Number of words of Training set	95
Fig. 25	Number of Words per Sentence	95
Fig. 26	Number of sentences of Training set	96
Fig. 27	BERT Tokenizers Comparison	96
Fig. 28	Top 100 words of Training set	97
Fig. 29	Top 100 words of Test set	98
Fig. 30	Top 20 words of 4 subsets	99

---

## LIST OF TABLES

---

Table 1	Some of the currently available Embeddings	13
Table 2	Deep Learning's Advantages and Limitations [6, 7]	20
Table 3	Selection of last-layer activation and loss functions	22
Table 4	ML approaches for Document Classification	41
Table 5	DL approaches for Document Classification	42
Table 6	Available Machine and Deep Learning Models	50
Table 7	Datasets Content	59
Table 8	Challenge's Submissions	60
Table 9	Search Space for the Hierarchical Attention Network	69
Table 10	Search Space for the BioBERT Dense	70
Table 11	Search Space for the BioBERT LSTM	71
Table 12	Search Space for the BioBERT CLS	72
Table 13	Results obtained on the test set	77

---

## INTRODUCTION

---

### 1.1 CONTEXT AND MOTIVATION

Over the last decades, due to the continuous growth of science and technology, there has been a consecutive increase in the number of per-year published articles [1, 2]. Considering the large quantity of available documents, it is almost an impossible task for a single researcher to read and organize all the information about a certain topic. To ease this time consuming process, a field named Biomedical Text Mining (BioTM) has emerged [3].

The main goal of BioTM is developing Text Mining tools in a biomedical context. BioTM approaches different tasks, from Information Extraction to Information Retrieval and Document Classification. These tasks have the potential to support different processes, for instance, the identification of new drugs and new disease treatments [2, 4].

Document Classification, a task in BioTM, aims to classify documents based on their content, assigning them different predefined labels. For binary classification, a common approach is labelling documents as relevant or non-relevant regarding a chosen topic. Document Classification can be done in an automated way using both rule-based approaches or Machine Learning algorithms, the latter being more adequate for pattern identification. Explicitly, Supervised Learning, a sub-field of Machine Learning, contains a variety of successful algorithms as Support Vector Machine (SVM), Naïve Bayes, Logistic Regression, Neural Networks, etc. [5, 6].

In recent years, Deep Learning, another sub-field of Machine Learning, has gained popularity within the scientific community. Deep Learning presents several advantages over conventional Machine Learning approaches. Besides an overall improvement in performance, it also presents

better data representation and it does not rely on feature engineering, a key step on traditional Machine Learning algorithms [7].

When applied to texts, deep learning models can learn features directly from complex and abstract representations of raw data. For Document Classification, Word Embeddings (e.g. GloVe and FastText) are commonly implemented due to their semantic text representation [7]. Algorithms such as the Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) and RNN variations as the Long Short-Term Memory (LSTM) and the Gated Recurrent Unit (GRU) have been showing improvements in different text related tasks [8–10].

Despite recent breakthroughs, BioTM and Document Classification for biomedical literature would still benefit from further improvements [11, 12]. For instance, PubMed, a widely used search engine for biomedical documents containing more than 28 million publications, has implemented their machine learning system only in 2017. Nevertheless, as far as we know, there is still no Deep Learning tool able to manage such amount of documents and subjects [13]. The current project focuses precisely on these challenges, attempting to overcome them through Deep Learning algorithms.

## 1.2 OBJECTIVES

Keeping the above context in mind, this project aims to create a Deep Learning package, enabling to develop pipelines capable of classifying a set of documents according to their relevance to a topic of interest.

The following scientific and technological objectives will be pursued:

- Literature Review on Deep Learning concepts, methods and tools, as well as its applications on text mining, namely on a biomedical context;
- Development of tools capable of converting unstructured articles into a standard structure for further use in Deep Learning models;
- Development a package in Python, allowing to develop pipelines to train Deep Learning models and predict document relevance on a given topic based on them, supporting the use of distinct models and architectures;



- Validation of the created package in a selected case study from a challenge related to precision medicine;
- Development of a RESTful API allowing to apply the trained models to simplify and facilitate its open use by the community.

### 1.3 THESIS ORGANIZATION

**Chapter 2** will be addressing general subjects as Biomedical Text Mining, document classification and Machine Learning. For Machine learning, different models for both supervised and unsupervised learning will be also explained, with a big focus on Deep Learning models. Additionally, Machine Learning concepts as vectorization (highlighting available pre-trained embeddings), regularization, transfer learning and hyperparameter optimization will also be described.

**Chapter 3** will explain the main fundamentals of the package developed in this work. Additionally, it will be described some of the main features of the package as well as the approaches used to implement them. At the end, the package will be discussed, together with some possible improvements for the future.

**Chapter 4** will exhibit all the results obtained in the different steps of this work, since a data analysis performed on a BioCreative's dataset, to a comparison between several pre-trained embeddings and scores of the application of some Deep Learning models on that same dataset. Throughout this chapter, some discussion will be presented along with possible improvements.

**Chapter 5** will describe some of the final conclusions of the entire project, together with some future work.

---

## STATE OF THE ART

---

### 2.1 BIOMEDICAL TEXT MINING

Currently, the amount of published scientific articles is increasing at an incredibly fast rate, leaving the scientific community with a huge amount of available documents. This makes the search for relevant information a really hard work for researchers, requiring from them an excessive amount of reading time. Consequently, researchers have been using computers' abilities, such as the high capacity of data processing and storage to speed up the process [1, 14]. However, the content interpretation of articles is not a straightforward task for a computer, since these documents are normally written as unstructured free text [5]. BioTM emerged to create and provide tools capable of identifying and extracting significant information from structured written biomedical texts [3]. BioTM is a branch of a more generic field named Text Mining, which aims to do the same task for general-purpose texts.

Text Mining can be defined as a group of technologies and tools with the scope to process, analyse and, consequently, discover relevant information in semi-structured and unstructured text [15, 16]. Some of the Text Mining achievements include the ability to generate database annotations, share information between fields and help companies with decision-making [4]. Since Text Mining overlaps the use of different tools for distinct tasks, its sub-division is not always clear. Given such fact, the present work will follow the division made by [16], which divides Text Mining into seven different groups:

1. **Information Retrieval (IR):** Commonly used by search engines like Google, Yahoo, Bing, etc. The main goal of this task is to return, from a database of documents, a list of indexed documents, which match a certain query including a logical combination of keywords, introduced by the user.

2. **Document Clustering:** It uses clustering algorithms (e.g Hierarchical clustering, k-means, Latent Semantic Indexing) to group documents into clusters using their content similarities. Depending on the method used, the number of clusters can be set by the user beforehand. Sometimes, Document Clustering is used by search engines as a preprocessing tool to IR.
3. **Document Classification:** Document Classification uses different algorithms for the specific task of assigning a label from a set of classes to each document of a given document set. This task will be explained in detail in subsection [2.1.1](#), as it plays a major role to accomplish the goal of this project.
4. **Web Mining:** With the increasing use of Internet and social media, an abundance of text-based data has emerged. Due to its dimensionality, a necessity to create a new branch oriented to the World Wide Web arose, giving rise to the field of Web Mining.
5. **Information Extraction:** Information Extraction is one of the most developed groups in Text Mining. It is the process of converting unstructured text to structured data by identifying and extracting the most relevant information and relations between topics. Information Extraction is not a trivial process to implement, since it requires complex systems with specific algorithms and software.
6. **Natural Language Processing (NLP):** NLP contains a wide range of tools (e.g., Tokenization, Part-of-speech Tagging, Lemmatization) capable of manipulating text. NLP has been increasingly used in Text Mining, also due to recent statistical techniques such as Word Embeddings. Some of these tools will be explained with more detail in subsection [2.1.4](#).
7. **Concept Extraction:** Concept Extraction, as its name suggests, aims to extract concepts from text. It attempts to group words and/or phrases by their meaning. It also contains tools aiming to retrieve any text interpretations, even though this is a challenging task for a machine.

Considering that the mentioned tasks are usually oriented to a specific case, existing general-purpose tools/software (e.g. predicting the stock market by twitter mood [17] or predicting a public protest based on big public data [18]), either lose their efficiency when applied to the Biomedical field or cannot be applied at all. To solve this problem, a new sub-field of text mining appeared: [BioTM](#)) [1, 19].

BioTM is a relatively recent field, at the 30 years old mark [20] that aims to use Text Mining tools in a biological context. Some tasks of BioTM aim to help biomedical researchers identifying biological entities and information about phenotypes, pathways, genes, proteins, disease relationship, etc., in an easier and faster way [1, 4]. Such information is searched in the different topics of biomedical literature, including Biology, Medicine, and Chemistry [21]. A second use of BioTM is the discovery of relationships between topics by gathering and connecting vast amounts of information. These relationships go sometimes unnoticed by researchers due to their complexity and multidisciplinary nature [21] (this phenomenon was named as "unseen public knowledge" by Swanson [22]). These BioTM techniques create a wide range of new opportunities and applications, such as drug discovery, study of disease emergence and improvement of diagnosis technologies [2, 23].

### 2.1.1 DOCUMENT CLASSIFICATION

Document Classification (sometimes also called as Text Classification or Text Categorization) pursues the identification of specific information or subjects within text in order to classify them into a label from a pre-defined set [24]. This is not to be confused with Document Clustering, which simply arranges documents into groups in an unsupervised way [25]. Sarkar [26] considers that the term "document" covers all the formats that can be classified by a Document Classification system, such as images, drawings, videos, recorded speech, writing, and so on.

If there are exclusively two label possibilities in a classification problem, the problem is referred to as a binary classification. If, on the other hand, there are more than two possibilities, it is referred to as a multi-class classification [27]. Document Classification can also classify different levels of text such as document level, paragraph level, sentence level and sub-sentence level [6].

Document Classification is one of the main techniques of text mining with varied applications, for instance spam filtering of emails, text indexing, web pages categorization, document sorting, text filtering, sentiment analysis, support in text summarization, etc. [5, 6, 16, 28]. For instance, regarding spam filtering, good indicators of a spam email can be simple patterns as the presence of a dollar sign ("\$\$") or a high number of exclamation points ("!!!!") [29].

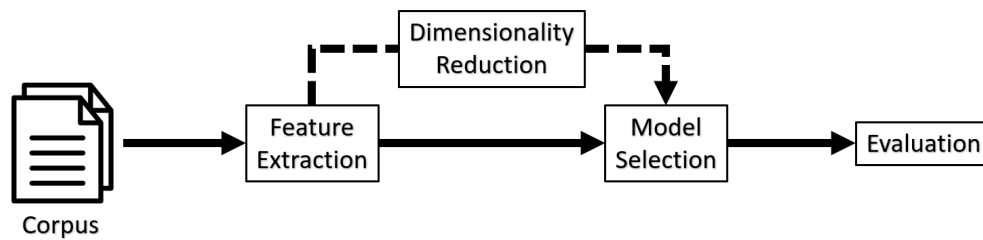


Fig. 1 – The most common Document Classification pipeline [6]

Initially, the task of Document Classification was done manually, having an associated high accuracy. However, due to scalability problems, a need for better techniques arose and rule-based techniques were implemented. This technique used manually created rules to categorize documents with certain words - a document with the words "money" and "bank" in it, should be categorized as a financial document. Yet, once again, a problem with scalability was still present since, despite having an equally substantial accuracy, the construction of lists with words related to a specific topic was a time consuming task [25].

In the last few years, Artificial Intelligence (AI) methodologies like Machine Learning (ML) models have been developed and used in Document Classification. Normally, in Supervised ML, a model learns the existing connections between features of documents with different labels to predict new instances of documents with unknown labels [24]. More recently, Deep Learning (DL), a sub-field of ML, has become increasingly important to improve the existent tools for the Document Classification task. Some AI models will be further discussed (section 2.3).

After gathering the textual data (generating the corpus), the common pipeline of a Document Classification system follows the following 4 main steps: Feature Extraction, Dimensionality Reduction (considered as an optional step), Model Selection and Evaluation (Fig. 1) [6].

### 2.1.2 PUBMED

PubMed is one of the most used scientific literature databases (with more than 28 million publications - many of them with free access - and an average of 3 million searches per day in

the year 2018 [13]). The document triage system of this repository has been evolving during the past years. Initially, it simply matched terms searched by the user to documents in the database. As result, it was possible to retrieve a list of documents, ordered from the most recently published to the oldest one. This system was efficient in organizing the newest information, but it was not appropriate in selecting the most relevant documents for a given query.

In 2013, a new technique was implemented in the PubMed system, Term Frequency - Inverse Document Frequency (TF-IDF) [30, 31], which is a feature extraction technique that takes into consideration not only the number of occurrences of the matched words ( $w$ ) within a specific document ( $d$ ), but also their frequency in all the collection of documents retrieved ( $D$ ) (Equation 1).

$$\text{TF-IDF}_{w,d} = \frac{\text{frequency}_{w,d}}{\text{Number of words in } d} \times \log \frac{D}{\text{Document frequency}_w} \quad (1)$$

Despite an improved performance over the first implemented method, and mainly due to ML methodologies showing good results in the research community, a ML model was then implemented in PubMed in 2017. Named "Best Match", the ML algorithm uses characteristics of articles such as the publication rate, history of usage, article type and an associated relevance score. This algorithm showed a more promising performance over previous ones. However, "Best Match" has its limitations. As an example, the algorithm considers that an article can be classified as more relevant if an user opens it on PubMed, which is not always an appropriate assumption. For instance, articles appearing on the front page are naturally more likely to be opened, inducing a bias in the system [13]. Recently, "Best Match" became the default sorting tool with the release of the new PubMed.

Despite the developers' intention in improving the current system by moving to a DL model, it is considered that more studies are still needed [13].

### 2.1.3 CORPORA AND IMBALANCED DATA SETS

Several methodologies and techniques have been used to get the best possible outcomes in BioTM tasks. BioCreAtIvE [32], BioNLP [33] and i2b2 [34] are some of the evaluation fora that

were created to provide a more generalized and standard evaluation of the different approaches used, providing a better overall perspective of the current BioTM community applied strategies. These competitions give access to different resources, as evaluation corpora (i.e. a set of documents manually annotated), curated dictionaries, stop words lists, among others.

For Document Triage (i.e. the process of classifying documents as relevant and non-relevant documents), the available corpora are often highly imbalanced, containing a reduced number of relevant documents [35]. When this happens, two main techniques are used to reduce the imbalance: the majority class reduction (under-sampling) or augmentation of the minority class (over-sampling). Over-sampling is achieved by replicating the existing examples or computing new ones by using ML models [36–38]. Jiang *et al.* got an F1-score of 0.75 for Document Classification when using under-sampling through the use of clustering methods [39].

#### 2.1.4 PREPROCESSING

Since ML models work exclusively with numerical vectors, text data has to be initially converted into vectors with a process called vectorization. However, before vectorization, some preprocessing steps are normally taken. A sentence splitter can be applied to break text into sentences, and also word tokenization, to split those sentences into words (also called as tokens). Sometimes, other types of tokens are used, such as characters and  $n$ -grams (overlapping combinations of  $n$  words organized by its order of appearance in text) [7].

Besides sentence splitting and word tokenization, Text Mining researchers sometimes complement this preprocessing phase with stop words removal (e.g., removing common words like 'the', 'for', 'of'), punctuation removal, stemming and lemmatization [40]. Tools with the capability of running these tasks can be found, for instance, in the Natural Language Processing Toolkit (NLTK) [41]. For traditional ML models (e.g. Naïve Bayes, SVMs, Logistic Regression), bag-of-words can be employed to make feature engineering, a process considered unnecessary for DL models [7].

Regarding the vectorization step, it can be achieved by using one-hot encoding. This method converts a previously given index for each word into resulting vectors with the size of the vocabulary used. Each vector will then contain a 1 in the respective word's index and zeros for

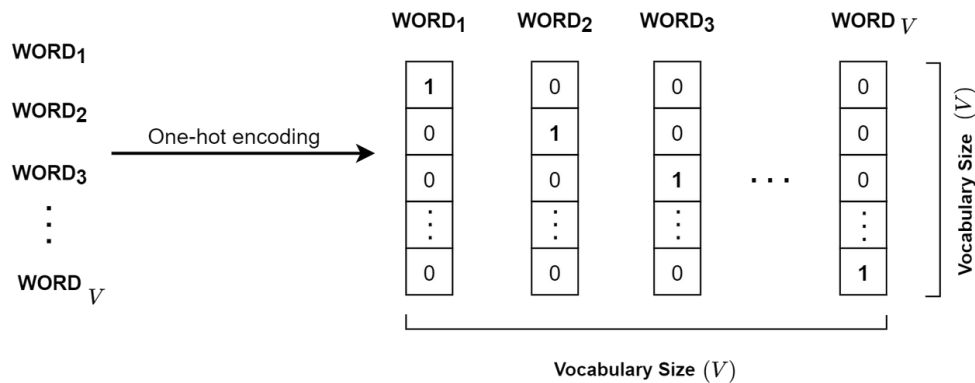


Fig. 2 – Vectorization process with One-hot encoding. One-hot encoding converts each word with an associated index from a vocabulary of size  $V$  into different vectors. Those vectors contain a 1 in the respective word's index and zeros in the remaining positions [7]

the remaining indices (Fig. 2). Due to the sparsity and high-dimensionality of these vectors, and considering that this method ignores the words' order and semantics, alternatives to one-hot encoding have emerged (e.g. token embeddings, section 2.1.5) [7, 42].

### 2.1.5 EMBEDDINGS

Token Embeddings (normally called Word Embeddings) are highly efficient distributed word representations created through a more complex word vectorization process, which results in low-dimensional dense vectors [7]. Despite the current popularity of Word Embeddings, the idea of low-dimensional word representations is more than 15 years old [43].

These vectors contain information about semantic and syntactic properties of words, meaning that semantically close words will have associated close vectors in the embedding space [42, 44]. This way, we can assume that the subtraction of the vector *Spain* to the vector *Madrid* and later sum of the vector *England* should give the vector *London*, or at least one very similar to it [45].

Word Embeddings can be trained with documents for document classification or used in their pre-trained versions to be introduced as input for a DL (or traditional ML) architecture. Furthermore, it should be noted that a Word Embedding pre-trained on a specific subject should be used for the classification of documents of similar subjects [42].



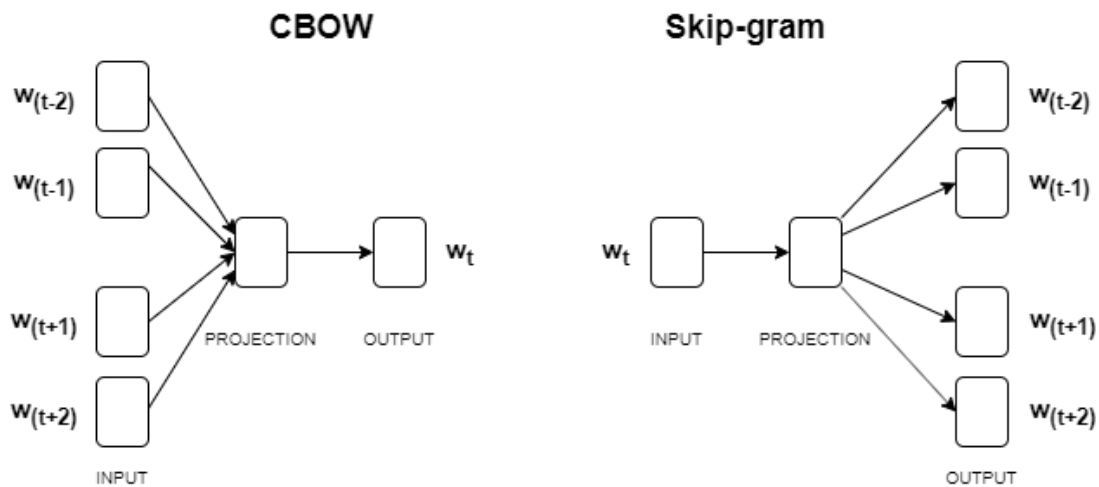


Fig. 3 – Architecture of two Word2vec's models (CBOW and Skip-gram). While CBOW outputs the center word ( $w_t$ ) using context words ( $w_{(t-2)}, w_{(t-1)}, w_{(t+1)}, w_{(t+2)}$ ), Skip-gram makes the opposite, outputting the context words using a center word. Adapted from [46].

### Word2vec

Word2vec is a group of neural network (section 2.3) based models such as the Continuous Bag-of-words (CBOW) and the Skip-gram, which are applied to generate Word Embeddings capable of recognizing linguistic patterns. Even though both models have an associated window with a certain size ( $s$ ) encompassing a number of words (where a larger size window gives better results but worse efficiency), these two models act differently [46]). While CBOW predicts a target word (also referred to as center word) through context words, Skip-gram makes the opposite, predicting the probability of the adjacent context words, using a center word ( $w_t$ ) [46, 47] (Fig. 3). These models also have a Negative Sampling version that increases efficiency by turning the models into a binary classification system. As a result of these two models, a weighting matrix located between the input and the projection layers is returned. This weighting matrix can be later used in ML models to compute word vectors.

Both these models have similar architectures, the latter showing better results in tasks such as word similarity [48]. Thus, the current project will focus on the Skip-gram model.

For each word within an input sequence ( $t = \{w_1, w_2, \dots, w_T\}$ ), the objective of the Skip-gram model is to maximize the average log probability (Objective function - Equation (2a) - with  $\theta$  representing the model's parameters) of a context word, given the occurrence of a center word

as input (Equation (2b)), where each word ( $w$ ), from the vocabulary with size  $V$ , has an input vector ( $u_w$ ) and an output vector ( $v_w$ ) representation [45, 47].

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T \sum_{-s \leq j \leq s, j \neq 0} \log p(w_{t+j} | w_t) \quad (2a)$$

$$p(w_i | w_j) = \frac{\exp(u_{w_i}^T v_{w_j})}{\sum_{w=1}^V \exp(u_w^T v_{w_j})} \quad (2b)$$

### *GloVe*

In addition to the local range window of Word2vec, GloVe implemented a global matrix ( $M$ ) containing the co-occurrence of word pairs (target word  $w_i$  - context word  $w_j$ ) to take advantage of the input text as a whole [49]. GloVe uses a weighted least squares model to generate vectors with a correlation between its space distance and its semantic distance [50]. The model is based on different equations, such as the constraint equation (3a) (each word,  $w_i$  and  $w_j$ , have an associated bias,  $bias_i$  and  $bias_j$ , which are parameters of the model), and an objective function (3b). The objective function encompasses a weighted function ( $f(M_{i,j})$ ) used to give different levels of importance to word pairs with different co-occurrence frequencies. In addition, it should be noted that GloVe can also be used for Named Entity Recognition (NER) tasks [49].

$$\log(M_{i,j}) = w_i^T \cdot w_j + bias_i + bias_j \quad (3a)$$

$$J(\theta) = \sum_{i,j=1}^V f(M_{i,j})(w_i^T \cdot w_j + bias_i + bias_j - \log(M_{i,j}))^2 \quad (3b)$$

### *Contextualized Word Representations*

Recently, a new type of Embeddings appeared: Contextualized Word Representations. Previous Word Embeddings, also called static embeddings, assigned only a vector for each word, being then unable to distinguish different possible meanings for the same word. These recent representations attempt to overcome such limitation through the use of Deep Neural Language models (such as ELMo [51] and BERT [52]). These models consider the context in which each

word appears through the allocation of a function for each word that takes into consideration the entire input sequence. These Deep Neural Language models have shown better results when compared to static Embeddings, with BERT overcoming ELMo in some tasks, through use of a Masked Language Model (MLM) that brings bidirectionality for the sequence [51, 52]. However, the training of such models can be computationally expensive, presenting a strong disadvantage to their implementation [9].

Table 1 – Some of the currently available Embeddings

Embedding	Description
Word2vec [46]	Contains well-known models as Skip-gram and CBOW
GloVe [49]	Embeddings based on a global co-occurrence matrix
ProtVec [53]	Representation of protein sequences for proteomics and genomics
Dis2vec [54]	Embeddings for disease characterization based on Word2vec
Pubmed Vectors [55]	Word2vec based embeddings trained on PubMed documents
Doc2vec [56]	Extension of the Word2vec algorithm for variable length documents
fastText [57]	Facebooks' algorithm with focus on unseen words
BioWordVec [58]	Uses subword information and MeSH terms to create embeddings on the biomedical context
"Pubmed-NCBI" [59]	Word2vec based embeddings trained on all existing PubMed abstracts in 2016
Context2vec [60]	Uses a Bidirectional LSTM to produce context-dependent representations
ELMo [51]	Generation of representations as functions of the input sentences
BERT [52]	Context word representations computed with a Bidirectional model
BioBERT [61]	BERT applied on PubMed Documents
SciBERT [62]	BERT applied on Documents from the Semantic Scholar database
BlueBERT [63]	BERT applied on PubMed Abstracts and clinical notes

### 2.1.6 NAMED ENTITY RECOGNITION

Beyond Embeddings, other features can also be extracted, including NER. In BioTM, NER is normally used to retrieve different entities such as genes, proteins, species, chemicals, etc. [64,

[65]. This process can be done in 3 different ways: using dictionaries and ontologies, which are used to support search and matching algorithms; using expression rules, which are used to fill the absence of some named entity variants by identifying possible bio-entities; and using ML (including DL [66]) models that learn how to predict certain entities [67].

For this task, we can use developed tools as @Note [68] and Pubtator [69]. For instance, Luo *et al.* [70] concatenated NER and Part-of-speech features with a Word Embedding, using it as input for a DL architecture to perform document triage. Slight improvements were observed in the models' scores presenting a higher influence of the NER features when compared to Part-of-speech. It is worth noting, however, that these improvements were not generally present in all models.

## 2.2 MACHINE LEARNING

ML is a sub-field of a bigger and well-known field called AI. AI was the first attempt to make computational algorithms capable of performing tasks traditionally exclusive to humans and which required a certain level of reasoning. AI initiated around the 1950's and it has since been an active field. ML is a more recent field, becoming more established around the 1990s. Recently, ML became the most popular sub-field of Artificial Intelligence, largely due to newer and improved hardware, together with higher amounts of available data. Despite the relation with the field of statistics, ML presents some disparities, being the data dimension one of those [7]. The popularity of ML is justified by its advantages, such as scalability and efficiency in managing demanding tasks (e.g. NLP, Document Classification, Image Classification, Speech Recognition) [71, 72]. It is to be noted that ML also presents limitations, the main one being its significant dependence on training data.

ML is normally subdivided into 4 different groups: Supervised Learning, Unsupervised Learning, Semi-Supervised Learning and Reinforcement Learning. Supervised Learning has the main goal of learning feature patterns associated with each label. On the contrary, Unsupervised Learning uses unlabeled data to perform tasks such as clustering, dimensionality reduction and data visualisation. Semi-Supervised Learning combines labeled examples with unlabeled examples on the training process with the aim of making a better model. Reinforcement Learning

includes models with different states containing associated actions. Depending on the actions, a reward can be assigned, being the final goal of these models the maximization of those rewards [73].

### 2.2.1 UNSUPERVISED LEARNING

The main characteristic in this class of ML is the use of unlabeled data sets. This class encompasses processes that aim to retrieve relevant data aspects. These processes can be divided into 3 groups: clustering (e.g. Document Clustering by different topics), dimensionality reduction (e.g. PCA and t-SNE) and outlier detection [71, 72]. These methodologies are normally used in data visualization, data dimensionality reduction, noise removal and interpretation of possible data correlation. Such correlations may ease the understanding of the entire data set. Considering the various existing applications, Unsupervised ML can be used as a previous step to a Supervised ML model [7].

Since Dimensionality Reduction has the capacity to represent data in fewer dimensions, the current project will make use of these functionalities to visualise and interpret data. Principal Component Analysis (PCA), non-negative matrix factorization, and recent approaches such as t-Distributed Stochastic Neighbor Embedding (t-SNE) and autoencoders, are some of the preferred dimensionality reduction approaches [6]. For generic data visualisation, Dimensionality Reduction aims to reduce the number of the data set dimensions to either two or three dimensions, while simultaneously maximizing relevant information. These new dimensions ease data visualization with, for instance, a scatter plot.

#### *Principal Component Analysis (PCA)*

PCA was initially created by Pearson in 1901 [74] and later developed by Hotelling [75]. Using data transformations through singular value decomposition (SVD) or the eigenvectors of covariance, PCA creates orthogonal variables (Principal Components), which represent the existing correlations between variables [76]. These Principal Components vary from highest to lowest variance explained in the data (PC1 will explain a higher proportion of the variance than PC2, which will explain in turn a higher variance than PC3, and so on) [73]. Moreover, one Principal

Component (e.g. PC2) will explain data variance that will not be present in the previous Principal Component (e.g. PC1). Some significant limitations of PCA include its dependence on data and immutability for the user, the latter making it impossible to tune with *a priori* knowledge [76].

#### *t-Distributed Stochastic Neighbor Embedding (t-SNE)*

t-SNE [77, 78] is an improved Dimensionality Reduction version of an older algorithm called Stochastic Neighbor Embedding (SNE) [79]. SNE gives a spatial location to dataset points into a two or three dimensional map. Secondly, it calculates Euclidean Distances which are then converted to conditional probabilities using a Gaussian Distribution. These conditional probabilities are a representation of the similarities between points (i.e. similar points will have a higher conditional probability). Despite satisfactory results, this algorithm presented limitations. A prime example is the optimization process and what Maaten and Hinton [77] call a "crowding problem", referring to points overlay due to high proximity. To solve this particular limitation, t-SNE implemented the Student-t distribution on a low-dimensional map and changed the cost function used. t-SNE then presented an improved data visualization. Still, the major limitation of t-SNE is its inability to transform data into more than three dimensions. Yet, this is not significantly problematic, since t-SNE is mostly used for low-dimension data visualization [78].

#### *Autoencoders*

Autoencoders were implemented using neural networks (section 2.3.1) for dimensionality reduction. The fundamental idea of Autoencoders is the use of a hidden layer with fewer neurons between same-length input and output layers [80]. This step forces the algorithm to learn a low-dimensional input representation. Even though the output has equal length as the input, it is expected to solely keep the most relevant features of the input [81]. Autoencoders were firstly implemented by Rumelhart *et al.* in 1985 [82], and a significant progress was made by 2006 when compared to PCA [83]. Recently, beyond Dimensionality Reduction and Feature Learning, Autoencoders have been used to improve and accelerate the data processing step [6]. The main limitation of Autoencoders still remains on their reliance on vast amounts of available data [6].

### 2.2.2 SUPERVISED LEARNING

In contrast to Unsupervised Learning, Supervised Learning relies on a labeled data set. The used data sets should contain a set of feature vectors with associated annotations. Upon completion of the training process, a new instance on the form of a feature vector can be introduced to predict its result. Depending on the final label type, supervised learning models can be categorized, for instance, as a classification (i.e. discrete variables) or a regression (i.e. continuous variables) model, the two most frequent tasks of Supervised Learning [7, 84]. Examples of algorithms used in Document Classification include Naïve Bayes, Logistic Regression, Decision Trees, Support Vector Machines, Ensembles (e.g. Bagging and Boosting), k-nearest Neighbors, Random Forests and Neural Networks (mainly for DL) [5, 6].

### 2.2.3 EVALUATION OF MACHINE LEARNING MODELS

Supervised ML classifiers can be evaluated by a comparison between the models' predictions and the real values. For binary classification problems, this leads to values for True Positives (TP), True Negatives (TN), False Positives (FP) (Type I error) and False Negatives (FN) (Type II error). In this project, relevant documents and non-relevant documents will be positive and negative instances, respectively. Those instances are considered True when the model hits a prediction.

Based on these values, frequently seen as a confusion matrix, one can compute different metrics such as Accuracy (Equation (4a)), Recall (Equation (4b)), Precision (Equation (4c)), F1-score (Equation (4d)) [85] and Matthews correlation coefficient (MCC) (Equation (4e)). All these metrics range between 0 and 1 (a score of 1 meaning the best result possible), with exception of MCC which ranges between negative and positive unity and is often applied to imbalanced data sets [39].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4a)$$

$$Recall = \frac{TP}{TP + FN} \quad (4b)$$

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

$$F1 = 2 \times \frac{(Precision \times Recall)}{(Precision + Recall)} \quad (4d)$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (4e)$$

In the case of document ranking, a metric commonly used is Average Precision (Equation (5)), which calculates the average precision of documents classified as relevant ordered by their confidence score, with  $n$  representing all the classified documents [73]. Thus, this metric is able to harshly penalize  $FP$  on the top of the retrieved rank and highly reward  $TP$  on those same positions.

$$Average\ Precision = \frac{\sum_{k=1}^n (Precision(k) \times relevance(k))}{|\{relevant\ documents\}|} \quad (5)$$

Frequently, in binary classification systems, these metrics may be complemented with Receiver Operating Characteristic (ROC) curves [86], Precision-Recall (PR) curves and their associated Area Under the Curve (AUC) measure [87, 88]. ROC plots True Positive Rate (TPR) (also known as Recall) against False Positive Rate (FPR) ( $FPR = \frac{FP}{FP+TN} = 1 - specificity$ ), whereas PR plots Precision against Recall. In cases of data imbalance containing few positive documents (relevant documents), ROC curves may give misleading results, once ROC curves do not consider  $FN$  instances. Because of that, in those cases, PR curves should be used instead [84]. Both ROC and PR curves have an associated AUC measure that is frequently used to compare distinct classifiers. This measure can range from 0 to 1 with a score of 1 being the best possible classifier.

All these metrics are used to compute both training and generalization errors of the models. To that end, the data set has to be previously divided into a larger set and a smaller set for training and test purposes, respectively. To provide an unbiased score of the model, the test set may never be used in its training process. For the hyperparameter optimization, a small training sub-set may also be used as validation set [81]. Commonly, a validation technique named  $k$ -fold cross-validation is applied to provide a more realistic and robust model evaluation.



This technique splits the training set into  $k$  non-overlapping sub-sets. Subsequently, for each interaction in a total of  $k$  interactions, cross-validation technique uses one sub-set as test set and the remaining as training set, computing for each interaction the resultant evaluation metrics.

## 2.3 DEEP LEARNING

DL, a sub-field of ML (Fig. 4), has recently achieved impressive results, becoming the sub-field of ML with most significant progress in the past years [7]. Through the implementation of Deep Neural Network (DNN)s, improved solutions were found to classical Artificial Intelligence problems, such as image classification, speech recognition, handwriting transcription, autonomous driving and search results [7, 89].

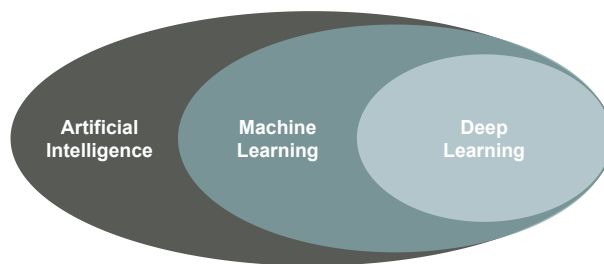


Fig. 4 – Deep Learning’s location in the field of Artificial Intelligence. Adapted from [7]

Unlike traditional ML algorithms, DL does not need the extensive and time-consuming process of feature engineering, since architectures of DL have the capacity to learn data representations [7]. In addition, those data representations may even contain hidden interactions, which could pass unnoticed with handcrafted features. Moreover, DL even provides scalability, complexity, architecture adaptability and most importantly, improved performance in diverse tasks [6]. Nevertheless, DL also has downsides, for instance, its requirement of large amounts of available data. Besides, the use of large quantities of data makes the models’ training process computationally expensive and consequently time-consuming (Table 2).

Table 2 – Deep Learning's Advantages and Limitations [6, 7]

Advantages	Limitations
<ul style="list-style-type: none"> <li>- No need for extensive feature engineering</li> <li>- Architecture adaptability               <ul style="list-style-type: none"> <li>- Scalability</li> </ul> </li> <li>- High complexity and abstraction</li> </ul>	<ul style="list-style-type: none"> <li>- Requirement of large amounts of data</li> <li>- Long training process</li> <li>- Difficulty to choose structure and architecture</li> <li>- Uninterpretable hidden layers' output (in most cases)</li> </ul>

### 2.3.1 NEURAL NETWORKS

Despite the recent DL popularity, the first article related to an Artificial Neural Network (ANN) is now more than 70 years old [90]. Not long after that publication, in 1957, an artificial neuron, Perceptron, was created [91]. Perceptrons would be of enormous importance, as they are one of the basis of the currently used complex ANNs.

The Perceptron (Fig. 5) first receives an input vector  $(x_1, x_2, \dots, x_n)$  and an weight vector  $(w_1, w_2, \dots, w_n)$ . The weights are present on the connections between the inputs and the node of the perceptron. The node then sums the weighted inputs  $Z = (w_1x_1 + w_2x_2 + \dots + w_nx_n)$ , which will become the input to the step function ( $f$ ). Initially, two step functions were normally used for the Perceptron: Heaviside step function and sign function (Equation (6) and Equation (7), respectively). The result of the step function represents the output of the Perceptron:  $h_w(x) = step(x^T w)$ .

$$heaviside(z) = \begin{cases} 0, & \text{if } z < 0 \\ 1, & \text{if } z \geq 0 \end{cases} \quad (6)$$

$$sgn(z) = \begin{cases} -1, & \text{if } z < 0 \\ 0, & \text{if } z = 0 \\ +1, & \text{if } z > 0 \end{cases} \quad (7)$$

Perceptrons, however, presented substantial limitations in solving certain types of nonlinear problems (e.g. XOR problem). It was later discovered that such limitations could be surpassed by stacking multiple Perceptrons, giving rise to the Multilayer Perceptron (MLP) [84].

MLPs are ANNs which can present one or more layers between the input and output layers. These layers, named hidden layers, produce outputs which are normally not seen/used by the

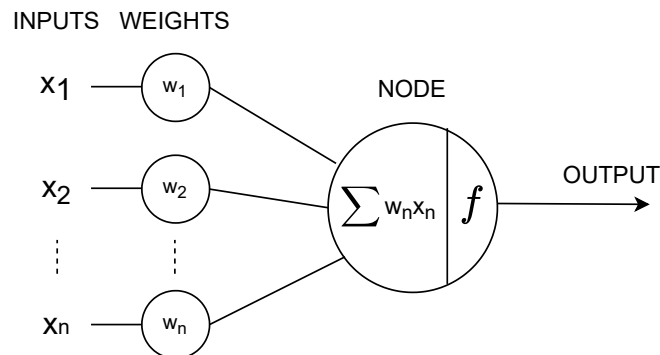


Fig. 5 – Diagram of the perceptron architecture (with  $f$  representing the step/activation function). Adapted from [92]

user, a characteristic that gives ANNs their known "black box" nature. With exception for the output layer, all the remaining layers of an ANN (hidden layers and input layer) present an additional bias neuron (normally  $x_0 = 1$ ) with an associated weight [84]. Those weights and bias values are the parameters of the ANN which are after adjusted in the training process [7]. Once step functions of Perceptrons did not have the required gradients for the training process, MLPs had to replace those step functions with activation functions, such as sigmoid function ( $\sigma(z) = \frac{1}{1+e^{-z}}$ ) and hyperbolic tangent function ( $\tanh(z) = 2\sigma(2z) - 1$ ) (Fig. 6). In addition, those activation functions provide non-linearity to models, resulting in a significant increase of complexity and abstraction of models. This fact facilitates the approximation of DL models to real life problems [84, 89].

In 1986, a well-known training algorithm emerged, named Back-Propagation [93]. Back-propagation is divided in two main steps: the forward pass and the backward pass. Firstly, the forward pass computes an output value. Then, through a cost function (e.g. Mean Squared Error (MSE)), Back-propagation measures the error obtained on the output's prediction by comparing it with the desired output. After that, the algorithm goes from the ANN's output layer to the input layer (backward pass), through a chain rule process, to measure the connections' influence on the previously measured error. In the end, Back-propagation relies on an optimization algorithm (e.g. gradient descent) to adjust the model's parameters according to their previously measured influence, in order to minimize the difference between the predicted and real values [7, 84].

Despite the successful implementation of Back-Propagation, ANNs were not as popular as they are now. This may be explained by the appearance of successful SVMs in the field

of ML. The current popularity of ANNs is heavily influenced by recent works [94] that used deeper architectures, named DNNs. For instance, Krizhevsky *et al.* [95] got an improvement of more than 10 percentage points of accuracy, using a CNN, which is a type of DNN, for image classification on the ImageNet competition. With the DNN's emergence, other activation functions appeared, such as Rectified Linear Unit (ReLU) ( $ReLU(z) = \max(0, z)$ ) (Fig. 6), becoming the most popular function used in the last years [89]. Likewise, other alternatives regarding Back-propagation functions appeared. For example, loss functions such as cross-entropy and hinge [42]) and optimization algorithms such as Stochastic Gradient Descent and Adam Optimizer [96]).

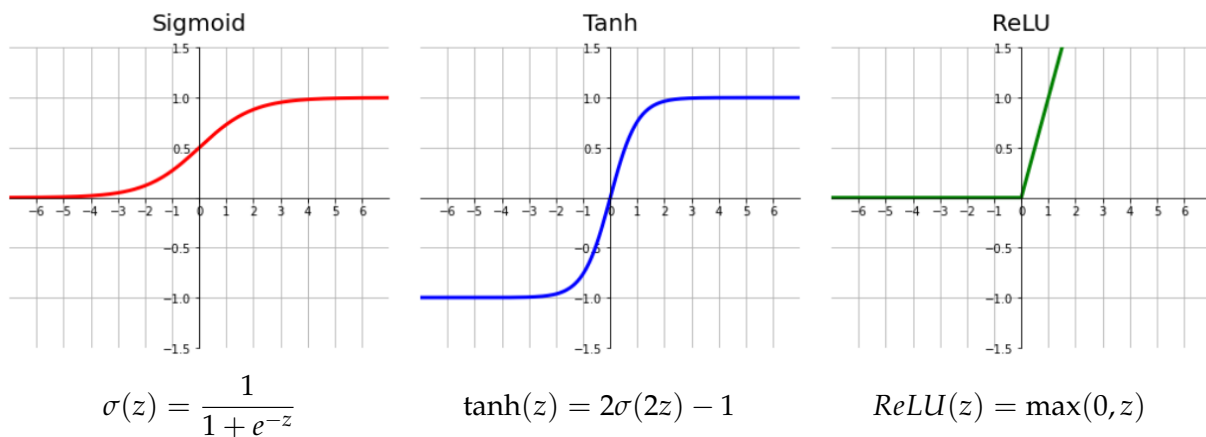


Fig. 6 – Representation of the most commonly used activation functions.

The selection of both loss and activation functions for the output layer depend directly on the task [7], being the most common choices presented in Table 3.

Table 3 – Selection of last-layer activation and loss functions. Source: [7]

Problem Type	Last-layer activation	Loss Funtion
Binary classification	sigmoid	binary_crossentropy
Multiclass, single-label classification	softmax	categorical_crossentropy
Multiclass, multilabel classification	sigmoid	binary_crossentropy
Regression to arbitrary values	None	MSE
Regression to values between 0 and 1	sigmoid	MSE or binary_crossentropy

Since, in this work, we will be dealing with a binary classification problem, we will then have to use a `sigmoid` function as activation for the output layer and `binary_crossentropy` as loss function. This function is commonly computed with the following Equation (8), where  $N$  represents the total number of training examples and  $y_{i \text{ pred}}$  represents the predicted probability for a certain instance ( $i$ ).

$$J = -\frac{1}{N} \sum_{i=1}^N y_{i \text{ real}} \times \log(y_{i \text{ pred}}) + (1 - y_{i \text{ real}}) \times \log(1 - y_{i \text{ pred}}) \quad (8)$$

### 2.3.2 TRAINING PROCESS

Training a Deep Neural Network is a complex and computationally expensive process that consists in 3 main steps:

1. Send the input examples through the network to get the predictions;
2. Compute the loss function by comparing predictions with their real values;
3. Adjust weights of the model in order to minimize the difference between the predicted and real values.

When this *training loop* goes through all the training examples it finishes an *epoch* (the number of epochs is a hyperparameter of the model that must be fine-tuned) [7]. Moreover, on the first epoch, model weights are commonly randomly initialized, following certain distributions. Furthermore, and as previously mentioned, the adjustment of weights/parameters can be done by using distinct optimization algorithms. These algorithms can be used to either minimize a function  $f(x)$  or maximize it by minimizing  $-f(x)$ .

In addition, these algorithms have different associated parameters that change their behaviour, with the most important one being learning rate  $\eta$ , which determines the step size of the update performed on weights. The learning rate is a really important parameter when training models since a too small  $\eta$  may result in a really slow learning process, while a too large  $\eta$  may not converge to a minimum. Some of the currently available algorithms are [81, 97, 98]:

- **Batch/Vanilla Gradient Descent:** Updates model parameters ( $\theta$ ) by computing gradients for the entire training set ( $\theta = \theta - \eta \times \nabla_{\theta} J(\theta)$ ). This optimizer has the advantage of

ensuring the convergence to a global minimum in a convex error surface or local minimum in the case of a non-convex error surface. This process is normally slow, since it needs to compute gradients on the entire dataset to make only one update. Additionally, this algorithm does not work on large datasets that cannot fit in memory. For these reasons, other alternatives emerged to optimize this implementation.

- **Stochastic Gradient Descent (SGD):** Algorithm that optimizes Batch Gradient Descent by computing gradients for each training example, instead of using the entire dataset at once ( $\theta = \theta - \eta \times \nabla_{\theta} J(\theta; x^{(i)}; y^{(i)})$ ). Despite being more efficient, this algorithm may not converge to a minimum since it keeps overshooting, which introduces instability to the model. Nevertheless, this can be countered by using a smaller learning rate that will approximate SGD to the Batch Gradient Descent behaviour.
- **Mini-batch Gradient Descent:** An optimizer that aims to get a balance between the two optimizers above by using mini-batches of the training set (i.e. non-overlapping sub-sets of  $n$  training examples) ( $\theta = \theta - \eta \times \nabla_{\theta} J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$ ). When using mini-batches of small size, instability can also be introduced in the model, which can be countered by fine-tuning the mini-batches size. In this way, it is possible to achieve the best balance between quality and efficiency. When using mini-batches, an epoch is divided in  $k$  iterations, calculated by  $k = \text{number of training examples} / \text{batch size}$ . Fundamentally,  $k$  represents the number of times that parameters are updated. This means that for SGD, once it updates parameters for each training example, the number of iterations will be equal to the number of training examples. Moreover, and since Mini-batch Gradient Descent and SGD can still be a little slow and have high fluctuation when training a model, momentum would be posteriorly introduced in these algorithms to accelerate and smooth the training process. Basically, momentum consists of using the history of previous gradients and an associated hyperparameter  $v$  with the latter determining the influence of those previous gradients when computing new ones.
- **Adaptive Gradient Algorithm (AdaGrad):** Contrarily to the above optimizers, where the learning rate is practically fixed, AdaGrad learning rate  $\eta$  is adjusted based on the existing parameters (i.e. sets a larger learning rate for parameters that are distant from the optimum and a smaller learning rate for parameters that are near to the optimum). Thus, AdaGrad is highly recommended when handling sparse data. AdaGrad updates model parameters by

using Equation (9), where  $\theta_i$  represents each parameter at a certain time-step  $t$ . To adjust the learning rate, this algorithm uses the computed gradients on the previous time-step and a constant  $\epsilon$  that prevents a division by zero.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \times \nabla_{\theta_i} J(\theta_{t,i}) \quad (9)$$

This automated process of adjusting learning rate eliminates the need to fine-tune this hyperparameter, when using this algorithm. However, due to the tendency of the denominator to increase with each time-step, the learning rate can evolve to a overly small value, which can make the model to stop learning sooner than expected.

- **Root Mean Square Propagation (RMSprop):** this optimizer aims to eliminate the problem of [AdaGrad](#) learning rate becoming extremely small by using a different equation to adjust the learning rate (Equation (10), where  $g_t$  represents  $\nabla_{\theta_t} J(\theta_t)$ ), with 0.9 being the recommended value for the hyperparameter  $\gamma$ .

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{(\gamma E[g^2])_{t-1} + 0.1g_t^2} + \epsilon} \times g_t \quad (10)$$

- **Adaptive Momentum Estimation (Adam):** a more recent optimizer that combines the learning rate adjustment technique from [RMSprop](#) with momentum. This way, this algorithm provides a faster convergence to an optimum and an automated adjustment of the learning rate throughout the model training. To use [Adam](#), two more hyperparameters are needed:  $\beta_1$  and  $\beta_2$ , with the recommended values being 0.9 and 0.999, respectively. The equations used by [Adam](#) are (11), where the numerator of  $\hat{v}_t$  represents the exponentially decaying average of previous squared gradients and the numerator of  $\hat{m}_t$  represents the same but for "non-squared" gradients.

$$\hat{m}_t = \frac{\beta_1 m_{t-1} + (1 - \beta_1) g_t}{1 - \beta_1^t} \quad (11a)$$

$$\hat{v}_t = \frac{\beta_2 v_{t-1} + (1 - \beta_2) g_t^2}{1 - \beta_2^t} \quad (11b)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \times \hat{m}_t \quad (11c)$$

- **AdamW**: a variation of the Adam algorithm that allows the implementation of weight decay, a regularization technique that will be explained in subsection 2.3.3. The authors of this algorithm argue that, contrarily to SGD, the common weight decay does not perform as it is supposed to, when using Adam. For that reason, they implemented a variation called "decoupled weight decay" ( $\lambda\theta_t$ ), which can be seen in Equation (12).

$$\theta_{t+1} = \theta_t - \eta \times \left( \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} + \lambda\theta_t \right) \quad (12)$$

### 2.3.3 REGULARIZATION

Two important concepts when training a ML model are *underfitting* and *overfitting*. *Underfitting* refers to the stage where a model is still learning the patterns of interest to make predictions on unseen data. While, on the opposite, *overfitting* refers to the stage where a model starts to learn too specific patterns (noise) from the training data that are not relevant for the prediction of new instances. When training a model, it is then important to find the best trade-off between these two phenomena (Fig. 7). The model capacity to predict new instances correctly is also called *generalization*. In general, ML models tend to *overfit* after some training,

which in the case of complex DL models, can be even more preponderant. So, it is of importance to control *overfitting* in those cases, which can be done with different regularization techniques, with the most simple one being the reduction of model complexity [7].

#### L1 and L2

Another option to reduce overfitting is by making *weight regularization*. This can be done with both *L1 regularization* and *L2 regularization*, the latter being sometimes called as *weight decay*. These techniques apply a regularization term to the loss function with the purpose of decreasing

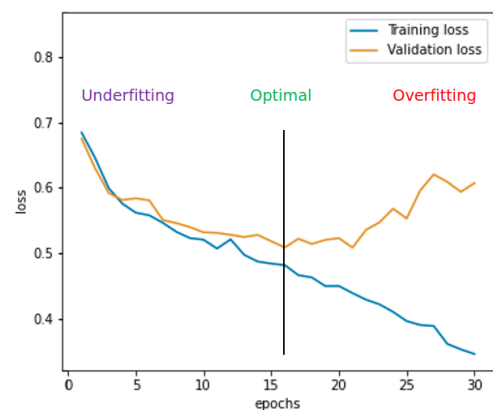


Fig. 7 – Graph representation of *underfitting* vs. *overfitting*. Adapted from [81]



the weights' scale. While *L1 regularization* uses the absolute value of the weight coefficients proportionally as regularization term, *L2 regularization* uses as regularization term the square value of those same coefficients [7, 81].

### Dropout

Dropout is a really popular regularization technique that tries to prevent a model from learning too specific patterns from the training data by randomly removing a fraction of neurons in a specific layer (Fig. 8). This is commonly done by converting a set fraction of outputs of a specific layer to zero. However, since dropout is only applied while training a model, when testing the model the inputs of layers trained with dropout need to be scaled using the same fraction used when training these layers. A huge advantage of dropout, beyond its commonly high positive impact, is the fact of being a computationally inexpensive technique [7, 81].

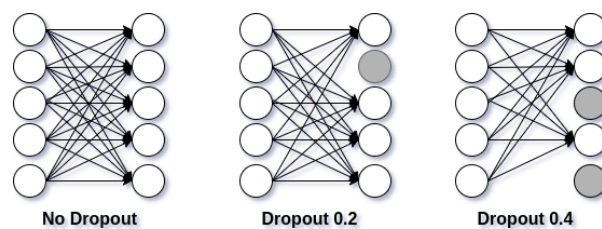


Fig. 8 – Dropout representation using different fractions.

### Early Stopping

Another commonly used technique to prevent overfitting is the application of early stopping during model training. Basically, early stopping consists on the use of a validation set to evaluate the model generalization capacity after each epoch. Early stopping acts on the model by stopping training when the evaluation metric used (commonly the cost function) is declining on the validation set for a certain number of epochs (this number of epochs has to be previously set and is commonly know as patience). This technique is also usually combined with `ModelCheckpoint` that can be used to either save the model weights after each epoch or to only save model weights after the epoch that got the best result on the validation set [7].

### 2.3.4 CONVOLUTIONAL NEURAL NETWORKS

**CNNs** are a specific type of Feedforward Neural Networks with the significant advantage of reducing the number of train parameters over a common **DNN**. Current **CNNs** were first introduced in 1990 by Lecun *et al.* [99], with the development of a **CNN** capable of classifying images of handwritten digits. Relying on the Back-Propagation algorithm to adjust the model parameters, this **CNN** obtained a low error rate of 1.1% for the image classification task. Most recently, **CNNs** have been similarly shown satisfactory results regarding Text Mining tasks [100], such as semantic parsing, search query retrieval, and especially, text classification.

Traditionally, a Convolutional Neural Network (**CNN**) comprises convolution layers, pooling layers and one, or several, dense layers. Convolution and dense layers rely on associated activation functions (commonly **ReLU** for convolution layers and **ReLU**, sigmoid or softmax ( $softmax(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^K \exp(z_j)}$ ) for dense layers). Later on, combinations of these layers resulted in well-known architectures, including LeNet-5 [101], AlexNet [102], GoogLeNet [103], VGGNet [104], and ResNet [105].

#### *Convolution Layer:*

Contrarily to a Dense layer, Convolution searches for local rather than global patterns. With this strategy, a convolution layer has a higher capacity to recognize, in arbitrary locations, a pattern found elsewhere. These layers contain a convolution operation ( $*$ ), which converts the input tensor ( $I$ ) into an output tensor ( $O$ ). To achieve that, convolution uses a Filter ( $F$ ), referred to as Kernel, containing a set of weights, which will be adjusted during the model training process. In a **CNN**, the weight matrix, or parameters, which can include an additional bias, is always smaller than the input tensor. In a 2D convolution, the output with a size of  $(x, y)$  is obtained with the Equation (13) [81].

$$O(x, y) = (I * F)(x, y) = \sum_i \sum_j I(i, j) F(x - i, y - j) \quad (13)$$

In Document Classification, 1D Convolutions are commonly applied. Although the input tensor is conventionally a 2D vector (e.g a matrix with  $n$  words and a Word Embedding of size  $j$  (Equation (14))), the filter is solely applied to the word related variable (Fig. 9) [106, 107].

$$O(x) = (I * F)(x) = \sum_{a=-\infty}^{\infty} I(a)F(x - a) \quad (14)$$

Applying an activation function (*Activation*) and a filter ( $F$ ) with a certain window size ( $w$ ) to an input sequence ( $I$ ), and then adding a bias ( $bias$ ), we obtain the following Equation (15) which is computed for every input window [70].

$$C = Activation(F \bullet I_{n:n+w-1} + bias) \quad (15)$$

Beyond filter size, convolution layers have other parameters, including padding and stride. Padding refers to values that can be added around the input tensor with the purpose of getting an output with the same size of the input. A common example is the zero padding, where all the added values are zero. Stride refers to the step that the filter takes when it goes through the input tensor. In a 2D convolution, stride may have different values for width and height [7].

Convolution layers contribute advantageously to CNNs, once these layers contain sparse weights by using a small filter. This causes the convolution layers to reduce the output dimensions but, at the same time, maintain the most important features. A second advantage is parameter sharing, which results from the use of the same filter across all the input tensor. This eases the training process as the number of learning parameters has lowered. It is also worth mentioning equivariant representations, which causes that a shift on an input be reflected on the respective output [7]. Lastly, it is important to note the major disadvantage of convolution layers, namely their difficulty in determining filter dimensions [108].

#### *Pooling Layer:*

Currently, several pooling operations are available, including max pooling, average pooling,  $L^2$  norm, weighted average pooling, among others.

Max pooling and average pooling are two commonly used poolings. Similarly to a convolution operation, pooling may present different sized dimensions, and a stride option, which acts in the exact same way. Max pooling (normally, the most successful pooling [7]) computes a

mathematical operation for each window with a predefined size, outputting the highest value within that window (Fig 9).

Pooling can be interesting when we are more interested in the presence of certain features rather than their location. This is possible due to the representation invariance of pooling, i.e. the learned function is flexible to variations in small translations [81].

Moreover, pooling has further advantages including dimensionality reduction, "dimension standardizer" (i.e. it can limit the size of different sized inputs), and improved training process efficiency, since this layer does not rely on learning parameters [81]. Examples of pooling disadvantages include the tendency to lose relevant information and the fact that invariance is not always desired [84]. In Document Classification, max pooling can be useful in the identification of semantic keywords by the DL model [109].

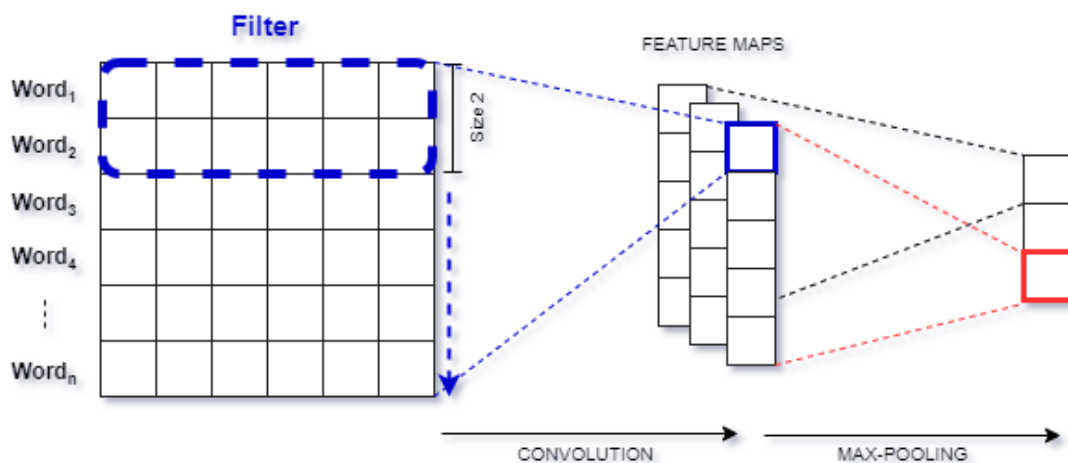


Fig. 9 – CNN operations such as 1D Convolution on a Word Embedding matrix and 1D Max pooling on the resulting feature maps from the Convolution step. Adapted from [110]

### 2.3.5 RECURRENT NEURAL NETWORKS

One problem with CNN and DNN models is that they receive the input as a whole, not taking into account the order of appearance of the input elements [7]. For sequential problems, such as text (i.e. a sequence of words/characters), these models can result in loss of information which

would, otherwise, be obtained through the words' order [73]. Therefore, RNNs have been used, often together with CNNs [9], to overcome this obstacle.

RNNs make use of a loop which iterates over its different hidden states (Fig. 10). In a simple RNN, for each given state, an output can be computed using the previous state. After, the same state is updated with the resulting output. This mechanism simulates a memory, where each output is calculated with information that was previously seen by the model [7]. RNNs have different applications such as sequence labeling, classification and generation [73].

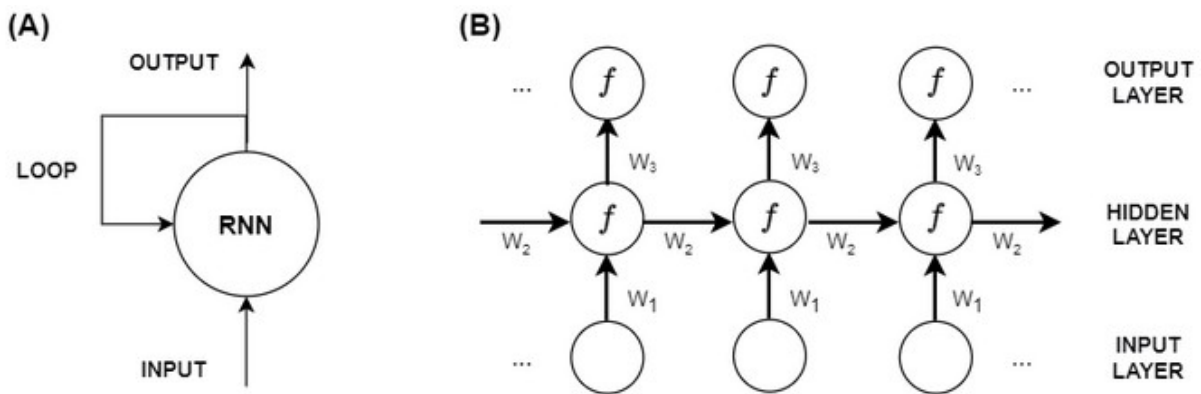


Fig. 10 – RNN model's architecture (A) and its unfolded version (B).  $f$  represent the activation functions and  $W$  the different weight matrices of the model. Adapted from [7] and [111], respectively.

RNNs can be divided into different types of architectures, regarding the number of inputs/outputs (e.g., one-to-many, many-to-one and many-to-many). The architecture is selected according to the desired task. In a many-to-many case, three matrices are used: firstly, one between the input and the hidden state ( $W_1$ ); secondly, one between the hidden states ( $W_2$ ); and thirdly, one between the hidden state and the output ( $W_3$ ). These matrices, coupled with two bias vectors ( $bias_1, bias_2$ ), are the parameters that will then be learned by the model (often using the Back-Propagation Through Time algorithm [112]). The forward propagation of a many-to-many model, starts with the hidden state at time-step 0 ( $s^{(0)}$ ) with a zero value. For consequent time-steps, the following equations are applied (16) using activation functions (e.g. tanh, softmax (*Activation*)) and an input for each time-step ( $i^t$ ) [81]:

$$z^{(t)} = W_1 \times i^{(t)} + W_2 \times s^{(t-1)} + bias_1 \quad (16a)$$

$$s^{(t)} = Activation(z^{(t)}) \quad (16b)$$

$$o^{(t)} = W_3 \times s^{(t)} + bias_2 \quad (16c)$$

$$\hat{y}^{(t)} = \text{Activation}(o^{(t)}) \quad (16d)$$

RNNs present several positive characteristics including its elasticity to different sized inputs and different data types [111]. However, it was found that RNNs struggle to store long-term sequences, due to a vanishing and exploding gradient problem [113, 114] when training the model. RNN variations, such as LSTM and GRU were later developed to tackle this problem. Bidirectional RNNs [115] also emerged to enable the processing of the input sequence in both directions. This property is particularly relevant in different scenarios such as Document Classification [9]).

### 2.3.6 LONG SHORT-TERM MEMORY

LSTMs [116] are a type of gated RNNs, with improved efficiency over standard RNNs [81]. Such improvement results from the implementation of memory blocks (Fig. 11) containing multiplicative gates and one or more memory cells, each with an associated recurrent self-loop. This technique solves the vanishing gradient problem in RNNs [111].

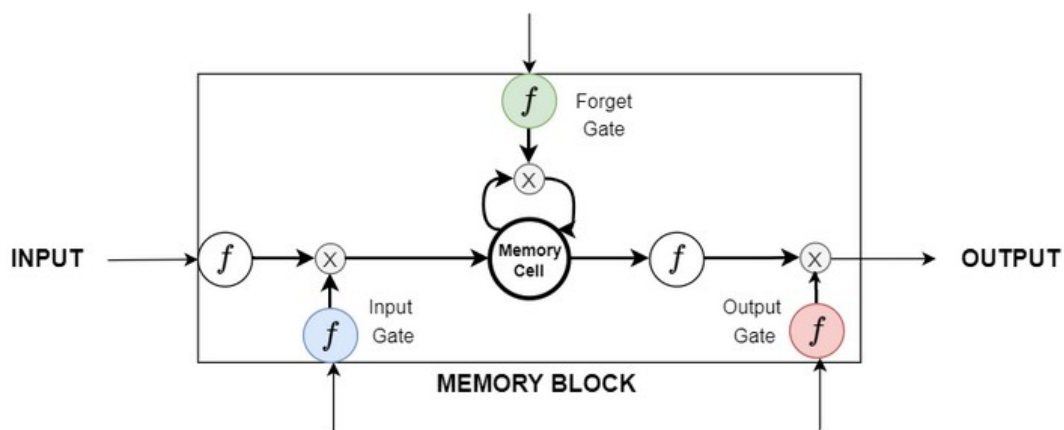


Fig. 11 – Representation of a memory block with a single memory cell from a LSTM model. This memory block contains an input gate, a forget gate and an output gate. The circles with the X represent the multiplication operations that occur within the memory block. Peephole connections are not present for simplicity purposes. Adapted from [111]

Initially, LSTMs comprised merely two nonlinear gates: the input gate and the output gate. Later on, some other improvements were implemented, such as the nonlinear forget gates [117]

and the peephole connections [118]. All these gates allow the use of data inputs from previous states of the model, as information for later states. Due to the absence of the vanishing gradient problem, **LSTMs** reached the ability to carry information for a larger number of states, when compared to **RNNs** [7, 111].

Similarly to **RNNs**, **LSTMs** have an initial state with a zero value. For the following states, different equations are recurrently applied to each memory block [111]. Despite having particular values for the model parameters (as weight matrices ( $W_1, W_2, W_3$ ) and bias vectors  $bias$ ), all the 3 gates (input ( $g_1$ ), forget ( $g_2$ ) and output ( $g_3$ ) gates) are computed for each memory block  $n$  and time-step  $t$ , using the same formula (Equation (17)).

$$g_n^{(t)} = Activation\left(\sum_j W_{1n,j}^g i_j^{(t)} + \sum_j W_{2n,j}^g s_j^{(t-1)} + bias_n^g\right) \quad (17)$$

Regarding the cell memory ( $c_i^{(t)}$ ), Equation (18) is used to update its state.

$$c_n^{(t)} = g_{2n}^{(t)} c_n^{(t-1)} + g_1^{(t)} Activation\left(\sum_j W_{1n,j} i_j^{(t)} + \sum_j W_{2n,j} s_j^{(t-1)} + bias_n\right) \quad (18)$$

The cell output ( $s_n^{(t)}$ ) is computed by Equation (19), where ( $g_3$ ) refers to the output gate.

$$s_n^{(t)} = Activation\left((c_n^{(t)}) g_{3n}^{(t)}\right) \quad (19)$$

Frequently, the activation function (*Activation*) applied to the gates is the logistic sigmoid, where a result of 0 represents a closed gate and 1 an opened gate. For the cell input and output, beyond logistic sigmoid, Tanh can also be applied [111]. Although **LSTMs** can have an equal number of inputs/outputs of an **RNN**, due to necessary weight matrices and bias for each gate, the total number of parameters in the model is consequently higher [81].

Variations of standard **LSTMs** have also been used, including Deeper **LSTMs** [119, 120] and Bidirectional Long Short-Term Memory (**Bi-LSTM**) [121, 122]. For instance, Zhou *et al.* [123]

implemented a feature selection algorithm for relation classification using a Bi-LSTM with an attention mechanism.

### 2.3.7 GATED RECURRENT UNITS

Due to the complexity and high number of parameters, LSTMs can be challenging to train. GRUs [124], with its lower number of multiplicative gates, are simpler alternatives to such models [125]. Despite their simplicity, GRUs can compute similar results when compared to LSTMs [126–128]. Furthermore, GRUs can also show a better detection of long-term dependencies than LSTMs [129].

GRUs rely solely on two gates: an update gate ( $g_4$ ) and a reset gate ( $g_5$ ). Similarly to LSTMs, these gates have their own values for the model parameters ( $W_1, W_2, bias$ ) and are computed with the Equation (20).

$$g_n^{(t)} = Activation\left(\sum_j W_{1n,j}^g i_j^{(t)} + \sum_j W_{2n,j}^g s_j^{(t-1)} + bias_n^g\right) \quad (20)$$

These gates can select information for the next states, i.e. an update gate of 1 means that information of the previous states stays preserved, while for a value of 0, the target state information is the one prevailing. A reset gate selects, by non-linearity, which information of the current state is kept to the next state [81]. The output ( $s_n^{(t)}$ ) is computed by Equation (21). As for the LSTMs, activation functions ( $Act$ ) can be for instance logistic sigmoid or tanh.

$$s_n^{(t)} = g_{4n}^{(t-1)} s_n^{(t-1)} + (1 - g_{4n}^{(t-1)}) Act\left(\sum_j W_{1n,j} i_j^{(t-1)} + \sum_j W_{2n,j} g_{5j}^{(t-1)} s_j^{(t-1)} + bias_n\right) \quad (21)$$

GRUs also have their variations, including Bidirectional Gated Recurrent Unit (Bi-GRU) and Minimal Gated Recurrent Unit (M-GRU) [130], where the reset gate is removed from the model leaving only the update gate. Fergadis *et al.* [40], developed a pipeline with two Bi-GRUs with attention layers, acting as sentence encoder and document encoder, and a final fully connected layer to make the binary classification.



### 2.3.8 ATTENTION MECHANISM

Attention mechanisms/attention layers are a recent approach developed by Bahdanau *et al.* in 2014 [131]. Firstly introduced within an encoder-decoder model for neural machine translation, this mechanism relies on the computation of weights that determine how much attention is given to certain inputs when producing outputs. By reducing the long distance dependencies, this mechanism is capable of reducing the impact of the short-term memory limitation of RNNs [132]. Yet, RNNs and attention layers are in many cases combined in a same model.

Later on, this mechanism would be successfully applied in several other tasks, such as computer vision, text summarization, information extraction, or document classification. [133].

Commonly in document classification, this mechanism is applied to capture the most important words/sentences that are used to produce a sentence and a document vector, respectively. For those cases, the common implementation uses the following equations (22), which in this particular case are used to produce a sentence vector  $sent_i$  (Equation (22c)) [8]. In this regard, the authors used a word annotation  $h_{i,t}$  that goes through an one-layer MLP, which returns a hidden representation  $u_{i,t}$  of this same word annotation. This hidden representation is then used in Equation (22b), where the authors manage to get the importance weight  $\alpha_{i,t}$  by comparing the similarity between the the word representation  $u_{i,t}$  and a context vector  $u_w$ , a fixed-size vector that is updated throughout the model training process. Finally, this computed similarity is normalized with a softmax function. The final sentence vector is then computed with a sum of all word annotations weighted by the resultant attention weights [8].

$$u_{i,t} = \tanh(W_w h_{i,t} + b_w) \quad (22a)$$

$$\alpha_{i,t} = \frac{\exp(u_{i,t}^\top u_w)}{\sum_t \exp(u_{i,t}^\top u_w)} \quad (22b)$$

$$sent_i = \sum_t \alpha_{i,t} h_{i,t} \quad (22c)$$

One advantage of attention mechanisms is their interpretability. For instance, Yang *et al.* [8] discovered that words as *delicious* and *amazing* had a huge impact when predicting the sentiment of a review from the Yelp 2013 dataset.

### 2.3.9 BERT

Over the years, other attention mechanisms appeared, such as the dot product attention, the scaled dot product attention, self-attention and multi-head self-attention [132, 134]. Together with these new attention mechanisms, a new model named Transformer emerged [134]. Once it does not use any recurrent models, transformers allow the parallelization of the training process, making it more efficient when compared to those recurrent models.

Transformers [134] follow an encoder-decoder structure, which was mainly developed to perform neural machine translation. Here, we will be only briefly explaining the encoder structure since it is the one that we will be using in our work. The encoder is composed of 4 main parts [132, 134]:

- **Positional Embeddings:** used to introduce the order information into the model. Since transformers do not use any recurrent or convolutional models, the authors felt the need to introduce this information through positional embeddings. The positional embeddings are used before the stack of encoders. Basically, each positional embedding consists in a vector with  $d_{model}$  dimensions that captures the words positions in a certain sentence. To do that, the authors used two Equations (23), with  $pos$  representing the words position and  $i$  the encoding dimension.

$$PE(pos, 2i) = \sin(pos/10000^{2i/d_{model}}) \quad (23a)$$

$$PE(pos, 2i + 1) = \cos(pos/10000^{2i/d_{model}}) \quad (23b)$$

- **Multi-head self-attention:** based on the scaled dot product attention (Equation (24)), which uses 3 matrices: one for queries  $Q$ , one for keys  $K$  and one for values  $V$ , with  $d_k$  representing the keys dimensions. This mechanism can, in this way, get knowledge, for instance if a word is a verb or to which subject the word "it" refers. The multi-head nomenclature results from the projection of these matrices in a certain number of times  $n$ . Each matrix goes later through an activation function, with its results being concatenated and projected another time to produce the final values (Equation (25) -  $W_i^Q, W_i^K, W_i^V, W^O$  are all parameters learned during the model training process). Lastly, the self-attention part results from the fact that each sentence pays attention to itself. With all this, the

encoder is able to go through all the positions in a sentence and give focus to different words.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (24)$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \quad (25a)$$

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_n)W^O \quad (25b)$$

- **Layer norm and residuals:** residuals are used to add the input vectors of a certain layer to the resultant output vectors of that same layer. After joining them, those grouped vectors are introduced in the normalization layer. Layer norm is used to normalize the outputs of either the multi-head self-attention layer or the feed-forward layer by using  $LayerNorm(x + Layer(x))$ .
- **Feed-forward network:** fully connected feed-forward network applied to each word input independently and is given by:

$$FFN = \max(0, xW_1 + b_1)W_2 + b_2 \quad (26)$$

Transformers use a stack of  $N_1$  transformer blocks/layers as the encoder and  $N_2$  layers as the decoder. In the case of Vaswani *et al.*, both  $N_1$  and  $N_2$  were set to 6. However, this number can vary as in the case of Bidirectional Encoder Representations from Transformers ([BERT](#)). [BERT](#) [52] consists on the use of the encoder part of transformers and it is available in two main versions: [BERT-BASE](#) and [BERT-LARGE](#) with 12 and 24 transformer blocks respectively. Moreover, [BERT-BASE](#) contains 12 self-attention heads and a hidden-size of 768, giving a total of 110M parameters, approximately, while [BERT-LARGE](#) contains 16 self-attention heads and a hidden-size of 1024, which gives around 340M parameters.

As briefly mentioned in subsection 2.1.5, [BERT](#) can be used to produce contextualized word representations. Also, due to the possibility of pre-training this model with unlabeled text, one can simply use it with one output layer on top to make predictions. Additionally, and by introducing bidirectionality in language models through a masked language model, the

authors were able to overcome different previously developed language models with either unidirectionality (e.g. OpenAI GPT) or independent bidirectionality. Basically, the **MLM** pre-training consists in randomly converting words to a defined mask, which will be used to predict those same initial words by using the words around them. Apart from the **MLM** pre-training, **BERT** also offers a task for next sentence prediction, which mainly predicts if a sentence comes after another.

## 2.4 TRANSFER LEARNING

Transfer learning [135] is a technique commonly used in **DL**, with a vast number of approaches implemented for **NLP**. This technique consists on the transfer of knowledge acquired in a certain task and domain by a certain model (named source model), to another model (named target model), which also has an associated task and domain. Additionally, transfer learning offers the possibility to use knowledge regardless of both source and target models having the same task and/or domain.

For document classification, pre-trained embeddings are widely used within **DL** models. These pre-trained embeddings consist of word representations learned by another model (knowledge) that are posteriorly transferred to the target model. This technique is an example of a commonly used type of transfer learning called feature-representation transfer.

With **BERT** models emerging, other versions pre-trained on specific domains appeared, as in the case of BioBERT and SciBERT, which were pre-trained with biomedical literature. These models can be used in 3 different ways:

1. Use of the entire model structure, but training it from scratch (no transfer learning involved).
2. Use of the pre-trained version within the target model and freeze the source weights when training (feature-representation transfer).
3. Use of a pre-trained version within the target model and fine-tune its weights throughout the training of the target model (parameter transfer).

One important advantage of transfer learning is that it allows knowledge acquired in large datasets to be applied on tasks with smaller datasets. This reduces significantly the time

needed for the training process of those models while incorporates a more complete knowledge, frequently resulting in improved performances [135].

## 2.5 HYPERPARAMETER OPTIMIZATION

Hyperparameters are defined as all the ML models' parameters that are not adjusted during the training process (e.g. dropout rate, learning rate, optimizer, layer units, etc.). Since these parameters are static and have an high impact on models, it is of extreme importance to tune them in advance. This will consequently result in a improved adjustment of the ML model to the specific problem used, which will accordingly enhance the obtained results. Due to the vast number of existing hyperparameters and the different possible values that they can take, the process of hyperparameter optimization should always be performed automatically. If performed manually, this process can easily become overwhelming to a researcher and subsequently result in sub-optimal models [7].

Currently, there are many distinct algorithms available for the community, which implement the hyperparameter optimization process by following different approaches [136]:

- **Grid Search:** as its name suggests, it creates a grid with the provided hyperparameters. Then, it iterates through each point on that grid to train a model and consequently evaluate it. Because it trains models independently, this algorithm can be easily parallelized. Nevertheless, it is recommended to use this algorithm with only few hyperparameters, once the number of hyperparameters increase exponentially its computational cost.
- **Random Search:** variation of grid search, where instead of iterating over each point on the grid, it generates random points within search space. Therefore, this model needs also a parameter (commonly called maximum number of trials) to stop the generation of new points. In most cases, random search is able to return better results when compared to grid search. As well as grid search, this algorithm can also be easily parallelized.
- **Bayesian Optimization:** it focuses on the minimization of the maximum number of trials needed to find a global optimum. For that, it attempts to find the best trade-off between searching for new information (called exploration) with the use of the already collected data (called exploitation). The objective of minimizing the maximum number of trials removes

the need for a user to previously set this parameter. This fact is one of the reasons why Bayesian optimization outperforms both random and grid search, besides being also more efficient. However, vanilla Bayesian Optimization cannot be parallelized due to its sequential process (i.e. one iteration is based on a previous iteration). Also, this algorithm is highly dependent on computational resources, which makes its application in DL models a really time-consuming process.

- **Hyperband:** based on random search for the generation of combinations of hyperparameters, complemented with both early stopping and a better resource allocation to improve efficiency. This algorithm starts by training different combinations of hyperparameters (the number of combinations can be set by the user) that are evaluated after a previously set budget (e.g. number of iterations). After the evaluation, the algorithm discards some of the worst models. The number of discarded models after the evaluation are also set by the user. This process occurs until the algorithm reaches a maximum number of epochs, which once again, should be set by the user.

For shallow ML models, grid and random search are two commonly used algorithms. However, since DL models have habitually a lot more hyperparameters, grid search is often not used. Instead, researchers tend to choose one of the other 3 algorithms (Random Search, Bayesian optimization or Hyperband).

To evaluate models on a certain combination of hyperparameters, a researcher commonly use either a validation set or cross-validation. Cross-validation tends to give more realistic scores, although at the expense of taking more time to compute.

## 2.6 RELATED WORK

Over the years, distinct approaches have been made towards the Document Classification task. Comparing articles that used traditional ML algorithms for that purpose (Table 4), it is possible to observe a prevalence of the SVM algorithm over the remaining algorithms (e.g. Logistic Regression (LR), Random Forest, Naïve Bayes (NB)). Regarding the Feature extraction process, all articles used term weighting with an high frequency of the TF-IDF method.

Table 4 – Different ML approaches used to perform Document Classification tasks.  
(prec - precision; rec - recall; acc - accuracy)

Author(s)	Feature Extraction	Models	Evaluation Metrics
Chen <i>et al.</i> [137]	Term lists, BOW, TF-IDF	LR, SVM, Random Forest	F1, prec, AUC
Seymour <i>et al.</i> [138]	TF-IDF, Information Gain	SVM, NB, Hierarchical SVM	AUC, acc
Genkin <i>et al.</i> [139]	TF-IDF	SVM, LR	F1
Kim <i>et al.</i> [140]	Term Weighting	NB	F1
Chen <i>et al.</i> [141]	TF, TF-IDF, TF-IGM	KNN, SVM	F1

Recently, DL alternatives have been applied for the same task of Document Classification (Table 5). However, in these cases, all articles relied on Word Embeddings for feature extraction, instead of term weighting. Furthermore, several articles coupled Word Embeddings with other feature extraction methods, such as TF-IDF, NER and Part-of-speech tagging (POS). For instance, Luo, L. *et al.* [70] obtained improvements in some models when using NER and POS.

For the Deep Learning architectures, models as CNNs, RNNs, LSTMs, CNNs, and their variations (e.g. Bi-LSTM, Bi-GRU) are some of the models used in these articles. Moreover, several articles implemented attention layers and a Hierarchical architecture. Hierarchical architectures allow stacking of DL models, each one being applied in different text levels. For instance, Yang [8] implemented Hierarchical Attention Networks using two Bi-GRUs as word and sentence encoders. Inside each encoder, an attention layer was applied to select the most important words and sentences in the word encoder and sentence encoder, respectively. This model obtained improvements in distinct data sets when comparing to CNNs, LSTMs and traditional ML models. Recently, an ensemble named Random Multimodel Deep Learning (RMDL) was created. This ensemble, which couples a CNN, a DNN and a RNN, attempts to solve the challenging task of choosing an architecture by turning the task into a learning parameter of the model. Kowsari *et al.* [142] achieved improvements over DL and ML models using the RMDL ensemble in the Web of Science data set.

Table 5 – Different DL approaches used to perform Document Classification tasks.  
(prec - precision; rec - recall; acc - accuracy)

Author(s)	Feature Extraction	Architectures	Attention Layer	Evaluation Metrics
Fergadis <i>et al.</i> [40]	Word2vec	Hierarchical B-RNN, B-GRU	•	prec, rec, F1
Dollah <i>et al.</i> [143]	POS, NER, Word2vec	CNN		acc, prec, rec, F1
Burns <i>et al.</i> [9]	GloVe, fasText	CNN, LSTM, CNN B-LSTM	•	acc
Yan <i>et al.</i> [108]	Document word sequence embedding	Boltzmann CNN, CNN+DNB		F1
Abdulkadhar <i>et al.</i> [109]	Word2vec	RCNN		prec, rec, F1
Luo <i>et al.</i> [70]	fastText, NER, POS	LSTM, CNN, LSTM-CNN, RNN, HieLSTM, Ensemble	•	prec, rec, F1
Yang <i>et al.</i> [8]	Word2vec	Hierarchical Attention Networks	•	acc
Kowsari <i>et al.</i> [142]	TF-IDF, GloVe	RMDL (ensemble of CNN, DNN and RNN)		acc
Kowsari <i>et al.</i> [144]	TF-IDF, GloVe	Hierarchical DNN		acc



---

## DEVELOPMENT

---

In this section, the the overall structure of the developed package (BioTMPy - Biomedical Text Mining with Python) will be explained. The package can be found on the following link: <https://gitlab.bio.di.uminho.pt/biotextminingpy/biotmpy> together with the relevant documentation. The package was developed using the Python programming language (version 3.8), which was chosen for its access to many different available packages that allow an easy implementation of deep learning models, being Keras and Tensorflow the two main ones used in this work.

### 3.1 PYTHON PACKAGES

BioTMPy was developed using several free open-source Python packages, which are frequently used when applying Machine Learning models on NLP. The main ones used in this work were:

- **NLTK [145]**: package developed to deal with textual data, providing methods to perform tokenization, stemming, parsing and semantic reasoning;
- **Numpy [146]**: provides efficient methods to create and process multi-dimensional arrays using central processing units (CPUs);
- **Pandas [147]**: developed to manage labeled arrays in two main types of data structures: `Series` and `DataFrame`. Additionally, it offers functions to load data in different file formats as `.csv`, and to transform `DataFrames` into for instance, HTML code;
- **Scikit-learn [148]**: the main objective of this package is to provide methods to easily and efficiently implement ML models for supervised and unsupervised tasks. Auxiliary

functions to perform feature selection, normalization, cross-validation, grid/random search, and data split are some of the other options available in this package.

- **Matplotlib [149]**: used to generate graphs that allow data visualization. It provides various options as barplots, heatmaps, boxplots, scatter plots, etc..
- **Keras/Tensorflow [7]**: a framework that allows an easy implementation of several Deep Learning models, complemented with several functions running over those models, allowing to train, save, load, make predictions, among others. Furthermore, it offers the possibility to run these models on both CPUs and Graphics Processing Unit (GPU)s. Commonly, this package is used with Tensorflow as backend, which allows Keras to deal with tensors in an efficient way.
- **HuggingFace's Transformers [150]**: facilitates the use of pre-trained transformers, containing also interoperability for both PyTorch and Tensorflow frameworks.
- **Flask [151]**: facilitates web development for Python. Some of the main features of this package are URL routing, template rendering, session management, and HTTP request parsing.

### 3.2 BIOTMPY - PACKAGE DESCRIPTION

The main aim of this package is to allow an easy implementation of complex deep learning models and some shallow machine learning models to perform document classification on biomedical literature. Moreover, this package offers a module that allows the deployment of the implemented machine learning models on the form of a web service to rank biomedical documents. The main users of this package would be people that are either starting to contact with the world of NLP and/or Machine/Deep Learning, or people that have some scientific literature that they would like to rank by relevance using an easy-to-use pipeline.

The package is divided in the following modules: *wrappers*, *data\_structures*, *preprocessing*, *mlearning*, *pipelines* and *web* (Fig. 12).

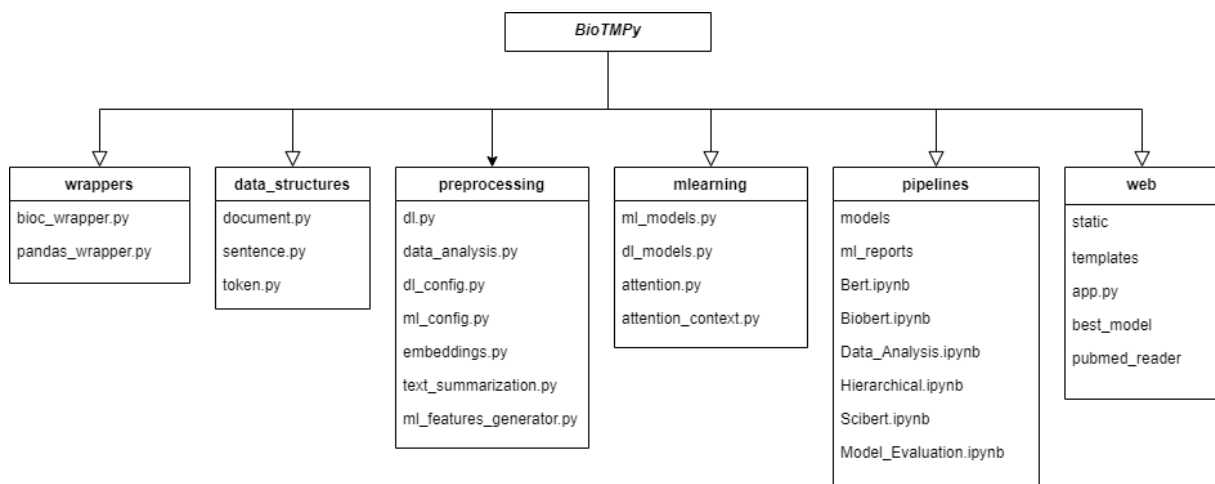


Fig. 12 – Diagram of the overall package structure developed (BioTMPy).

### 3.3 WRAPPERS AND DATA STRUCTURES

The wrappers module provides a set of alternatives for converting different input files into `Document` objects. Thus, it includes a set of different wrappers for distinct input file formats, namely: a *bioc\_wrapper*, a *csv\_wrapper*, a *dictionary\_wrapper* and a *pandas\_wrapper*. *bioc\_wrapper*, *csv\_wrapper* and *dictionary\_wrapper*, which are responsible for converting different input files (bioc.xml, .csv and dictionary.txt formats, respectively (Fig 13)). For that, these wrappers start by splitting text into sentences with the help of the `sent_tokenize` function from the `NLTK` package. Those sentences are then split into tokens using the `word_tokenize` function, again from the `NLTK` package. Then, each sentence and token is converted into different instances that are later saved inside a `Document` object. Furthermore, this `Document` object can save attributes such as document id and title/abstract/full\_text on both string format and list of token instances.

While creating the `Document` objects, the user can also select different options to process his text:

- `lower`: set to `True` to remove cased words;
- `stop_words`: *set* of words (such as 'a', 'the', 'of', 'how', etc.) to remove from text;
- `remove_punctuation`: set to `True` to remove punctuation;

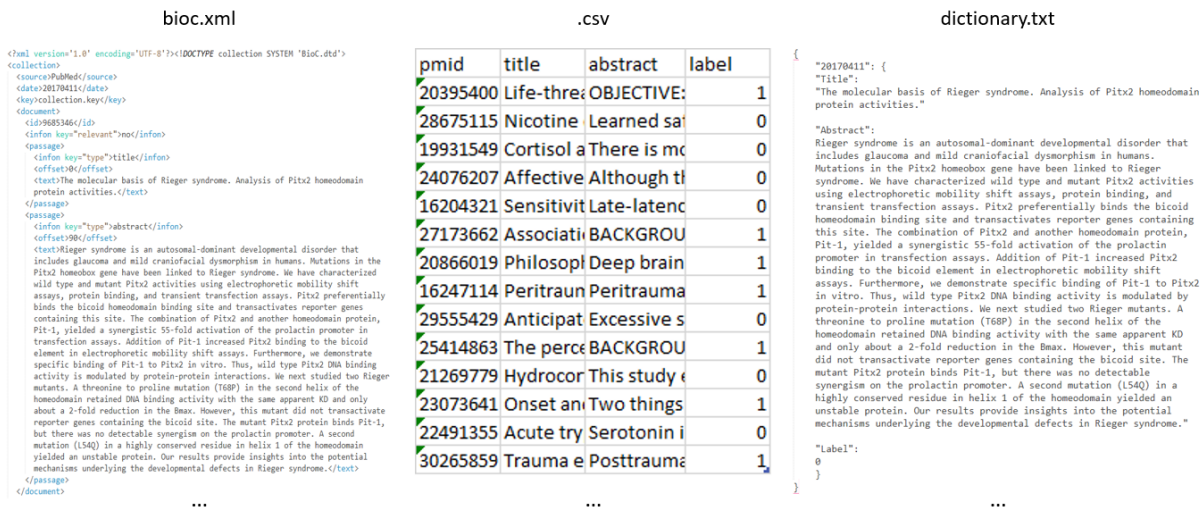


Fig. 13 – Different examples of input files accepted by the distinct developed wrappers.

- **split\_by\_hyphen:** set to True to split compound words by hyphen (e.g. "membrane-bound" is split into 2 different words: "membrane" and "bound");
- **stems:** set to True to perform stemming on words (e.g. "activities" becomes "activ"). For stemming, the package uses the PorterStemmer function from the NLTK package;
- **lemmatization:** set to True to perform lemmatization on words (e.g. "activities" become "activity"). For lemmatization, WordNetLemmatizer from the NLTK package was the one chosen for this work.

Afterwards, these Document objects can be converted into a dataframe with the *pan-das.wrapper*, which relies on the pandas package. This will result in a dataframe with 2 columns: one column with the Document objects, and another column with the associated relevance (Table 14). The relevance can be used to create an instance of the class Relevance from our data\_structures module. Moreover, this class can save the following 4 attributes: label, document id, confidence score, which can be saved when a label is predicted, and a description that indicates the topic of the dataset used on the model training process.

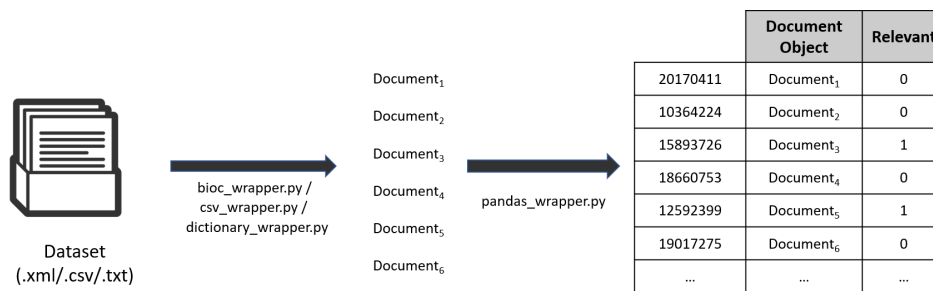


Fig. 14 – Diagram of the developed pipeline with wrappers and data structures.

### 3.4 PREPROCESSING

The preprocessing module has the main objective of converting the dataframe produced on the previous wrappers module into input formats for both shallow machine learning and deep learning models (Fig. 15). In this way, we can generate the numerical inputs that these models require to work.

For the shallow machine learning models, this module allows one to generate features on two levels: sentence level and token level. For the sentence level, it generates some statistics like sentence size (for title, abstract and full text) and number of matches between words in a pre-defined terms list against summarized text. For the text summarization method, we adapted code from the following repository: <https://gist.github.com/edubey/cc41dbdec508a051675daf8e8bba62c5>. Regarding the word level, different features can be produced too, such as the number of tokens, number of distinct words after performing stemming, number of stop words, POS tagging, token size, NER, etc. Furthermore, it is also possible to produce features by computing TF-IDF on all documents. The creation of all these features will then result in a final dataframe that can be used on distinct machine learning models.

For the deep learning models, 3 types of inputs can be created: one for common deep learning models (e.g. CNN, LSTM, GRU), one for hierarchical attention networks and one for BERT models. For the common deep learning models, and by using the tokenizer selected by the user to convert words into indexes, a 2D array can be created. This 2D array is composed of documents on the vertical axis, and words on the horizontal axis. Since all these inputs should have the same length, 3 parameters need to be selected in order to perform the conversion: maximum number of words, padding and truncating. The maximum number of words will

determine the length of the horizontal axis, while padding and truncating will be used to equalize the length of all documents. While padding is used to fill documents containing less than the set maximum number of words with the value 0, truncating is used for the opposite (i.e. cutting documents with more than the set maximum number of words). When performing either padding or truncating, it is necessary to choose where these methods will operate, having for that two possible options: 'pre' and 'post', meaning that these methods will either act at the start or at the end of the document text, respectively.

Regarding Hierarchical Attention Networks, one additional parameter is needed: maximum number of sentences. With this parameter, instead of a 2D array, our module will create a 3D array as output. This 3D array is composed of a dimension with documents, another with sentences and another with words. In this case, the maximum number of words is set for each individual sentence, instead of the maximum number of words per document. This means that, for each document, a 2D array will be used as input for the model.

Finally, for Bert models, a list is created containing 3 arrays with 2 dimensions each, being these arrays equal to the first example (documents  $\times$  maximum number of words). The difference in this case - when compared to the first one - is that 2 more arrays are created: one array to distinguish if the index on the first array is a word index or a padding value (i.e. '1' for a word index and '0' for a padding value) and another array to assign each word index to the associated sentence (i.e. '0' if it belongs to the first sentence and '1' if it belongs to the second sentence) (Fig 15). In the case of Bert models, before creating these arrays, two special tokens need to be added to text: "[CLS]" and "[SEP]" tokens. While the "[CLS]" token is always added at the start of the text, the "[SEP]" token depends on the choice of the user regarding the number of sentences to be used. If the user chooses to use only one sentence, the "[SEP]" token is added at the end of the text. However, if the user chooses to use two sentences, the "[SEP]" token will be added not only at the end of text, but also between the title and the abstract. This means that, in this way, the title will be considered as one sentence and the abstract as another sentence.

In addition, and in order to use pre-trained word embeddings, this module offers functions to convert files containing word vectors into an embedding matrix, which can be later used inside deep learning models. For that, the user has to pass 5 parameters: embedding path, word index from the trained tokenizer, embedding dimension, maximum number of words and embedding format - which can be either GloVe, Word2Vec or fastText formats. For both Word2Vec and

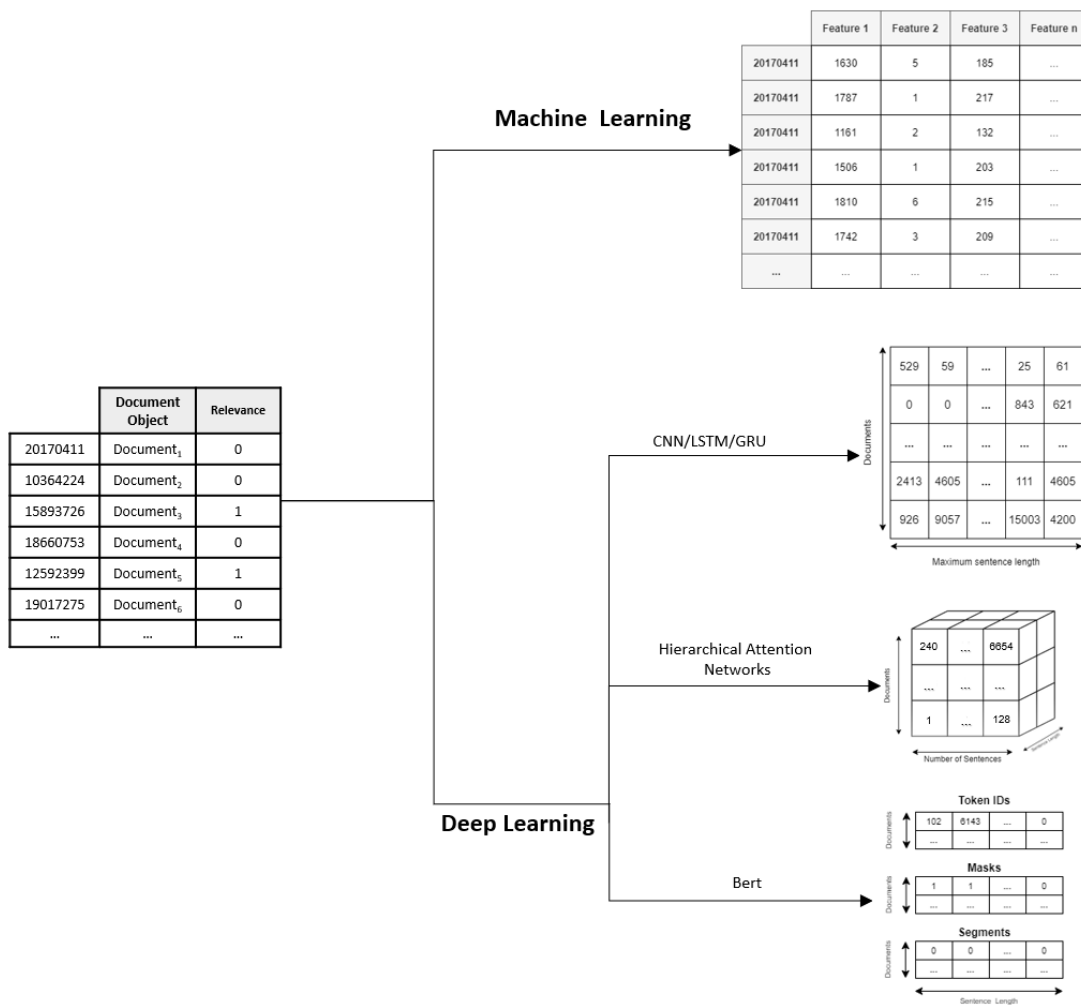


Fig. 15 – Diagram demonstrating the different types of inputs that can be produced with the preprocessing pipeline.

fastText models, the function `load_word2vec_format` from the `gensim` package is used to load the file.

All the parameters such as the ones used for preprocessing text, the ones used to produce inputs for the models, as well as instances like tokenizers, can be saved as attributes of a Config object (`MLConfig` for shallow machine learning models and `DLConfig` for deep learning models). Those Configs can later be saved in order to not only evaluate the model afterwards, but to also ease the process of replicating a certain model. In addition, these Configs have functions for both saving inputs/features and writing all the results on a `.txt` file, in case of shallow machine learning, or a `.csv` file, when using deep learning models.

Lastly, this module also provides functions to perform data analysis on the dataset. This allows not only a comparison between train and test sets, but also between their labels. The possible comparisons implemented are: top number of words (for every existent n-grams), word count, number of sentences per document, number of words per sentence, balance of labels, and Latent Semantic Analysis. It is also possible to perform t-SNE and UMAP using two different feature extractors: TF-IDF and bag-of-words. For these plots, 3 main packages were used: Yellowbrick, scikit-learn and matplotlib. Regarding pre-trained embeddings, it is also possible to visualize the location of the closest word vectors to a chosen word vector as reference.

### 3.5 MACHINE LEARNING

The developed Machine Learning module is responsible for providing distinct models for both shallow machine learning and deep learning.

On the case of shallow machine learning, the module also provides functions to train, make predictions and evaluate the model. The available models can be seen on Table 6. For the implementation of these models, the scikit-learn package was used, since it offers an easy integration of those models.

Regarding Deep Learning models (Table 6), the BioTMPy package uses the Keras package with the Tensorflow backend for the majority of the models, with exception of transformers, where it uses additionally the Transformers library. All the Deep Learning models were implemented using the Functional API from the Keras package, allowing the use of several Keras functions, such as fit, evaluate, predict, save, load, etc..

Table 6 – Distinct available models for both shallow Machine Learning and Deep Learning

Machine Learning	Deep Learning
- SVM	- DNN / CNN / LSTM / GRU
- Naïve Bayes	- CNN-BiLSTM
- Random Forest	- Hierarchical Attention Networks
- Nearest Neighbors	- Bert / BertForSequenceClassification
- Logistic Regression	- SciBert
- Decision Tree	- BioBert



Concerning the implementation of Hierarchical Attention Networks, two attention layers were created: `AttentionLayer` and `AttentionWithContext`. These layers were adapted respectively from two different sources<sup>1,2</sup>.

It is also important to mention that for `SciBert`<sup>3</sup> and `BioBert`<sup>4</sup> models, `BioTMPy` package uses the `transformers-cli` function from the `Transformers` library that can convert a `Tensorflow` checkpoint to a `Pytorch` file. This `Pytorch` file, coupled with the `config.txt` file from `Bert`, can then be used as a custom layer in the Deep Learning model. This layer is created using the `TFBertModel` from `Transformers` and it can be later used to perform fine-tuning (an optional step that can be performed by setting the hyperparameter `static_bert` to `True`). Fine-tuning is optional since the `BioTMPy` package offers different versions of models containing `BERT` with and without the pre-trained weights being frozen. Additionally, this package has also implemented a specific model of `BERT` (also from the `Transformers` library) named `TFBertForSequenceClassification`, which is basically a `Bert` model with a linear layer on top of the pooled output.

### 3.6 PIPELINES

On this module, one can access different Jupyter notebook files, which contain examples for the implementation of all types of models implemented in our package. The main objective of this module is to give intuitive and step-by-step examples on how to implement pipelines that are can be used to classify text documents. For that, these pipelines make use of all the previously mentioned modules: `wrappers`, `data_structures`, `preprocessing` and `mlearning`.

All of these Jupyter notebooks follow a similar structure with some variations depending on the model used. Also, in these pipelines, it is possible to divide the available data into training and validation sets by giving the desired percentage for the validation set. This allows a posterior observation of the training history on the training set and scores obtained on the validation set. Also, it allows to perform the consequent plots of the resulting scores obtained on those sets after the training process. Regarding the test set, it is also possible to plot ROC and Precision-Recall Curves, after performing predictions.

1 <https://gist.github.com/cbaziotis/7ef97ccf71cbc14366835198c09809d2>

2 [https://humboldt-wi.github.io/blog/research/information\\_systems\\_1819/group5\\_han/](https://humboldt-wi.github.io/blog/research/information_systems_1819/group5_han/)

3 <https://github.com/allenai/scibert>

4 <https://github.com/dmis-lab/biobert>

Each run of a model pipeline can be saved within the `models` folder, which will contain a `.xlsx` file named `results_model_name`. This file will then contain all the hyperparameters used on the model and the final results obtained on both training and test sets. In the same folder, other files can also be saved to ease the replication process. More precisely, a user can save a file containing weights, a config file and a `yaml` file describing the model's structure.

Additionally, all the results achieved during this work can be easily reproduced by using the same hyperparameters and seed values. This is possible since the developed pipelines contain a seed option that can be set on the distinct functions that rely on random values. For instance, when using the preprocessing module, the seed value assures that a user can reuse the same split of training and validation sets by removing the randomness during data splitting. This will consequently allow a better comparison of several hyperparameters and/or algorithms.

Regarding the models, setting a seed value provides a control over their weight initialization by recreating the same initial weights for each training process. Also, the seed is capable of controlling the behaviour of layers that rely on the use of random values (e.g. dropout layers). However, it is worth to mention that this same control cannot be applied when using a the pre-trained `BERT` models from the `Transformers` package, since this package does not offer an option to control the randomness present on the layers of those models (e.g. dropout layers), For that reason, models containing these custom layers may present some variation associated, even with a fixed seed value.

Reducing the training time is also a factor of importance accounted during the creation of these pipelines. In fact, an option to choose multiple `GPUs` to accelerate the training process was added, which is available with the `MirroredStrategy` function from `Tensorflow`. Thus, this function offers the possibility to use multiple `GPUs` that will reduce considerably the training time of complex Deep Learning models.

Finally, these pipelines also include modules to optimize and evaluate different `ML` algorithms. For the optimization process, one can easily select different optimization algorithms (Grid Search/Random Search for shallow `ML` and Random Search/Hyperband for `DL`). In addition, the user can also evaluate the different models by performing cross-validation. Some examples on how to implement hyperparameter optimization or cross-validation can be found on files containing either `hp` or `cv` within their names, respectively (e.g. `cv_bert.py` and `hp_bert.py`). For the hyperparameter optimization, an additional file is saved inside the `hp_results` folder containing

the best hyperparameters obtained with the hyperparameter optimization algorithm selected. For cross-validation, a `.xlsx` file is saved inside the `models` folder containing the hyperparameters used for model training and the respective results obtained on the different folds of the cross-validation.

### 3.7 WEB

Lastly, BioTMPy offers the possibility to be deployed as a Web Service using this web module. The main aim of this module is to create an easy-to-use, interactive and freely accessible interface that allows a user to rank biomedical documents with our best model developed in this work. The Web Server can be found on the following link: <https://biotmpyppi.bio.di.uminho.pt>

This module is composed by the following components:

- `app`
- `templates`
- `static`
- `best_model`
- `pubmed_reader`
- `DockerFile`

The `app` file starts by creating a Flask application using the `Flask` package. Then, the app starts by rendering the template `home.html` from the `templates` folder by using the `render_templates` function from `Flask`. In order to style the HTML pages, there is also a `styles.css` file written in CSS language that is accessed on the `static` folder. Also, in order to make the home page interactive, the `static` folder contains two other files written in Javascript, which use the `jQuery` package. With this, the home page is able to offer different options for the user to input text documents, such as PubMed IDs, Search Term and PDF files (Fig 16).

All the inputs are converted into text with functions from the `pubmed_reader.py` file. Regarding PubMed IDs, text data is retrieved using the `effetch` function from the `Biopython` package. This function is able to retrieve both title and abstract, given a PubMed ID and email. In the case of the Search Term, the user has to introduce a search term and the maximum number

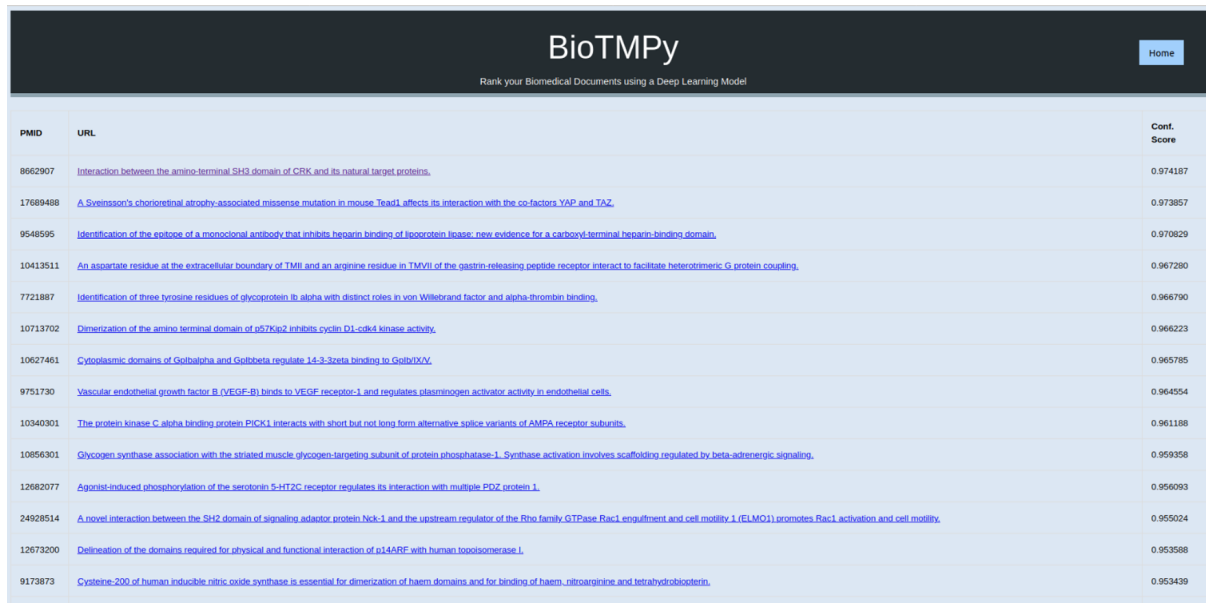
Fig. 16 – Home page of the BioTMPy Web Service. On this example, the input type selected is the PubMed IDs.

of documents to retrieve. Subsequently, these inputs are used on the function `esearch`, equally from the `Biopython` package, which returns PubMed IDs from the search results of the PubMed database. Following that, the retrieved Pubmed IDs are converted into text data by making use of the first function mentioned. Regarding PDF files, BioTMPy starts by using the function `get_title_from_io` from the `pdftitle` package. This function returns, when available, the title found on a PDF file. Consequently, this title is used to search the PubMed ID by using the `esearch` function, which, in turn, is again used on the first function.

All the retrieved data, which resulted from the inserted inputs, is then converted into a dictionary. This dictionary is next converted into `Document` objects using the `dictionary_wrapper`. Thereafter, those `Document` objects go through the necessary preprocessing steps in order to make the final predictions with the best model selected.

In the end, the predicted documents are ranked by the confidence score associated to each prediction, which are returned as a pandas dataframe. However, before ranking, the confidence scores of documents predicted as non-relevant have to be converted to negative values. This way, the returned ranked documents will contain the documents classified as relevant with the highest confidence score at the top of the table and the opposite at the bottom (i.e. non-relevant documents with the highest confidence score associated). Lastly, the final

dataframe is converted to HTML code using the `to_HTML` function from the `pandas` package. Thus, the final result can be easily sent to the `result.html` that is rendered by our app (Fig 17).



PMID	URL	Conf. Score
8662907	<a href="#">Interaction between the amino-terminal SH3 domain of CRK and its natural target proteins.</a>	0.974187
17689498	<a href="#">A Sryensson's choroideremia-associated missense mutation in mouse Tead1 affects its interaction with the co-factors YAP and TAZ.</a>	0.973857
9548595	<a href="#">Identification of the epitope of a monoclonal antibody that inhibits heparin binding of lipoprotein lipase: new evidence for a carboxyl-terminal heparin-binding domain.</a>	0.970829
10413511	<a href="#">An aspartate residue at the extracellular boundary of TMII and an arginine residue in TMVII of the gastrin-releasing peptide receptor interact to facilitate heterotrimeric G protein coupling.</a>	0.967280
7721887	<a href="#">Identification of three tyrosine residues of glycoprotein Ib alpha with distinct roles in von Willebrand factor and alpha-thrombin binding.</a>	0.966790
10713702	<a href="#">Dimerization of the amino terminal domain of p57Kip2 inhibits cyclin D1-cdk4 kinase activity.</a>	0.966223
10627461	<a href="#">Cytoplasmic domains of Golbalpha and Golbbeta regulate 14-3-3zeta binding to GolbXVV.</a>	0.965785
9751730	<a href="#">Vascular endothelial growth factor B (VEGF-B) binds to VEGF receptor-1 and regulates plasminogen activator activity in endothelial cells.</a>	0.964554
10340301	<a href="#">The protein kinase C alpha binding protein PICK1 interacts with short but not long form alternative splice variants of AMPA receptor subunits.</a>	0.961188
10856301	<a href="#">Glycogen synthase association with the striated muscle glycogen-targeting subunit of protein phosphatase-1. Synthesis activation involves scaffolding regulated by beta-adrenergic signalling.</a>	0.959358
12682077	<a href="#">Agonist-induced phosphorylation of the serotonin 5-HT2C receptor regulates its interaction with multiple PDZ protein 1.</a>	0.956093
24928514	<a href="#">A novel interaction between the SH2 domain of signaling adaptor protein Nck-1 and the upstream regulator of the Rho family GTPase Rac1 engulfment and cell motility 1 (ELMO1) promotes Rac1 activation and cell motility.</a>	0.955024
12673200	<a href="#">Delineation of the domains required for physical and functional interaction of p14ARF with human topoisomerase I.</a>	0.953588
9173873	<a href="#">Cysteine-200 of human inducible nitric oxide synthase is essential for dimerization of haem domains and for binding of haem, nitroarginine and tetrahydrobiopterin.</a>	0.953439

Fig. 17 – Results page of the BioTMPy Web Service. This page shows the ranked documents as a table with hyperlinks of the documents on the PubMed database.

Additionally, this module contains a DockerFile that allows the creation of a container for the developed app. This way, we can assure the functionality of the app on practically every machine.

### 3.8 DISCUSSION

The package developed in this work is divided in different modules in order to make a clear division of the different steps needed to perform document classification. This also allows a user to easily adapt code from different modules and use it independently in a different context.

Despite the focus on biomedical documents, this package can also be easily used with documents from topics not related to the biomedical field since it contains, beyond the biomedical specific methods, generic methods such as functions to read `.csv` and `.txt` files, general feature extractors (e.g. POS tagging, [NER](#), [GloVe](#)) and [ML](#) models (e.g SVM, [BERT](#), Hierarchical Attention Networks, etc.).

Additionally, and despite our use of a binary classification problem, our package offers the possibility to be applied on multi-classification problems. Also, we consider that a really positive feature of this package is the given pipeline examples for the implementation of complex DL models, which can be a really overwhelming process for someone with little experienced on those models. Furthermore, this package offers the possibility to make data analysis on text documents that can give important insights to both select hyperparameters and understand the models' results. Finally, it is worth mentioning that the production of different reports of ML models can really facilitate the comparison between models and also their reproducibility, once all their hyperparameters are also reported.

One limitation of this package is the implemented hyperparameter optimization process. Since DL models have a vast number of hyperparameters and structure possibilities (e.g. number of layers, number of dropouts, dropout rate for each dropout layer, etc.), we faced some difficulties when making this process simple to use. Because of that, the actual implementation may force the user to do some manual implementation of the different possibilities that he may want to test. So, in the next steps of this work, it would be interesting to make this process simpler and more intuitive, without losing the complexity involved needed. Also, in the future, it may be interesting to add a configuration file where the user can select all the necessary options for each step (e.g. feature selector, ML model, optimizer, metrics) and allow to just run a pipeline on the terminal with that configuration associated. This would possibly facilitate users that may want to classify documents without going into much detail on the ML models implementation, having only to create two sets of documents for that (one for model training and one with documents to be classified).

This package offers the possibility to implement a web server that allows the classification of documents in a fast and practical way. The web server module is also divided in different folders, which allows a user to easily adapt it for its needs. However, the developed web server has also a limitation regarding the uploading of PDF files. This process starts after the upload of these files by the user, which consequently goes to the web server. There, the web server attempts to find the title of the documents present in those files. However, the package used to find titles on PDF files (`pdftitle`) has some difficulties in doing so in several documents. Hence, it would be also interesting to improve this process in the future, since it is a highly practical way to

classify documents. Moreover, one possible improvement is the expansion of the web server to other scientific databases, rather than using only the PubMed database.

In the future, this package can also be extended to include other [BioTM](#) tasks such as information extraction.

---

## RESULTS AND DISCUSSION

---

### 4.1 DATASET DESCRIPTION AND CHALLENGE OVERVIEW

The first step of this work was the selection of a dataset to perform document classification. For that, we decided to use a dataset from the track 4 (“Mining protein interactions and mutations for precision medicine”) of the Task VI from the BioCreative forum<sup>1</sup>. The topic of Protein-protein Interactions (PPI) had already been approached by the BioCreative forum in previous tasks, more precisely, in task III, where teams had to extract articles with PPI mentions. Understanding PPI has big importance regarding the study of cellular structure and functionality. Because of that, in this most recent track, they decided to couple this knowledge with information about mutations, which can alter the behaviour of those PPI. This knowledge has high relevance for precision medicine, which attempts to improve disease treatment and prevention by adapting the existing approaches for each individual patient.

For this more recent track, BioCreative decided to challenge different text mining teams of the biomedical community to compete in 2 distinct sub-tasks regarding this topic: Document Triage and Relation Extraction. In this work, we chose to only develop the Document Triage Task that consists on a binary classification problem to identify relevant documents about PPI altered by genetic mutations. The available data is composed of a training set and a test set with 4082 and 1464 documents, respectively (Table 7). Those documents are composed of `id`, `title` and `abstract` and were manually curated by curators from the BioGRID database [152]. These datasets are also available in two formats: `json` and `xml`, with the latter being the one chosen for this work.

---

<sup>1</sup> <https://biocreative.bioinformatics.udel.edu/tasks/biocreative-vi/track-4/>



Table 7 – Content of the two Datasets from the track 4 of the Task VI from the BioCreative forum.

Dataset	Documents	Relevant	Non-Relevant
Training	4082	1729 ( $\approx 42,4\%$ )	2353 ( $\approx 57,6\%$ )
Test	1464	730 ( $\approx 49,9\%$ )	734 ( $\approx 50,1\%$ )

It is important to mention that despite the use of this dataset, the challenge itself already ended, resulting in a publication in 2019 with all the results submitted by 10 different teams [152]. The BioCreative team allowed each team to submit up to 3 runs, which resulted in a total of 22 submissions. Additionally, it is noteworthy that we already did a previous work using this same dataset, but applying only shallow machine learning models with a focus on feature engineering (i.e. implementation of distinct methods to extract features from text). In Table 8, it is possible to see the best results submitted on the challenge by the participating BioTM teams, together with the baseline set by the BioCreative team and the best score we got in our previously developed work. Despite managing to get an f1-score relatively close to the best one submitted, we still wanted to improve our results by implementing the most recent Deep Learning models. So, in this work, we will be using our ML model result as baseline for new implementations.

The team that got the best f1-score and precision metrics used a Hierarchical Attention Network with 2 Bi-GRUs - one Bi-GRU as sentence encoder and another as document encoder [40]. An interesting part of this work is the fact that the authors' implementation did not pass the title sentence vector, which results from the sentence encoder, into the document encoder. Instead, the title vector is passed directly from the sentence encoder to the final layer, where it is consequently concatenated with the abstract vector outputted from the document encoder. The main objective behind this decision was to keep the title integrity and representation, since the authors consider that, most of the times, the title has within itself the most important information of the entire text regarding the document relevance.

Table 8 – Best results submitted on the challenge by the different teams and best score from a previously developed Machine Learning model by the author. The values from the submissions marked in bold represent the best result for that specific metric of all teams.

		<b>Avg Prec</b>	<b>Precision</b>	<b>Recall</b>	<b>F1</b>
	421	<b>0.7253</b>	0.6073	0.7997	0.6904
<b>Submissions</b>	418	0.7158	<b>0.6289</b>	0.7656	<b>0.6906</b>
	414	0.5077	0.5022	<b>0.9801</b>	0.6641
<i>Baseline</i>		0.6515	0.6122	0.6435	0.6274
<b>Previous Model</b>	LR	0.6379	0.5736	<b>0.7912</b>	<b>0.6651</b>

## 4.2 PREPROCESSING

All documents were split by sentence and each sentence subsequently split by token. Furthermore, and beyond the string with the full text, it is possible to access the title and abstract strings separately. This happened when trying BERT models with sentence pairs (title was defined as one sentence and abstract as another sentence). In most cases, we also lowered all text, with exception for BioBERT once it was pre-trained on words containing capital letters. Stemming and Lemmatization were not performed in order to keep the integrity of all words. Additionally, all stop words were removed from text using the set of English stop words from the NLTK package. Stop words removal was only not performed in BERT models since we believe that their vast number of attention layers is capable of automatically remove that noise.

Finally, words were also split by hyphen, resulting in two separate words. Once again, this last step was not performed on BERT models, since their tokenizers are also able to divide those same words into sub-words, which may have impact on the model's learning process. More precisely, BERT tokenizers are able to divide a word not present in its vocabulary until it gets sub-words present in the vocabulary used (e.g. ["phosphorylation"] may become ['p', '##hos', '##ph', '##ory', '##lacion']). Contrarily to BERT tokenizers, when using static pre-trained word embeddings, which do not have the capacity to split words into sub-words, we had to set and index to all the unknown tokens found. In our case, this was performed by using the 'OOV' token.

### 4.3 DATA ANALYSIS

To analyse our data, we performed the methods developed on the Data\_Analysis pipeline, which compares different properties of the documents used. This allowed us to have a picture of the composition of our data and to choose some hyperparameters for distinct models (e.g. maximum number of words per sentence for the Hierarchical Attention Networks).

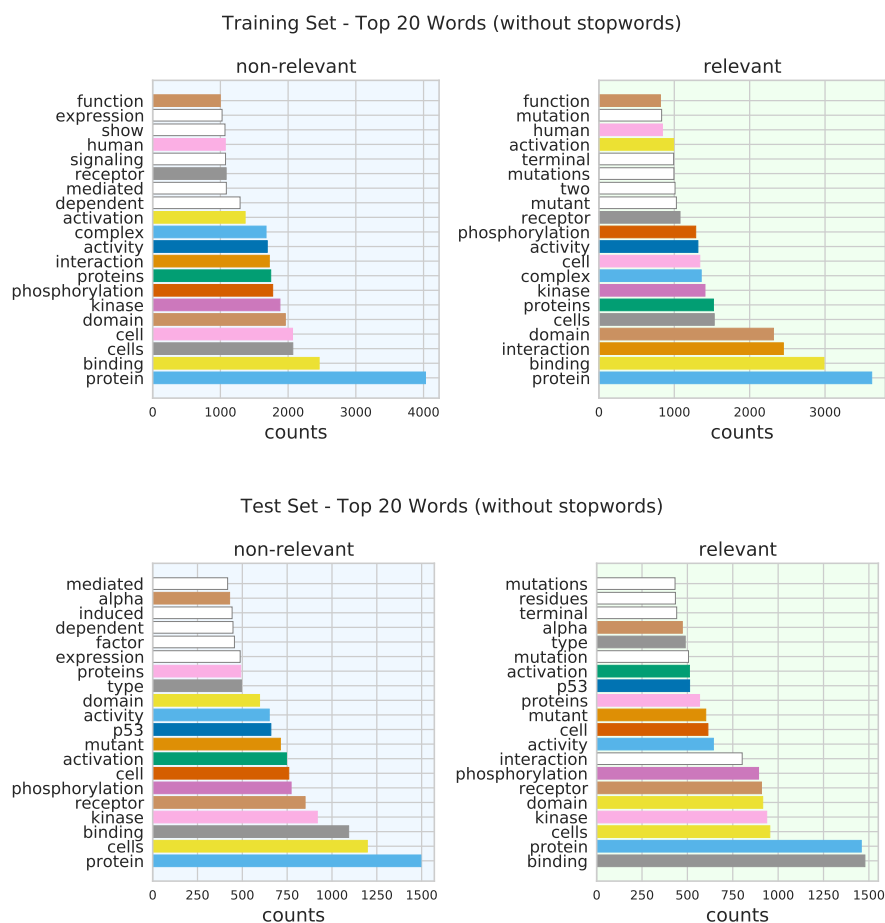


Fig. 18 – Top number of words (without stop words) for the two different labels, for both training and test sets. White bars represent the words that do not match between the two labels on a dataset.

Firstly, the top number of words according to their frequency (after removing stop words), in both relevant and non-relevant documents, considering both training and test datasets, were identified. From the results shown in Fig. 18, it can be concluded that from the 20 words that appear more often, only 5 (25%) differ in the different labels (represented by the white bars on the figure) on both training and test sets. This shows a high similarity between the content of

documents from the different labels, which might indicate that distinguishing these labels might be an hard task for the models.

On the other hand, some words such as "mutation", "interaction" and "mutant" (words directly related to our topic of interest) are only present on the top 20 words of the relevant sets. Moreover, both training and test sets contain many words in common (e.g. "binding", "cells", "protein", "phosphorylation"), which might be an indicator of a similar distribution between the training set and test set. This is important since models should be tested in similar data to which the model was trained on.

Additionally, the same comparison was performed, but now for the top 100 words, and the results are provided on Supplementary Material (Figures 28 and 29). In this comparison, one can observe that only 13% and 20% of the words differ between relevant and irrelevant documents, on the training and test set, respectively, which supports the high similarity between documents from the two labels. Finally, when observing the frequency of the previously mentioned words ("mutation", "interaction" and "mutant") more related to our topic, in this set of top 100 words, we can see that this time they are present on both labels, although with a lower frequency on the non-relevant documents.

Another comparison performed was related to the total number of words per document (title and abstract), considering the two labels (Fig. 19). The results seem to show, as expected that there is no obvious relation between the number of words and the respective label. Of note only the fact that there is a higher frequency of longer documents on the test set when compared to the training set.

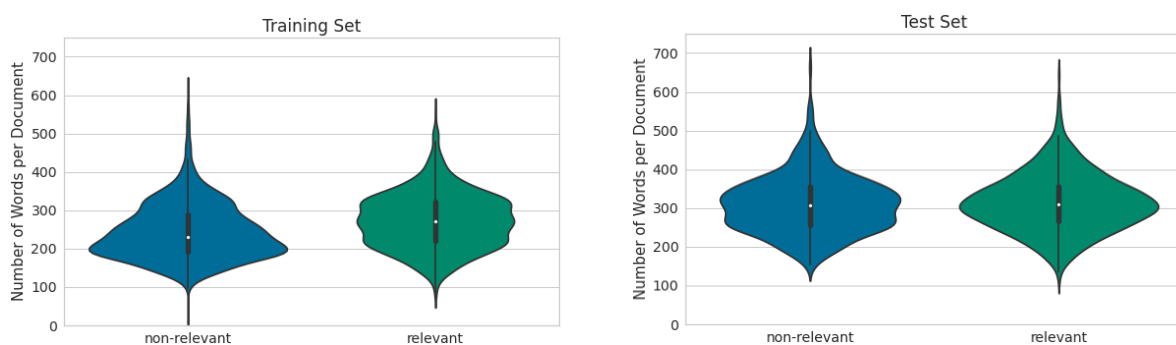


Fig. 19 – Total number of words per document on both training and test set.

Next, the **t-SNE** method (from the `Yellowbrick` package) was performed, using 50 components on both training and test sets (Fig. 20). To perform this analysis, we used features produced by the **TF-IDF** method from the `scikit-learn` library. On both sets, it is possible to notice the formation of some clusters, which are grouping the most similar documents together. However, and at least in these 2 dimensions created, it is not possible to see a clear division of the two labels of interest (non-relevant, relevant). This, once again, is in line with the previous observations. However, and despite the fact that we removed stop words before performing t-SNE, these results may still contain some noise that can be interfering with the analysis.

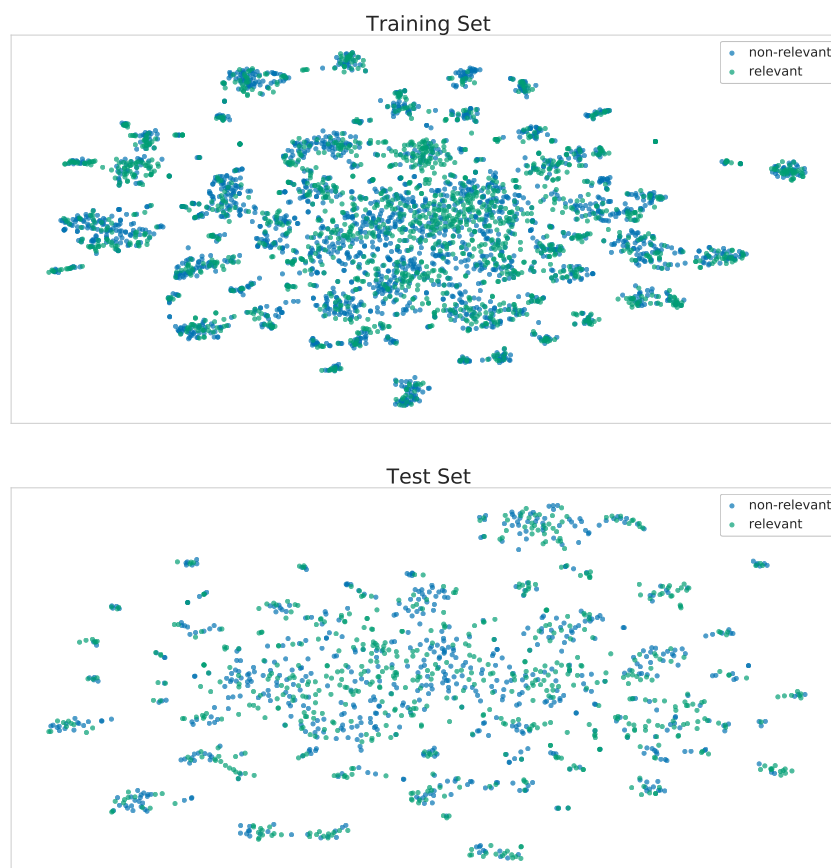


Fig. 20 – **t-SNE** results of both sets using features produced by **TF-IDF**, which were decomposed in 50 components.

## 4.4 DEEP LEARNING RESULTS

### 4.4.1 COMPUTATIONAL RESOURCES

Since complex Deep Learning models are computationally expensive, their training process can be sometimes a really time-consuming operation. Nevertheless, the time spent on both the training process and hyperparameter optimization of these models can be highly reduced by the use of powerful hardware. For that reason, all models were trained on 4 Tesla T4 GPUs, which we managed to do by using the Mirrored Strategy function from the Tensorflow package. These GPUs were launched by NVIDIA in September 2018 and each one possesses 16 GB GDDR6 of memory and 320 tensor cores. These tensor cores were introduced by NVIDIA with the main purpose of improving the GPU performance on the training of ML models.

### 4.4.2 EMBEDDINGS

For the feature extraction process, we decided to firstly compare 4 different pre-trained embeddings: GloVe, [49] BioWordVec [58], PubMed Vectors ("pubmed\_pmc") [55] and word vectors trained on PubMed in 2016 ("pubmed\_ncbi") [59]. With the exception of GloVe, the other 3 embeddings were pre-trained on scientific documents:

- GloVe: pre-trained on text from Wikipedia available in 2014 and Gigaword fifth edition archive. It was pre-trained on uncased text (i.e. lowered text) and provides vectors in 4 different dimensions: 50, 100, 200 and 300.
- BioWordVec: "bio\_embedding\_extrinsic" version published in 2018 recommended for text classification tasks. It is available in a binary format and contains more than 2M lowered words, which are converted into 200-dimensional vectors. These vectors result from words of both PubMed documents and Medical Subject Headings (MeSH). Two interesting parts of the implementation of these embeddings are the use of MeSH words to complement the biomedical literature and the use of a fastText model to incorporate sub-word information for the creation of a more complete vocabulary.

- PubMed\_PMC: word2vec model - more precisely, the skip-gram version - pre-trained on PubMed and PMC documents available in 2013, which resulted in a set of 200-dimensional word vectors.
- PubMed\_NCBI: Similarly to PubMed\_PMC, the authors of these embeddings used the word2vec model to produce word vectors, which are available also in a binary format. However, in this case, the model was trained on PubMed abstracts from 2016. Also, these pre-trained embeddings are only available in vectors of 100 dimensions.

In order to compare these embeddings, the first step was to visualize the 9 most similar vectors to the "protein" vector for each pre-trained embedding. To accomplish that, the cosine similarity was computed between all the vectors present in each pre-trained embeddings against the "protein" vector, selecting at the end the top 9 most similar ones. The results can be seen in Fig. 21. As it was expected, all embeddings were able to group words with very similar meanings.

In the case of BioWordVec, it is also possible to see that it contains many variants of the word "protein" (e.g. proteins, protein8, protein2) highly close the protein vector. This might be a consequence of their focus on the integration of sub-word information into their model. On the contrary, the same does not happen when observing vectors from GloVe, which does not seem to have such a complete vocabulary, with some more distant words appearing close to the protein vector, such as carbohydrates and sodium. This is perhaps due to the use of more general purpose documents by these pre-trained embeddings, instead of the high focus on biomedical documents given by the rest of the pre-trained embeddings tested in this work.

Before deciding which pre-trained embeddings will be used, a comparison of their performance was done. For that, a  $k$ -fold cross validation process, with 10 folds, was run for 2 different Deep Learning models (a CNN-BiLSTM with attention, which was adapted from Burns *et al.* [9], and a Hierarchical Attention Network with 2 Bi-LSTMs). Using a seed value (123123), it was assured that the same folds were used across the different cross validations performed. Also, inside each model, a comparison was performed to assess the impact of splitting words by hyphen. Besides that, all the other parameters were kept the same, in order to only measure the impact of using different embeddings and word splitting. Embedding dimensions were another parameter controlled to make a fair comparison. For this, we chose the pre-trained embeddings'

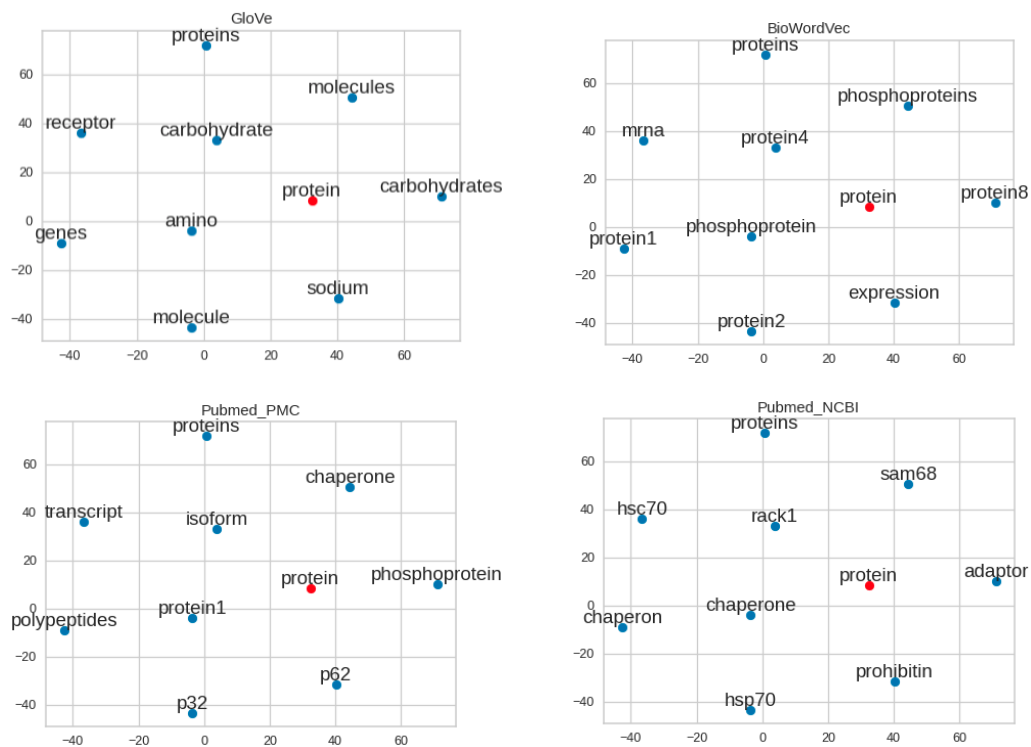


Fig. 21 – t-SNE 2D plots of the closest vectors to the protein vector using different Pre-trained Embeddings.

versions that produce vectors with 200 dimensions, with the exception of "pubmed\_ncbi" which was only available with 100 dimensions.

The results from the performed cross-validations can be seen in Fig. 22. Results were grouped by embedding type and by word splitting. A Wilcoxon test was performed on these results, which allowed to see that only GloVe embeddings showed to be significantly different from the other pre-trained embeddings, presenting a lower mean when compared to the rest. The other 3 types of embeddings (BioWordVec, "pubmed\_pmc" and "pubmed\_ncbi") did not show any differences between each other.

Nevertheless, BioWordVec embeddings were chosen for the rest of the work, since it showed the best mean on the cross validation results over all the tested embeddings. Also, regarding word splitting, no significant differences were observed. However, once again, the choice was to split words since it showed a higher mean when compared to non-split words. Finally, by performing these comparisons, it is easy to observe an overall better performance of the Hierarchical Attention Network over the [CNN-Bi-LSTM](#) model.



As expected, for the reasons mentioned above, GloVe got the worst results of all embeddings. However, that difference was relatively small, possibly indicating that the data used to pre-train these embeddings was quite extensive. Moreover, BioWordVec showed the best results of all of the tested embeddings with an extremely small advantage over "pubmed\_ncbi". This might be explained by two reasons: firstly, since it was the most recently developed approach, it allowed the authors to access and consequently use updated and more complete data, which was also complemented with MeSH information. Secondly, this approach also uses the fastText model that gives a big focus on unknown words through the incorporation of sub-word information. Finally, we were also able to find an almost direct correlation between these results with the number of words of our training set not present in the vocabularies of these embeddings. More precisely, the percentage of unknown words on these embeddings were: 50.96% for GloVe, 10.07% for BioWordVec, 36.61% for "pubmed\_pmc" and 8.85% for "pubmed\_ncbi".

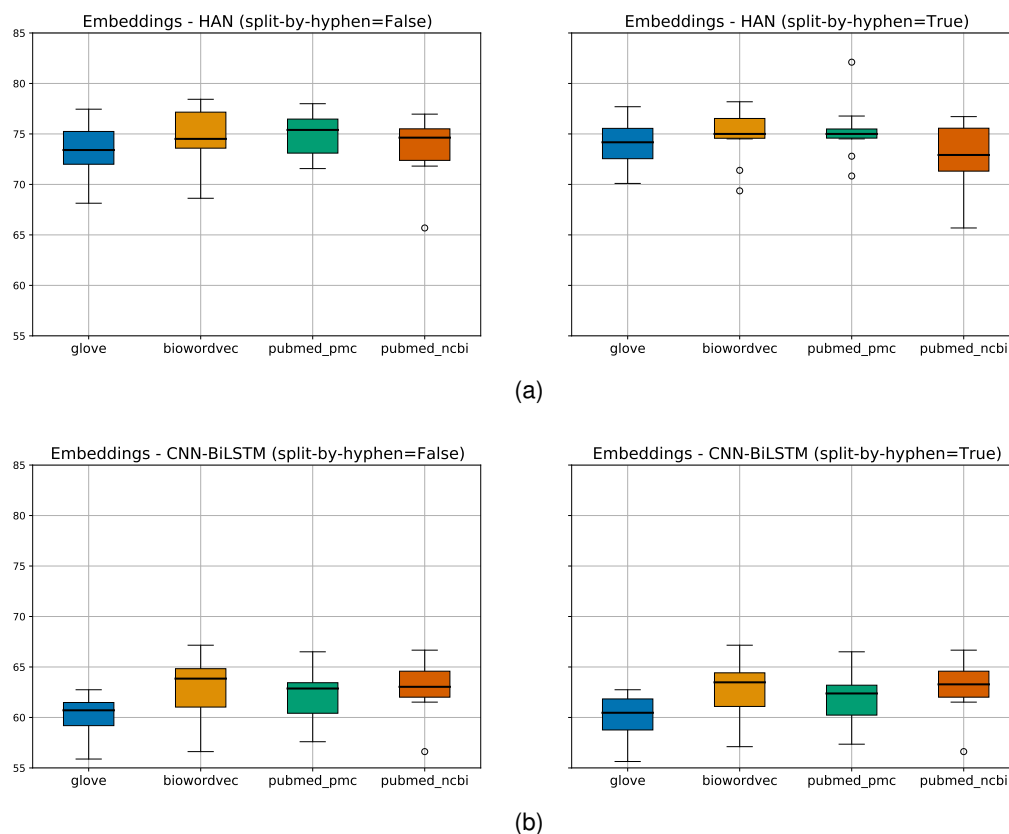


Fig. 22 – Accuracy results from a 10 fold cross validation using different pre-trained Embeddings. (a) Comparison between two different preprocessing options (split and non-split words by hyphen) on a Hierarchical Attention Network (HAN) with two Bi-LSTMs. (b) Same comparison for a CNN-Bi-LSTM model.

#### 4.4.3 HYPERPARAMETER OPTIMIZATION

One important part of Machine Learning is the optimization of the different models' hyperparameters. In the case of Deep Learning, due to its vast number of possibilities, that process can be really overwhelming and also computationally expensive. For that reason, the option was to use the Hyperband tuner from the Keras Tuner package. This tuner optimizes the RandomSearch method by using early stopping (subsection 2.3.3) and adaptive resource allocation.

To perform optimization, the training set was split into a 90% training set a 10% validation set, randomly generated with a random seed (123123). Since it is a computationally expensive process, a validation set was used instead of the common cross-validation approach. For all optimizations, the factor parameter from the Hyperband algorithm was set to its default value (3). This means that after each evaluation performed by the algorithm at a certain number of epochs, only 1/3 of those models are kept for training until the next evaluation occurs. This process will then happen punctually until it reaches the maximum number of epochs previously set by the use. Additionally, the seed on the Hyperband algorithm was set to ensure reproducible results. However, it is noteworthy that BERT models might show some variance due to the randomness existent within the used package (HuggingFace).

The first model being optimized was the Hierarchical Attention Network, where the search space presented in Table 9 was used. For this model, the `hyperband_iterations` was set to two, meaning that this optimization process ran two times over the same search space. The maximum number of epochs was set to 40, since it was observed that the model would take around 38 epochs to converge when using a learning rate of  $1e-4$ . Moreover, it is recommended to give a small margin of epochs to guarantee convergence when using Hyperband.

Afterwards, the models containing BERT architectures were also optimized. However, since BERT models take longer to train, the option was to restrict the optimization process for a BERT model pre-trained specifically on the biomedical context, more precisely using BioBERT. Also, due to efficiency reasons, the Base version of these models was selected (which has approximately 1/3 of the parameters of the Large version) to perform optimization. Moreover, and contrarily to Hierarchical Attention Networks, the number of `hyperband_iterations` was set to only one, again, due to the high complexity of these models.

Table 9 – Search space used to optimize the Hierarchical Attention Network. In bold are marked the hyperparameters which were returned as the best hyperparameters for this model by the Hyperband Tuner after 2 iterations.

Hyperparameter	Values
Dropout 1	[0, 0.1, 0.2, 0.3, <b>0.4</b> , 0.5]
Layer 1	[' <b>LSTM</b> ', 'GRU']
Layer 1 Units	[64, <b>128</b> , 256, 512]
Dropout 2	[0, <b>0.1</b> , 0.2, 0.3, 0.4, 0.5]
Layer 2	[' <b>LSTM</b> ', 'GRU']
Layer 2 Units	[64, 128, <b>256</b> , 512]
Optimizer	['Adagrad', 'RMSprop', ' <b>Adam</b> ', 'SGD']
Learning Rate	[ $1e-2$ , <b><math>1e-3</math></b> , $1e-4$ ]
Epochs	[ <b>40</b> ]

The first model with BioBERT being optimized contained also a Dense layer on top of BioBERT, with BioBERT parameters being further fine-tuned in the course of the training process. The search space used for this optimization process can be seen in Table 10. For this optimization, the decision was to try different options, such as the use of batch normalization, distinct dropout rates and the number of dense units. Also, for these models, two optimizers were used: Adam and AdamW, with the latter having also an associated weight decay of 0.01. The maximum number of epochs was set to 5 with a batch size of 16. The chosen batch size and learning rates had as basis the recommended values by Devlin *et. al* [52] for the fine-tuning of a BERT model.

Then, hyperparameter optimization of a model containing BioBERT was performed, but this time with a Bi-LSTM on top of it. In this case, the optimization process was performed for two different versions of this model: one with BioBERT static (i.e. pre-trained BioBERT weights are frozen in this model) and another model with BioBERT being fine-tuned. Since BioBERT static represents less parameters to train, the number of values used for the learning rate parameter was decreased, and consequently increased the number of epochs used. Additionally, the batch size of the model with static BioBERT was set to 32 once it fit in the available memory, contrarily to the BioBERT fine-tuning, where there was the need to set a batch size of 16. The entire search space used can be seen in Table 11.

Table 10 – Search space used to optimize the BioBERT model with a Dense layer on top. The best hyperparameters, returned by the Hyperband algorithm after 1 iteration, are marked in bold.

Hyperparameters	BioBERT Dense
Batch Normalization	[True, <b>False</b> ]
Dropout	[0.0, 0.1, 0.2, <b>0.3</b> , 0.4, 0.5]
Dense Units	[ <b>64</b> , 128, 256]
Optimizer	[ <b>'Adam'</b> , 'AdamW']
Learning Rate	[ <b>2e-5</b> , 3e-5, 5e-5]
Weight Decay	[ <b>0.01</b> ]
Epochs	[ <b>5</b> ]
Batch Size	[ <b>16</b> ]

Finally, another model containing BioBERT was optimized, but using only the [CLS] token retrieved from it. This is a token introduced specifically on BERT models that allows its use in classification tasks as a compressed representation of the entire sequence. This can be done, since in theory, BERT models have the capacity to concentrate the most important information in that single token [52]. For the optimization process of these models, and similarly for the optimization of the model with BioBERT + Bi-LSTM, the same 2 versions were performed: static and fine-tuned, with a batch-size of 32 and 16, respectively. The search space used for the optimization of these two models can be seen in Table 12

After optimizing all hyperparameters, one can see some patterns when comparing the obtained results. For instance, it is possible to observe a negative impact of batch normalization, with all the models where it was tested providing worse results. This may be a consequence of the normalization techniques used within BERT models which may be possibly eliminating the need to use this approach. Additionally, it is possible to see that the best optimizer retrieved by the Hyperband algorithm was Adam in all the tested models. Still, this could have given different results if the weight decay parameter of the AdamW optimizer was tuned. Other than that, when comparing the results from the models *BioBERT + Bi-LSTM* with the models *BioBERT + CLS token*, it is possible to see a quite interesting pattern, although not easy to explain, with models containing BioBERT static getting the lowest values for the number of dense units while, on the contrary, fine-tuned versions got the highest values.

Table 11 – Search space used to optimize the BioBERT model with a **Bi-LSTM** for two different versions: BioBERT static and BioBERT fine-tuned. In bold are marked the best parameters resultant from the Hyperband algorithm after 1 iteration.

Hyperparameters	BioBERT LSTM (static)	BioBERT LSTM (tuned)
LSTM units	[64, 128, <b>256</b> ]	[ <b>64</b> , 128, 256]
Dropout 1	[0.0, 0.1, 0.2, 0.3, 0.4, <b>0.5</b> ]	[0.0, 0.1, <b>0.2</b> , 0.3, 0.4, 0.5]
Dense units	[ <b>64</b> , 128, 256]	[64, 128, <b>256</b> ]
Dropout 2	[ <b>0.0</b> , 0.1, 0.2, 0.3, 0.4, 0.5]	[0.0, 0.1, 0.2, 0.3, <b>0.4</b> , 0.5]
Optimizer	[ <b>'Adam'</b> , 'AdamW']	[ <b>'Adam'</b> , 'AdamW']
Learning Rate	[1e-2, <b>1e-3</b> , 1e-4]	[2e-5, <b>3e-5</b> , 5e-5]
Weight Decay	[ <b>0.01</b> ]	[ <b>0.01</b> ]
Epochs	[ <b>11</b> ]	[ <b>5</b> ]
Batch Size	[ <b>32</b> ]	[ <b>16</b> ]

Finally, it is important to notice that despite having an optimized performance, the used optimization algorithm (Hyperband) has also its downsides. In fact, the Hyperband's methodology of discarding models throughout the training process based on their evaluation may cause the loss of some good models that may need more epochs to converge to a good solution. So, in an attempt to counter this phenomenon, it was decided to also fine tune the learning rate values to provide different rates of convergence to the models. However, it may be interesting in the future to compare this optimization process with other commonly used alternatives for DL models, such as the Random Search and Bayesian optimization algorithms.

#### 4.4.4 MODELS

The first model used in this work was adapted from Burns *et al.* [9] and it consists of a CNN with a **Bi-LSTM**, and an attention layer on top. In the middle of these layers, the authors also added 3 dropout layers with a 0.4 probability each. In the end, there is also a dense layer with a sigmoid function to make the final prediction. For the embeddings, the BioWordVec pre-trained embeddings were used. The maximum number of words was set to 300 (based on Figure 24 from the Supplementary Material A) and a vocabulary size equal to the number of unique words found on the training set by the used tokenizer (around 30k words).

Table 12 – Search space used to optimize a model containing BioBERT with the selection of the CLS token on top. Optimization was done for two different versions: BioBERT static and BioBERT fine-tuned. In bold are marked the best parameters resultant from the Hyperband algorithm after 1 iteration.

Hyperparameters	BioBERT CLS (static)	BioBERT CLS (tuned)
Batch Normalization	[True, <b>False</b> ]	[True, <b>False</b> ]
Dropout 1	-	[ <b>0.0</b> , 0.1, 0.2, 0.3, 0.4, 0.5]
Dense units	[ <b>64</b> , 128, 256, 512]	[64, 128, 256, <b>512</b> ]
Dropout 2	[0.0, 0.1, <b>0.2</b> , 0.3, 0.4, 0.5]	[0.0, 0.1, 0.2, <b>0.3</b> , 0.4, 0.5]
Optimizer	[ <b>'Adam'</b> , 'AdamW']	[ <b>'Adam'</b> , 'AdamW']
Learning Rate	[1e-2, <b>1e-3</b> , 1e-4]	[ <b>2e-5</b> , 3e-5, 5e-5]
Weight Decay	[ <b>0.01</b> ]	[ <b>0.01</b> ]
Epochs	[ <b>11</b> ]	[ <b>5</b> ]
Batch Size	[ <b>32</b> ]	[ <b>16</b> ]

The second model used is a Hierarchical Attention Network with a word encoder and a sentence encoder. Both encoders contain a [Bi-LSTM](#) and an attention layer with a spatial dropout at the bottom. This model starts by converting words into word embeddings by using the BioWordVec pre-trained embeddings. At the bottom of the model, once again, a dense layer with a sigmoid function is used to make the final prediction. For this model, the maximum number of words per sentence is set to 50 and a maximum number of sentences per document to 15. These values were based on Figures [25](#) and [26](#) from the Supplementary Material [A](#), respectively. For both [CNN + Bi-LSTM](#) and Hierarchical Attention Network, a token ('OOV') was added to give an index to the unknown words (i.e. words not found on the vocabulary of the pre-trained embeddings).

The third model uses [BERT](#) with some layers on top, which can vary as already explained on the subsection [4.4.3](#). So, this model results in 3 main versions: one model with [BERT](#) and two dense layers on top, one with [BERT](#) and a [Bi-LSTM](#) with also two dense layers on top, and a final version, similar to the first one, but using the *CLS* token instead of the embeddings of all tokens resulting from the [BERT](#) model. All models have also dropout layers, with the first two versions of this third model ([BERT](#) + Dense and [BERT](#) + *CLS*) containing also a 1D global average pooling to convert the embeddings into a 1D vector. For these models, the maximum number of words was set to 512 since it is the maximum value allowed on [BERT](#) models. The

number of sentences was set to only one - meaning that title and abstract are grouped together as a single sentence. It was also tested to set this number to two, with the title used as one sentence and the abstract as another sentence, but no improvements were observed. Moreover, beyond the 3 mentioned versions, the use of either static (i.e. frozen) and non-static weights were also tried from pre-trained [BERT](#) models. For these pre-trained [BERT](#) models, 3 different options were tried :

- **BERT-base-uncased:** pre-trained on lower-cased English texts from Wikipedia and Book-Corpus using a [MLM](#) objective. The structure of this model was previously explained on subsection [2.3.9](#).
- **SciBERT-base-uncased:** based on the [BERT](#) model and pre-trained on more than 1 million lower-cased full text documents from the Semantic Scholar database. This model also offers a custom vocabulary (around 30000 words) named scivocab, with the aim of improving tokenization of words related to the biomedical context.
- **BioBERT-base-cased:** version 1.1 was selected, which is based on the [BERT](#) model and pre-trained on more than 1 million PubMed documents. In this model, the provided vocabulary is similar to the one used in the [BERT](#) model (around 30000 words), but containing also cased words. The authors decided to not create a new vocabulary for two reasons: they wanted to use Google's pre-trained [BERT](#) which needs the same vocabulary and for believing that the sub-word technique of [BERT](#) is capable of retrieving similar information by splitting unknown words.

All these pre-trained [BERT](#) models follow a similar structure with a hidden size of 768, 12 attention heads, 12 hidden layers and a dropout probability of 0.1 for both attention and hidden layers. Moreover, as activation function for the hidden layers, all these pre-trained models use a function called Gaussian Error Linear Unit (GELU), which was recently proposed as an alternative to [ReLU](#) and [ELU](#).

For all models, truncating and padding were set to post. The main reasons for this choice were: firstly, to assure that words belonging to the title were not removed when performing truncation, and secondly, to keep the title always in the starting positions of the input, when performing padding.

Then, a 10-fold cross validation was performed on these models to see how they would perform on our training set. When comparing the results obtained (Fig. 23), it is possible to observe that BioBERT static + CLS and CNN-Bi-LSTM were the two models with the worst results of all. On the contrary, BERT models got the best results, with BERT models pre-trained on scientific literature (SciBERT and BioBERT) getting a slightly better performance when compared to BERT. Also, when comparing SciBERT with BioBERT (SciBERT+Dense(tuned) vs BioBERT+Dense(tuned) and SciBERT+CLS(tuned) vs BioBERT+CLS(tuned)), it is possible to see that BioBERT managed to slightly overcome models with SciBERT, with a better average precision and f1-score means, which in fact, were the best results of all models. Finally, and despite our dataset being relatively small, one can also observe a positive impact of fine-tuning BERT models, showing better scores when compared to the static versions.

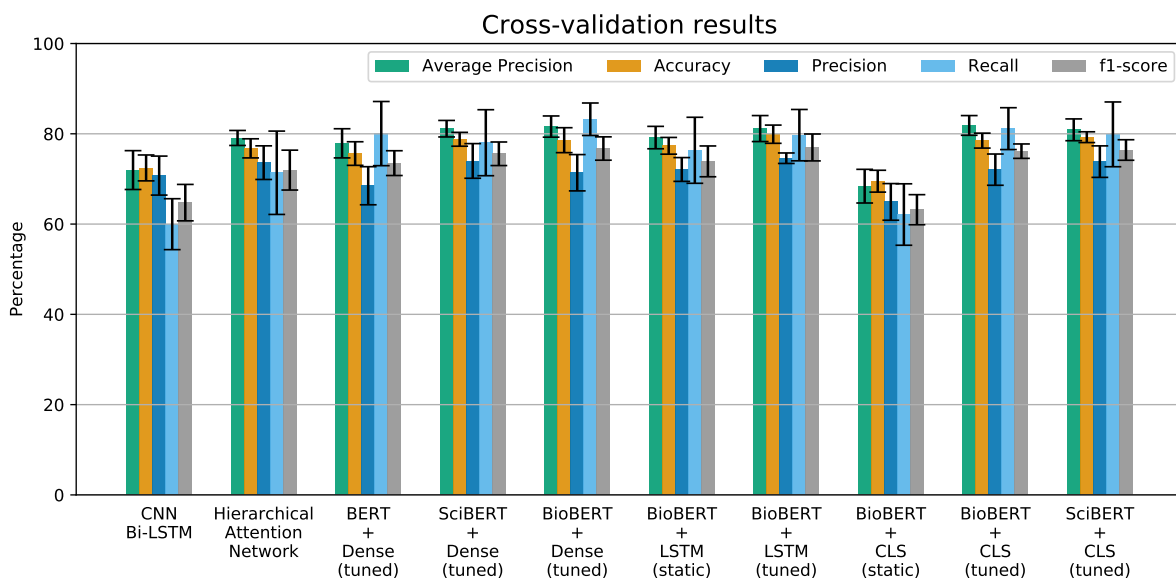


Fig. 23 – 10 fold cross-validation results (mean and standard deviation for each metric) performed on the training set using different DL architectures

As expected, due to their high complexity and for being state-of-the-art models in many NLP tasks, BERT models (BERT, SciBERT and BioBERT) managed to outperform all the other models, with SciBERT and BioBERT outperforming BERT. SciBERT and BioBERT outperforming BERT was also expected since these 2 models were pre-trained specifically on scientific documents. Also, in general, models with BioBERT outperformed models containing SciBERT. This might have happened since BioBERT was pre-trained on documents from the PubMed database, the same database used to create the dataset used in this work.



Nevertheless, it is important to mention that these slightly better results of BioBERT over SciBERT can also be a consequence of the followed approach for the models' hyperparameters optimization, where only the top layers of these models were optimized using the BioBERT pre-trained model, due to their highly time-consuming training process. In any case, this decision was based on a previous evaluation of these models with some manual tuning, where BioBERT already had presented better scores over the other 2 models. However, it would be valuable to compare these models after optimizing hyperparameters specifically for each one of them. Another difference between the BioBERT and SciBERT versions used in this work is the use of capital letters on BioBERT's vocabulary. Nonetheless, we suspect this had no impact on the results, since it showed also no impact when tested in other models (CNN-Bi-LSTM and Hierarchical).

Regarding the two worst models (CNN-Bi-LSTM and BioBERT + CLS (static)), there are also some possible reasons that may explain the obtained results. For instance, the worst performance of the CNN-Bi-LSTM model may be related to the non application of the hyperparameter optimization process on this model. The main motivation to use this model with the exact same parameters used by Burns *et. al* [9] was the intention to use this model as a basis to the other DL implementations. Nevertheless, it is still possible to affirm that this model obtained relatively good results, considering the fact that it has a much lower complexity when compared to the other models used (containing almost 2k less trainable parameters than for instance models with BERT).

Regarding BioBERT + CLS (static) model, there is a likely reason that may explain its results related with the use of the CLS token. The use of this token means that the model will use a representation that attempts to incorporate, in a compressed way, all the important information of a single document. So, the obtained results might mean that the static version of this model is not able to capture, and consequently integrate, that important information into that token if not trained on the new data. In fact, since BioBERT with CLS (tuned) showed better results, it seems that tuning this model is almost mandatory when using CLS, to allow the model to learn how to properly capture the relevant information into that token.

#### 4.4.5 EVALUATION

Afterwards, these models were evaluated on our test set. For the the final training of **CNN + Bi-LSTM** and Hierarchical attention network, the early-stopping callback was used (with a patience of 5) to control the validation loss. This callback is highly recommended to be used after performing the hyperparameter optimization with the Hyperband algorithm. Additionally, this callback was combined with the `ModelCheckpoint` callback, that in our case was used to save model weights from the epoch with the best validation loss. However, early-stopping was not used when fine-tuning **BERT** models once these models take only a few epochs to converge (around 2-4). Instead, we evaluated them using different number of epochs on the training process, which were as well saved with the `Modelcheckpoing` callback. For all models, a randomly generated validation set with 10% of the training set was used to either allow the use of early-stopping or visualize training history. The results obtained on the test set can be seen in Table 13.

Observing the results obtained on the test set, it is possible to see that they were quite satisfactory, with almost every model containing **BERT** models pre-trained on scientific text (BioBERT and SciBERT - with exception for BD2 and SC2) getting better scores for all the metrics when compared to the challenge's best submission. Additionally, we can see that our best Hierarchical model managed to get a slightly better f1-score and more 2.67 percentage points on the average precision metric in comparison to the best submission, which also used a Hierarchical network. There are some reasons which may have caused this slight improvement, such as a better hyperparameter optimization, the use of **LSTMs** instead of **GRUs**, distinct preprocessing steps, or the use of title together with the abstract throughout the entire model, contrarily to the approach used by the authors of the best submission, where title was passed directly from the sentence encoder to the final dense layer. Also, this little improvement can be associated with the use of BioWordVec as feature extractor, which showed slightly better results over the pre-trained embeddings used by that same team ("pubmed\_pmc").

Furthermore, one can see that the two best models were BioL1 and BioL2 with the two best values of f1-score. In spite of BioL2 getting the best f1-score result of all, we consider that our best model is BioL1. This is because, despite the slightly worst f1-score, BioL1 has a better average precision (an important metric for document relevance) with an higher difference

Table 13 – Results obtained on the test set using different models. In bold are marked the values that beat the best submission result on that metric, while in red are marked the best values that we got for each metric.

ID	Model	Avg Prec	Precision	Recall	F1-score
CLB	CNN-BiLSTM	0.6756	0.5980	<b>0.7670</b>	0.6721
H1	Hierarchical Bi-LSTM	<b>0.7560</b>	<b>0.6614</b>	0.7102	0.6849
H2	Hierarchical Bi-LSTM	<b>0.7437</b>	<b>0.6404</b>	0.7415	0.6873
H3	Hierarchical Bi-LSTM	<b>0.7425</b>	0.6145	<b>0.8040</b>	<b>0.6966</b>
BD1	BERT (tuned) + Dense	0.7141	0.6141	<b>0.8182</b>	<b>0.7016</b>
BD2	BERT (tuned) + Dense	0.7145	<b>0.6400</b>	0.7628	<b>0.6960</b>
BioD1	BioBERT (tuned) + Dense	<b>0.7763</b>	<b>0.6376</b>	<b>0.7997</b>	<b>0.7095</b>
BioD2	BioBERT (tuned) + Dense	<b>0.7709</b>	<b>0.6245</b>	<b>0.8267</b>	<b>0.7115</b>
BioD3	BioBERT (tuned) + Dense	<b>0.7727</b>	<b>0.6755</b>	0.7599	<b>0.7152</b>
BioD4	BioBERT (tuned) + Dense	<b>0.7798</b>	<b>0.6331</b>	<b>0.8239</b>	<b>0.7160</b>
SD1	SciBERT (tuned) + Dense	<b>0.7599</b>	<b>0.6459</b>	<b>0.7798</b>	<b>0.7066</b>
SD2	SciBERT (tuned) + Dense	<b>0.7678</b>	<b>0.6548</b>	<b>0.7813</b>	<b>0.7124</b>
BioC1	BioBERT (tuned) + CLS	<b>0.7675</b>	<b>0.6402</b>	<b>0.7784</b>	<b>0.7026</b>
BioC2	BioBERT (tuned) + CLS	<b>0.7727</b>	<b>0.6439</b>	<b>0.8040</b>	<b>0.7151</b>
BioC3	BioBERT (tuned) + CLS	<b>0.7828</b>	<b>0.6471</b>	<b>0.7969</b>	<b>0.7104</b>
SC1	SciBERT (tuned) + CLS	<b>0.7370</b>	<b>0.6298</b>	<b>0.7926</b>	<b>0.7019</b>
SC2	SciBERT (tuned) + CLS	<b>0.7672</b>	<b>0.6927</b>	0.6918	<b>0.6923</b>
BioL1	BioBERT (tuned) + LSTM	<b>0.7883</b>	<b>0.6611</b>	<b>0.7955</b>	<b>0.7221</b>
BioL2	BioBERT (tuned) + LSTM	<b>0.7787</b>	<b>0.6635</b>	<b>0.7955</b>	<b>0.7235</b>
SL1	SciBERT (tuned) + LSTM	<b>0.7455</b>	<b>0.6414</b>	<b>0.7798</b>	<b>0.7038</b>
SL2	SciBERT (tuned) + LSTM	<b>0.7343</b>	<b>0.6311</b>	<b>0.7898</b>	<b>0.7016</b>
<b>Challenge's Best Submission</b>		0.7158	0.6289	0.7656	0.6906

between the two models. Nevertheless, it is worth mentioning that the difference observed on the results of these two models is only a cause of the variation associated with the fine-tuning process of BERT models, once they resulted from a same training process with 2 epochs. When comparing our best model with the challenge's best submission, we can see that our model managed to improve results in all metrics with a significant difference of 7.25% for average precision, 3.22% for precision, 2.99% for recall and 3.15% for f1-score.

After the evaluation of our models, the next step was to understand some of the possible reasons behind the wrong predictions. In order to do so, 4 subsets were created of the test set based on the confusion matrix results (FP, FN, TP and TN) and plotted the top 20 words for each one of them. The results can be seen in Supplementary Material A - Fig. 30. Interestingly, documents classified as relevant (positives) contained in their top 20 words, words related to our topic ("mutation", "alpha" and "interaction") that are not present in the top 20 words of documents classified as non-relevant (negatives). Oppositely, documents classified as non-relevant contain words as "dependent", "induced" and "p53", which are not present in the top 20 words of the other class. By comparing words present on the top 20 words of all subsets, it is also possible to see an higher frequency of words such as "binding" and "receptor", which are also related to our topic, in the positive class over the negative class.

Despite these curious patterns, it should be noted that this analysis only considers one aspect used in the model, and that other variables may have an equal or even higher impact on the final prediction, such as words' position and order in a sentence. Finally, it should be also noted that when creating the dataset used in this work, curators defined some documents that did not have any mentions to the interacting proteins on both title and abstract as relevant documents (i.e. documents that had information on title/abstract suggesting that the relevant parts about the topic would be on the full text, curators would consider this fact enough to consider that document as relevant) [152]. Yet, this brings a problem to the dataset, since it only provides title + abstract. Consequently, this may produce some FN in our predictions, once the model cannot find those mentions in the used text.

By looking into our evaluation scores, it is also possible to see that models got slightly worst results on the test set when compared to the cross-validation results, which was performed in subsection 4.4.4. This phenomenon might indicate that our test set has a slightly different distribution from our training set, turning its prediction a more difficult task for our models.

Finally, and despite outperforming the challenge's best submissions, there might still be space to enhance our results by changing some of the approaches used in the distinct steps of this work:

- **Dataset:** possibly complement title and abstract with other information/text about the documents. For instance, Burns *et. al* [9] observed that using captions of documents benefited the document triage on their dataset.
- **Preprocessing:** BERT models have a limit of 512 words per document. So, in this work, documents were truncated with more than 512 by removing the last words of the abstract. However, this can remove some important words that could possibly have an impact on the final prediction. In any case, we believe this did not happen on this dataset, since BioBERT managed to overcome SciBERT, even with the latter being able to produce less tokens. The reason SciBERT produces less tokens is related to its available custom vocabulary, specifically designed for biomedical documents (Supplementary Material - Fig. 27). Nevertheless, it would be interesting to implement in the future some alternatives to the used truncation process. Some of the known alternatives used are chunking and text summarization. Recently, Sun *et. al* [153], in a article focused on the fine-tuning of BERT for text classification, found that a different method of truncation might be enough to improve results ("head+tail" - where they used the first 129 tokens together with the last 389).
- **Hyperparameter Optimization:** for this process it would be important to both increase the number of iterations used, and to fine-tune the parameter search based on the hyperparameters obtained on the already done optimization process. Also, it would be interesting to test other possibilities not used in our optimization process such as other activation functions or L2 regularization. It would also be possible to optimize the structure and/or some of the hyperparameters of BERT models (e.g dropout rate of self-attention layers).
- **Validation set:** in this work, a randomly generated validation set was used for two different steps: hyperparameter optimization and training process. In ML models, it is important to make sure that the validation set follows a similar distribution as the test, which was not possible since there was a pre-defined held-out test set. One of the impacts of this

is, for instance, the overfit of the hyperparameters, from the hyperparameter optimization process, to the used validation set. A possible alternative is the use of the cross-validation technique in this process, not used since it would take a very long time to run cross-validation on models with BERT layers. Furthermore, the randomly generated validation set may introduce another problem related to the use of an imbalanced dataset.

- **Models:** possibly test in the future a recently developed model called SPECTER [154]. This model uses transformer language models such as SciBERT with a new objective function called "Citation-Based Pretraining Objective". The intuition behind this objective function is the use of cited documents of a certain document to approximate their document embeddings. Also, it would be possible to implement an ensemble of our final models.
- **Early-stopping:** for early-stopping validation loss was used. However, it could be interesting to try the approach of Fergadis *et. al* [40], where they used a metric that was being evaluated by the challenge (f1-score). In this work, the option was to not do it since the common implementation of this metric on the training process of Keras gives a result that consists on the mean of all the f1-scores obtained on each batch for a certain epoch, which might be misleading.

In case a user needs to classify documents not related to PPI altered by mutations, one can always use our developed package to train his dataset and consequently make new predictions.

---

## CONCLUSION

---

One of the main objectives of this thesis was the development of a package (BioTMPy) to provide an easy-to-use and intuitive tool to perform document classification on biomedical literature. BioTMPy is divided into different modules that allows its use on independent tasks. These modules provide methods to read datasets in different formats, create document objects, preprocess text data, generate features, perform data analysis (e.g. [t-SNE](#), label balance, top number of words) and implement both shallow [ML](#) (e.g. [SVM](#), Random Forest, Logistic Regression) and [DL](#) models (e.g. Hierarchical Attention Networks, SciBERT, BioBERT). Furthermore, it offers step-by-step pipelines with examples on how to perform the document classification task from start to finish. BioTMPy also offers pipelines to perform hyperparameter optimization and cross-validation. At the end, a user can as well deploy the developed models in a web server to the community using the web server module. The package can also be used in other types of documents not related to the biomedical field and be easily adapted to be used in other tasks that rely on the power of [NLP](#), such as the prediction of compound-protein interactions. In the future it would be interesting to also extend this package to other [BioTM](#) tasks such as information extraction and implement some other features/modules with the aim of simplifying its use.

The package was then applied on a dataset from the BioCreative's track 4 from task VI about [PPI](#) altered by mutations. Analysing the dataset, it was possible to observe the presence of an high similarity between documents from the two existent labels and that cluster algorithms had difficulties in making a clear division of documents based on their relevance, which indicates that classifying documents by relevance is an hard task on this dataset. With a comparison made between different pre-trained embeddings we were able to conclude that BioWordVec was

able to obtain a slightly better performance over GloVe and two other embeddings pre-trained with pubmed documents ("pubmed\_pmc" and "pubmed\_ncbi").

Afterwards, the package was used to perform hyperparameter optimization, cross-validation and evaluation of different DL models. The results confirmed that BERT models, state-of-the-art in many NLP tasks, performed better also on the document triage of the used dataset. Also, this work confirms a better performance of BERT models pre-trained on scientific text (BioBERT and SciBERT) when fine-tuned on biomedical documents, with a slight advantage of BioBERT over SciBERT. One of the possible reasons to this advantage of BioBERT is the fact that it was pre-trained on documents from PubMed, the same database used for the generation of the dataset used in this work. As a result, a model with BioBERT with a Bi-LSTM on top would give the best results on the test set with a difference of 7.25% for average precision, 3.22% for precision, 2.99% for recall and 3.15% for f1-score when compared the the challenge's best submission. Nevertheless, further application of different approaches may still improve the obtained results, such as improving the truncating process, hyperparameter optimization (e.g. by fine-tuning the search space), creation of the validation set, and try new DL models such as SPECTER. To the best of our knowledge, this was the first implementation of BERT models on this specific dataset.

In a future work, it would then be extremely important to extend the approach used to other topics and to try this same pipeline on an independent biomedical dataset. Nevertheless, the web server can be really helpful to rank documents about PPI altered by mutations since, beyond the model's good results, the search term system works over documents retrieved by the relevance system of PubMed, functioning as a second filter of those results. To the best of our knowledge, this is the first tool available to rank biomedical documents relying on Deep Learning models.



---

## BIBLIOGRAPHY

---

- [1] Krallinger, M., & Valencia, A. (2005). Text-mining and information-retrieval services for molecular biology.
- [2] Gong, L. (2018). Application of Biomedical Text Mining, In *Artificial intelligence - emerging trends and applications*. InTech.
- [3] Shatkay, H., & Craven, M. (2012). *Mining the Biomedical Literature (Computational Molecular Biology)*. <https://doi.org/10.1111/mec.12185>
- [4] Westergaard, D., Stærfeldt, H. H., Tønsberg, C., Jensen, L. J., & Brunak, S. (2018). A comprehensive and quantitative comparison of text-mining in 15 million full-text articles versus their corresponding abstracts. *PLoS Computational Biology*, 14(2), e1005962. <https://doi.org/10.1371/journal.pcbi.1005962>
- [5] Feldman, R., & Sanger, J. (2007). *The text mining handbook : advanced approaches in analyzing unstructured data*. Cambridge University Press.
- [6] Kowsari, K., Meimandi, K. J., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text classification algorithms: A survey. <https://doi.org/10.3390/info10040150>
- [7] Chollet, F. (2018). *Deep Learning with Python*.
- [8] Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016). Hierarchical attention networks for document classification, In *2016 conference of the north american chapter of the association for computational linguistics: Human language technologies, naacl hlt 2016 - proceedings of the conference*. <https://doi.org/10.18653/v1/n16-1174>
- [9] Burns, G. A., Li, X., & Peng, N. (2019). Building deep learning models for evidence classification from the open access biomedical literature. *Database : the journal of biological databases and curation*, 2019. <https://doi.org/10.1093/database/baz034>
- [10] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014a). Learning phrase representations using RNN encoder-decoder for statistical machine translation, In *Emnlp 2014 - 2014 conference on empirical methods in natural language processing, proceedings of the conference*. <https://doi.org/10.3115/v1/d14-1179>
- [11] Mirończuk, M. M., & Protasiewicz, J. (2018). A recent overview of the state-of-the-art elements of text classification. *Expert Systems with Applications*, 106, 36–54. <https://doi.org/10.1016/J.ESWA.2018.03.058>
- [12] Airola, A., Pyysalo, S., Björne, J. Et al. (2008). All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning. <https://doi.org/10.1186/1471-2105-9-S11-S2>

- [13] Fiorini, N., Canese, K., Starchenko, G., Kireev, E., Kim, W., Miller, V., Osipov, M., Kholodov, M., Ismagilov, R., Mohan, S., Ostell, J., & Lu, Z. (2018). Best Match: New relevance search for PubMed. *PLOS Biology*, 16(8), e2005343. <https://doi.org/10.1371/journal.pbio.2005343>
- [14] 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(3), 575–603.
- [15] Kao, A., & Poteet, S. R. (2007). *Natural Language Processing and Text Mining*. [https://doi.org/10.1007/978-1-84628-754-1\\_1](https://doi.org/10.1007/978-1-84628-754-1_1)
- [16] Miner, G., Delen, D., Elder, J., Fast, A., Hill, T., & Nisbet, R. A. (Eds.). (2012a). Chapter 2 - the seven practice areas of text analytics, In *Practical text mining and statistical analysis for non-structured text data applications*. Boston, Academic Press. <https://doi.org/10.1016/B978-0-12-386979-1.00002-5>
- [17] Bollen, J., Mao, H., & Zeng, X. (2011). Twitter mood predicts the stock market. *Journal of Computational Science*, 2(1), arXiv 1010.3003, 1–8. <https://doi.org/10.1016/j.jocs.2010.12.007>
- [18] Kallus, N. (2014). Predicting crowd behavior with big public data, In *Proceedings of the 23rd international conference on world wide web - www '14 companion*, New York, New York, USA, ACM Press. <https://doi.org/10.1145/2567948.2579233>
- [19] Simpson, M. S., & Demner-Fushman, D. (2012). Biomedical Text Mining: A Survey of Recent Progress, In *Mining text data*. Boston, MA, Springer US. [https://doi.org/10.1007/978-1-4614-3223-4\\_14](https://doi.org/10.1007/978-1-4614-3223-4_14)  
exemp3
- [20] Zweigenbaum, P., Demner-fushman, D., Yu, H., & Cohen, K. B. (2007). Frontiers of biomedical text mining: Current progress. <https://doi.org/10.1093/bib/bbmo45>
- [21] Rodriguez-Esteban, R. (2009). Biomedical Text Mining and Its Applications (F. Lwitter, Ed.). *PLoS Computational Biology*, 5(12), e1000597. <https://doi.org/10.1371/journal.pcbi.1000597>
- [22] Swanson, D. R. (1988). Migraine and magnesium: eleven neglected connections. <https://doi.org/10.1353/pbm.1988.0009>
- [23] Saffer, J. D., & Burnett, V. L. (2014). Introduction to biomedical literature text mining: Context and objectives. *Methods in Molecular Biology*, 1159. [https://doi.org/10.1007/978-1-4939-0709-0\\_1](https://doi.org/10.1007/978-1-4939-0709-0_1)
- [24] Cohen, A. M., & Hersh, W. R. (2005). A survey of current work in biomedical text mining. *Briefings in Bioinformatics*, 6(1), 57–71. <https://doi.org/10.1093/bib/6.1.57>
- [25] Ignatow, G., & Mihalcea, R. (2018). *An Introduction to Text Mining : Research Design, Data Collection, and Analysis*. <https://doi.org/10.1017/CBO9781107415324.004>
- [26] Sarkar, D. (2019). Text classification. In *Text analytics with python: A practitioner's guide to natural language processing* (pp. 275–342). Berkeley, CA, Apress. <https://doi.org/10.1007/978-1-4842-4354-1-5>

- [27] Bengfort, B., Billbro, R., & Ojeda, T. (2018). *Applied text analysis with Python : enabling language-aware data products with machine learning*.
- [28] Miner, G., Delen, D., Elder, J., Fast, A., Hill, T., & Nisbet, R. A. (Eds.). (2012b). Chapter 5 - text mining methodology, In *Practical text mining and statistical analysis for non-structured text data applications*. Boston, Academic Press. <https://doi.org/https://doi.org/10.1016/B978-0-12-386979-1.00005-0>
- [29] Miner, G., Delen, D., Elder, J., Fast, A., Hill, T., & Nisbet, R. A. (Eds.). (2012c). Chapter 7 - text classification and categorization, In *Practical text mining and statistical analysis for non-structured text data applications*. Boston, Academic Press. <https://doi.org/https://doi.org/10.1016/B978-0-12-386979-1.00035-9>
- [30] Luhn, H. P. (1957). A Statistical Approach to Mechanized Encoding and Searching of Literary Information. *IBM Journal of Research and Development*, 1(4), 309–317. <https://doi.org/10.1147/rd.14.0309>
- [31] Spärck Jones, K. (2004). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 60(5), 493–502. <https://doi.org/10.1108/00220410410560573>
- [32] Hirschman, L., Yeh, A., Blaschke, C., & Valencia, A. (n.d.). Overview of BioCreAtIvE: critical assessment of information extraction for biology. <https://doi.org/10.1186/1471-2105-6-S1-S1>
- [33] Kim, J.-D., Pyysalo, S., Ohta, T., Bossy, R., Nguyen, N., & Ichi Tsujii, J. ' . (2011). *Overview of BioNLP Shared Task 2011* (tech. rep.). <http://patricbrc.org>
- [34] Uzuner, Ö. (2009). Obesity NLP Challenge Recognizing Obesity and Comorbidities in Sparse Data. *AMIA*, 16, 561–570. <https://doi.org/10.1197/jamia.M3115>
- [35] Poux, S., Arighi, C. N., Magrane, M., Bateman, A., Wei, C. H., Lu, Z., Boutet, E., Bye-A-Jee, H., Famiglietti, M. L., Roechert, B., & UniProt Consortium, T. (2017). On expert curation and scalability: UniProtKB/Swiss-Prot as a case study. *Bioinformatics (Oxford, England)*, 33(21), 3454–3460. <https://doi.org/10.1093/bioinformatics/btx439>
- [36] Rahman, M. M., & Davis, D. N. (2013). Addressing the Class Imbalance Problem in Medical Datasets. *International Journal of Machine Learning and Computing*, 224–228. <https://doi.org/10.7763/ijmlc.2013.v3.307>
- [37] Schneider, G., Clematide, S., & Rinaldi, F. (2011). Detection of interaction articles and experimental methods in biomedical literature. *BMC Bioinformatics*, 12(SUPPL. 8). <https://doi.org/10.1186/1471-2105-12-S8-S13>
- [38] Laza, R., Pavón, R., Pavón, P., Reboiro-Jato, M., & Fdez-Riverola, F. (2011). Evaluating the effect of unbalanced data in biomedical document classification. <https://doi.org/10.2390/biecoll-jib-2011-177>
- [39] Jiang, X., Ringwald, M., Blake, J. A., Arighi, C., Zhang, G., & Shatkay, H. (2019). An effective biomedical document classification scheme in support of biocuration: addressing

- class imbalance. *Database : the journal of biological databases and curation*, 2019. <https://doi.org/10.1093/database/baz045>
- [40] Fergadis, A., Baziotis, C., Pappas, D., Papageorgiou, H., & Potamianos, A. (2018). Hierarchical bi-directional attention-based RNNs for supporting document classification on protein-protein interactions affected by genetic mutations. *Database*, 2018(2018), 76. <https://doi.org/10.1093/database/bay076>
- [41] Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., & McClosky, D. (2015). The Stanford CoreNLP Natural Language Processing Toolkit, Association for Computational Linguistics (ACL). <https://doi.org/10.3115/v1/p14-5010>
- [42] Parwez, M. A., Abulaish, M., & Jahiruddin. (2019). Multi-Label Classification of Microblogging Texts Using Convolution Neural Network. *IEEE Access*, 7, 68678–68691. <https://doi.org/10.1109/ACCESS.2019.2919494>
- [43] Bengio, Y., Ducharme, R., & Vincent, P. (2003). A neural probabilistic language model, In *Advances in neural information processing systems*.
- [44] Levy, O., & Goldberg, Y. (2014). Neural word embedding as implicit matrix factorization (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger, Eds.). In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 27*. Curran Associates, Inc. <http://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matrix-factorization.pdf>
- [45] Mikolov, T., Le, Q. V., & Sutskever, I. (2013). Exploiting Similarities among Languages for Machine Translation, arXiv 1309.4168. <https://code.google.com/p/word2vec/%20http://arxiv.org/abs/1309.4168>
- [46] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space, arXiv 1301.3781. <http://ronan.collobert.com/senna/%20http://arxiv.org/abs/1301.3781>
- [47] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality, In *Advances in neural information processing systems*. <http://arxiv.org/abs/1310.4546>
- [48] Chiu, B., Crichton, G., Korhonen, A., & Pyysalo, S. (2016). How to Train good Word Embeddings for Biomedical NLP, Association for Computational Linguistics (ACL). <https://doi.org/10.18653/v1/w16-2922>
- [49] Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global vectors for word representation, In *Emnlp 2014 - 2014 conference on empirical methods in natural language processing, proceedings of the conference*. <https://doi.org/10.3115/v1/d14-1162>
- [50] Makarek, V., Shapira, B., & Rokach, L. (2016). Language Models with Pre-Trained (GloVe) Word Embeddings, arXiv 1610.03759. <http://nlp.stanford.edu/projects/glove/%20http://arxiv.org/abs/1610.03759>

- [51] Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep Contextualized Word Representations. <https://doi.org/10.18653/v1/n18-1202>
- [52] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, arXiv 1810.04805. <https://github.com/tensorflow/tensor2tensor%20http://arxiv.org/abs/1810.04805>
- [53] Asgari, E., & Mofrad, M. R. (2015). Continuous distributed representation of biological sequences for deep proteomics and genomics. *PLoS one*, 10(11), e0141287.
- [54] Ghosh, S., Chakraborty, P., Cohn, E., Brownstein, J. S., & Ramakrishnan, N. (2016). Characterizing diseases from unstructured text: A vocabulary driven word2vec approach, In *Proceedings of the 25th acm international on conference on information and knowledge management*. ACM.
- [55] Moen, S., & Ananiadou, T. S. S. (2013). Distributional semantics resources for biomedical text processing. *Proceedings of LBM*, 39–44.
- [56] Le, Q., & Mikolov, T. (2014). Distributed representations of sentences and documents, In *International conference on machine learning*.
- [57] Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5, 135–146.
- [58] Zhang, Y., Chen, Q., Yang, Z., Lin, H., & Lu, Z. (2019). BioWordVec, improving biomedical word embeddings with subword information and MeSH. *Scientific data*, 6(1), 52. <https://doi.org/10.1038/s41597-019-0055-0>
- [59] Kim, S., Fiorini, N., Wilbur, W. J., & Lu, Z. (2016). Bridging the gap: Incorporating a semantic similarity measure for effectively mapping pubmed queries to documents.
- [60] Melamud, O., Goldberger, J., & Dagan, I. (2016). Context2vec: Learning generic context embedding with bidirectional lstm, In *Proceedings of the 20th sigll conference on computational natural language learning*.
- [61] Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., & Kang, J. (2019). BioBERT: a pre-trained biomedical language representation model for biomedical text mining [btz682]. *Bioinformatics*. <https://doi.org/10.1093/bioinformatics/btz682>
- [62] Beltagy, I., Lo, K., & Cohan, A. (2019). SciBERT: A Pretrained Language Model for Scientific Text, arXiv 1903.10676. <http://arxiv.org/abs/1903.10676>
- [63] Peng, Y., Yan, S., & Lu, Z. (2019). Transfer Learning in Biomedical Natural Language Processing: An Evaluation of BERT and ELMo on Ten Benchmarking Datasets, arXiv 1906.05474, 58–65. <http://arxiv.org/abs/1906.05474>
- [64] Krallinger, M., Leitner, F., Rabal, O., Vazquez, M., Oyarzabal, J., & Valencia, A. (2015). CHEMDNER: The drugs and chemical names extraction challenge. *Journal of Cheminformatics*, 7(S1), S1. <https://doi.org/10.1186/1758-2946-7-S1-S1>

- [65] Smith, L., Tanabe, L. K., Ando, R., Kuo, C. J., Chung, I. F., Hsu, C. N., Lin, Y. S., Klinger, R., Friedrich, C. M., Ganchev, K., Torii, M., Liu, H., Haddow, B., Struble, C. A., Povinelli, R. J., Vlachos, A., Baumgartner, W. A., Hunter, L., Carpenter, B., . . . Wilbur, W. J. (2008). Overview of BioCreative II gene mention recognition. <https://doi.org/10.1186/gb-2008-9-s2-s2>
- [66] Crichton, G., Pyysalo, S., Chiu, B., & Korhonen, A. (2017). A neural network multi-task learning approach to biomedical named entity recognition. *BMC Bioinformatics*, 18(1), 368. <https://doi.org/10.1186/s12859-017-1776-8>
- [67] Rodrigues, R., Costa, H., & Rocha, M. (2018). Automating the extraction of essential genes from literature, In *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)*. [https://doi.org/10.1007/978-3-319-95786-9\\_6](https://doi.org/10.1007/978-3-319-95786-9_6)
- [68] Lourenço, A., Carreira, R., Carneiro, S., Maia, P., Glez-Peña, D., Fdez-Riverola, F., Ferreira, E. C., Rocha, I., & Rocha, M. (2009). @Note: A workbench for Biomedical Text Mining. *Journal of Biomedical Informatics*, 42(4), 710–720. <https://doi.org/10.1016/j.jbi.2009.04.002>
- [69] Wei, C. H., Kao, H. Y., & Lu, Z. (2013). PubTator: a web-based text mining tool for assisting biocuration. *Nucleic acids research*, 41(Web Server issue). <https://doi.org/10.1093/nar/gkt441>
- [70] Luo, L., Yang, Z., Lin, H., & Wang, J. (2018). Document triage for identifying protein-protein interactions affected by mutations: A neural network ensemble approach. *Database*, 2018(2018), 1–12. <https://doi.org/10.1093/database/bay097>
- [71] Baştanlar, Y., & Özuysal, M. (2014). Introduction to machine learning. *Methods in Molecular Biology*, 1107, 105–128. [https://doi.org/10.1007/978-1-62703-748-8\\_7](https://doi.org/10.1007/978-1-62703-748-8_7)
- [72] Attaran, M., & Deb, P. (2018). Machine Learning: The New 'Big Thing' for Competitive Advantage. *International Journal of Knowledge Engineering and Data Mining*, 5(1), 1. <https://doi.org/10.1504/ijkedm.2018.10015621>
- [73] Burkov, A. (2019). The Hundred-Page Machine Learning Book-Andriy Burkov. *Expert Systems*, 5(2), arXiv arXiv:1011.1669v3, 132–150. <https://doi.org/10.1111/j.1468-0394.1988.tb00341.x>
- [74] Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11), 559–572. <https://doi.org/10.1080/14786440109462720>
- [75] Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6), 417–441. <https://doi.org/10.1037/h0071325>
- [76] Mishra, S., Sarkar, U., Taraphder, S., Datta, S., Swain, D., Saikhom, R., Panda, S., & Laishram, M. (2017). Principal Component Analysis. *International Journal of Livestock Research*, 1. <https://doi.org/10.5455/ijlr.20170415115235>

- [77] Van Der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9, 2579–2625.
- [78] Van Der Maaten, L. (2015). Accelerating t-SNE using tree-based algorithms. *Journal of Machine Learning Research*, 15, 3221–3245.
- [79] Hinton, G., & Roweis, S. (2003). Stochastic neighbor embedding, In *Advances in neural information processing systems*. [https://www.researchgate.net/publication/2934728%7B%5C\\_%7DStochastic%7B%5C\\_%7DNeighbor%7B%5C\\_%7DEmbedding](https://www.researchgate.net/publication/2934728%7B%5C_%7DStochastic%7B%5C_%7DNeighbor%7B%5C_%7DEmbedding)
- [80] Liang, H., Sun, X., Sun, Y., & Gao, Y. (2017). Text feature extraction based on deep learning: a review. Springer International Publishing. <https://doi.org/10.1186/s13638-017-0993-1>
- [81] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. <http://www.deeplearningbook.org>
- [82] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation* (tech. rep.). California Univ San Diego La Jolla Inst for Cognitive Science.
- [83] Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786), 504–507. <https://doi.org/10.1126/science.1127647>
- [84] Géron, A. (2019). *Hands-on machine learning with scikit-learn, keras, and tensorflow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.
- [85] Baldi, P., Brunak, S., Chauvin, Y., Andersen, C. A., & Nielsen, H. (2000). Assessing the accuracy of prediction algorithms for classification: An overview. <https://doi.org/10.1093/bioinformatics/16.5.412>
- [86] Yonelinas, A. P., & Parks, C. M. (2007). Receiver operating characteristics (rocs) in recognition memory: A review. *Psychological bulletin*, 133(5), 800.
- [87] Hanley, J. A., & McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1), 29–36.
- [88] Boyd, K., Eng, K. H., & Page, C. D. (2013). Area under the precision-recall curve: Point estimates and confidence intervals (H. Blockeel, K. Kersting, S. Nijssen, & F. Železný, Eds.). In H. Blockeel, K. Kersting, S. Nijssen, & F. Železný (Eds.), *Machine learning and knowledge discovery in databases*, Berlin, Heidelberg, Springer Berlin Heidelberg.
- [89] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444. <https://doi.org/10.1038/nature14539>
- [90] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4), 115–133. <https://doi.org/10.1007/BF02478259>
- [91] Rosenblatt, F. (1957). *The Perceptron - A Perceiving and Recognizing Automaton* (tech. rep.). Buffalo, NY, Cornell Aeronautical Laboratory. <https://doi.org/85-460-1>

- [92] Tacchino, F., Macchiavello, C., Gerace, D., & Bajoni, D. (2019). An artificial neuron implemented on an actual quantum processor. *npj Quantum Information*, 5(1), arXiv 1811.02266. <https://doi.org/10.1038/s41534-019-0140-4>
- [93] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088), 533–536. <https://doi.org/10.1038/323533a0>
- [94] Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7), 1527–1554. <https://doi.org/10.1162/neco.2006.18.7.1527>
- [95] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6), 84–90. <https://doi.org/10.1145/3065386>
- [96] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- [97] Ruder, S. (n.d.). *An overview of gradient descent optimization algorithms* \* (tech. rep.). <http://caffe.berkeleyvision.org/tutorial/solver.html>
- [98] Zhang, J. (2019). Gradient Descent based Optimization Algorithms for Deep Learning Models Training, arXiv 1903.03614. <http://arxiv.org/abs/1903.03614>
- [99] LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., & Jackel, L. (1990). Handwritten digit recognition with a back-propagation network, 396–404.
- [100] Jin, R., Lu, L., Lee, J., & Usman, A. (2019). Multi-representational convolutional neural networks for text classification. *Computational Intelligence*, 35(3), 599–609. <https://doi.org/10.1111/coin.12225>
- [101] Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324. <https://doi.org/10.1109/5.726791>
- [102] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks (F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger, Eds.). In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 25*. Curran Associates, Inc. <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
- [103] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). Going deeper with convolutions. *CoRR*, abs/1409.4842arXiv 1409.4842. <http://arxiv.org/abs/1409.4842>
- [104] Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.
- [105] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385arXiv 1512.03385. <http://arxiv.org/abs/1512.03385>



- [106] Zhou, P., Qi, Z., Zheng, S., Xu, J., Bao, H., & Xu, B. (2016). Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling, In *Coling 2016 - 26th international conference on computational linguistics, proceedings of coling 2016: Technical papers*.
- [107] Jacovi, A., Sar Shalom, O., & Goldberg, Y. (2019). Understanding Convolutional Neural Networks for Text Classification. <https://doi.org/10.18653/v1/w18-5408>
- [108] Yan, Y., Yin, X. C., Yang, C., Li, S., & Zhang, B. W. (2018). Biomedical literature classification with a cnns-based hybrid learning network. *PLoS ONE*, 13(7). <https://doi.org/10.1371/journal.pone.0197933>
- [109] Abdulkadhar, S., Murugesan, G., & Natarajan, J. (2017). Recurrent convolution neural networks for classification of protein-protein interaction articles from biomedical literature, In *Proceedings - 2017 3rd IEEE international conference on research in computational intelligence and communication networks, icrcicn 2017*, Institute of Electrical; Electronics Engineers Inc. <https://doi.org/10.1109/ICRCICN.2017.8234505>
- [110] Xu, H., Kotov, A., Dong, M., Carcone, A. I., Zhu, D., & Naar-King, S. (2016). Text classification with topic-based word embedding and Convolutional Neural Networks, In *Acm-bcb 2016 - 7th acm conference on bioinformatics, computational biology, and health informatics*, New York, New York, USA, ACM Press. <https://doi.org/10.1145/2975167.2975176>
- [111] Graves, A. (2012). Supervised sequence labelling, In *Supervised sequence labelling with recurrent neural networks*. Springer.
- [112] Williams, R. J., & Zipser, D. (1995). Gradient-based learning algorithms for recurrent. *Backpropagation: Theory, architectures, and applications*, 433.
- [113] Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J. Et al. (2001). Gradient flow in recurrent nets: The difficulty of learning long-term dependencies. A field guide to dynamical recurrent neural networks. IEEE Press.
- [114] Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2), 157–166. <https://doi.org/10.1109/72.279181>
- [115] Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11), 2673–2681.
- [116] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>
- [117] Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with lstm.
- [118] Gers, F. A., Schraudolph, N. N., & Schmidhuber, J. (2002). Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(8), 115–143.

- [119] Graves, A., Mohamed, A. R., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks, In *Icassp, ieee international conference on acoustics, speech and signal processing - proceedings*. <https://doi.org/10.1109/ICASSP.2013.6638947>
- [120] Pascanu, R., Gulcehre, C., Cho, K., & Bengio, Y. (2014). How to construct deep recurrent neural networks, In *2nd international conference on learning representations, iclr 2014 - conference track proceedings*, International Conference on Learning Representations, ICLR.
- [121] Graves, A., & Schmidhuber, J. (2005a). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6), 602–610.
- [122] Graves, A., & Schmidhuber, J. (2005b). Framewise phoneme classification with bidirectional lstm networks, In *Proceedings. 2005 ieee international joint conference on neural networks, 2005*. IEEE.
- [123] Zhou, P., Shi, W., Tian, J., Qi, Z., Li, B., Hao, H., & Xu, B. (2016). Attention-based bidirectional long short-term memory networks for relation classification, In *54th annual meeting of the association for computational linguistics, acl 2016 - short papers*. <https://doi.org/10.18653/v1/p16-2034>
- [124] Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014b). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.
- [125] Ravanelli, M., Brakel, P., Omologo, M., & Bengio, Y. (n.d.). *Improving speech recognition by revising gated recurrent units* (tech. rep.).
- [126] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.
- [127] Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An empirical exploration of recurrent network architectures, In *International conference on machine learning*.
- [128] Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks, In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*.
- [129] Poon, H. K., Yap, W. S., Tee, Y. K., Lee, W. K., & Goi, B. M. (2019). Hierarchical gated recurrent neural network with adversarial and virtual adversarial training on text classification. *Neural Networks*, 119, 299–312. <https://doi.org/10.1016/j.neunet.2019.08.017>
- [130] Zhou, G. B., Wu, J., Zhang, C. L., & Zhou, Z. H. (2016). Minimal gated unit for recurrent neural networks. *International Journal of Automation and Computing*, 13(3), arXiv 1603.09420, 226–234. <https://doi.org/10.1007/s11633-016-1006-2>
- [131] Bahdanau, D., Cho, K. H., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, arXiv 1409.0473, 1–15.

- [132] Aurélien Géron. (2019). *Hands-on machine learning with Scikit-Learn, Keras and Tensor-Flow: concepts, tools, and techniques to build intelligent systems*. <https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/>
- [133] Galassi, A., Lippi, M., & Torroni, P. (2020). Attention in Natural Language Processing. *IEEE Transactions on Neural Networks and Learning Systems*, arXiv 1902.02181, 1–18. <https://doi.org/10.1109/tnnls.2020.3019893>
- [134] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems, 2017-Decem(Nips)*, arXiv arXiv:1706.03762v5, 5999–6009.
- [135] Sarkar, D., Bali, R., & Ghosh, T. (2018). *Hands-on transfer learning with python: Implement advanced deep learning and neural network models using tensorflow and keras*. Packt Publishing.
- [136] Yu, T., & Zhu, H. (2020). Hyper-parameter optimization: A review of algorithms and applications.
- [137] Chen, Q., Chandrasekarasastri, N., Elangovan, A., Davis, M., & Verspoor, K. (2017). Document triage and relation extraction for protein-protein interactions affected by mutations.
- [138] Seymour, E., Damle, R., Sette, A., & Peters, B. (2011). Cost sensitive hierarchical document classification to triage PubMed abstracts for manual curation. *BMC Bioinformatics*, 12(1). <https://doi.org/10.1186/1471-2105-12-482>
- [139] Genkin, A., Lewis, D. D., & Madigan, D. (2007). Large-scale bayesian logistic regression for text categorization. *Technometrics*, 49(3), 291–304. <https://doi.org/10.1198/004017007000000245>
- [140] Sang-Bum Kim, Kyoung-Soo Han, Hae-Chang Rim, & Sung Hyon Myaeng. (2006). Some effective techniques for naive bayes text classification. *IEEE Transactions on Knowledge and Data Engineering*, 18(11), 1457–1466. <https://doi.org/10.1109/TKDE.2006.180>
- [141] Chen, K., Zhang, Z., Long, J., & Zhang, H. (2016). Turning from TF-IDF to TF-IGM for term weighting in text classification. *Expert Systems with Applications*, 66, 1339–1351. <https://doi.org/10.1016/j.eswa.2016.09.009>
- [142] Kowsari, K., Heidarysafa, M., Brown, D. E., Meimandi, K. J., & Barnes, L. E. (2018). RMDL: Random multimodel deep learning for classification, In *Acm international conference proceeding series*. <https://doi.org/10.1145/3206098.3206111>
- [143] Dollah, R., Sheng, C. Y., Zakaria, N., Othman, M. S., & Rasib, A. W. (2019). Deep learning classification of biomedical text using convolutional neural network. *International Journal of Advanced Computer Science and Applications*, 10(8), 512–517. <https://doi.org/10.14569/ijacsa.2019.0100867>
- [144] Kowsari, K., Brown, D. E., Heidarysafa, M., Jafari Meimandi, K., Gerber, M. S., & Barnes, L. E. (2017). HDLTex: Hierarchical Deep Learning for Text Classification, In *Proceedings - 16th IEEE international conference on machine learning and applications, icmla 2017*,

- Institute of Electrical; Electronics Engineers Inc. <https://doi.org/10.1109/ICMLA.2017.0-134>
- [145] *Natural language toolkit*. (n.d.). <https://www.nltk.org/> (accessed: 27.10.2020)
- [146] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J. Et al. (2020). Array programming with numpy. *Nature*, 585(7825), 357–362.
- [147] McKinney, W., & Team, P. (2015). Pandas: Powerful python data analysis toolkit. *Pandas—Powerful Python Data Analysis Toolkit*, 1625.
- [148] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. Et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12, 2825–2830.
- [149] Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science Engineering*, 9(3), 90–95. <https://doi.org/10.1109/MCSE.2007.55>
- [150] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M. Et al. (2019). Huggingface’s transformers: State-of-the-art natural language processing. *ArXiv*, arXiv–1910.
- [151] Copperwaite, M., & Leifer, C. (2015). *Learning flask framework*. Packt Publishing Ltd.
- [152] Islamaj Doğan, R., Kim, S., Chatr-aryamontri, A., Wei, C.-H., Comeau, D. C., Antunes, R., Matos, S., Chen, Q., Elangovan, A., Panyam, N. C., Verspoor, K., Liu, H., Wang, Y., Liu, Z., Altinel, B., Hüsünbeyi, Z. M., Özgür, A., Fergadis, A., Wang, C.-K., . . . Lu, Z. (2019). Overview of the BioCreative VI Precision Medicine Track: mining protein interactions and mutations for precision medicine [bay147]. *Database*, 2019. <https://doi.org/10.1093/database/bay147>
- [153] Sun, C., Qiu, X., Xu, Y., & Huang, X. (2019). How to fine-tune BERT for text classification? *CoRR*, abs/1905.05583arXiv 1905.05583. <http://arxiv.org/abs/1905.05583>
- [154] Cohan, A., Feldman, S., Beltagy, I., Downey, D., & Weld, D. S. (2020). Specter: Document-level representation learning using citation-informed transformers.

## SUPPLEMENTARY MATERIAL

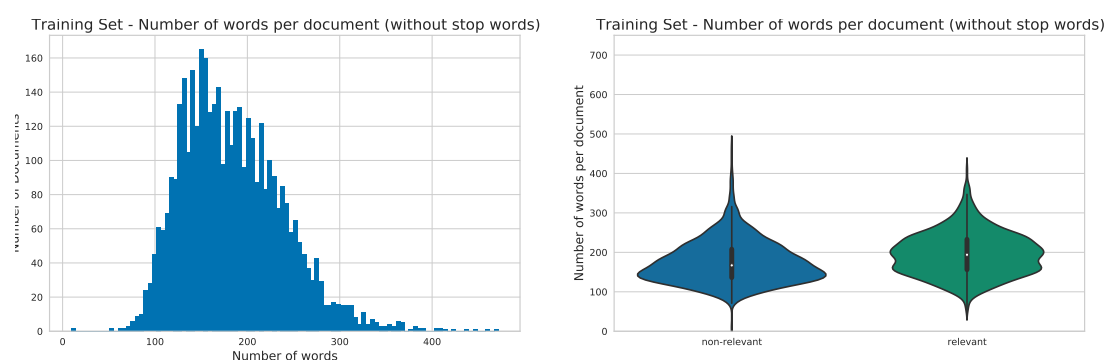


Fig. 24 – Number of words per document of the training set.

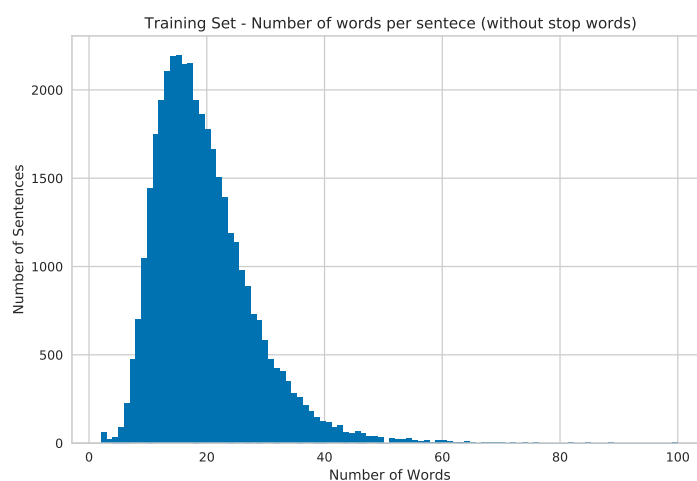


Fig. 25 – Number of Words per sentence of the training set.

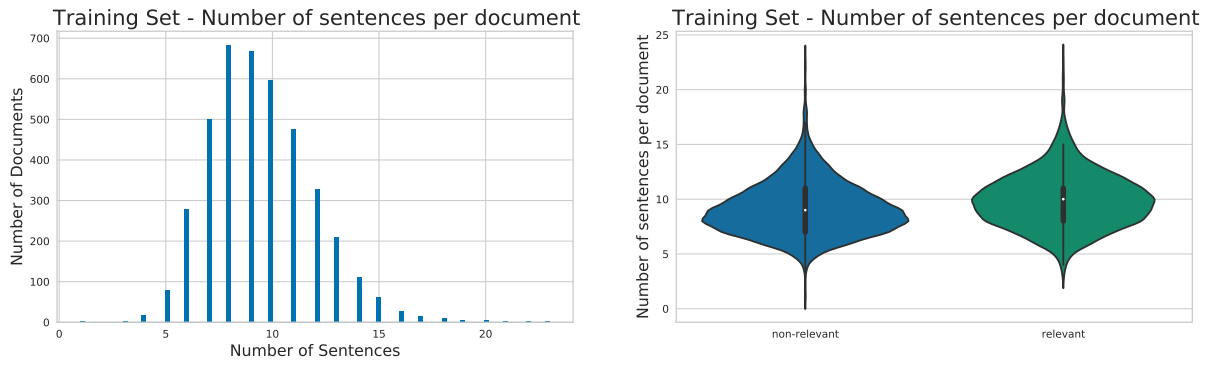


Fig. 26 – Number of sentences per document of the training set.

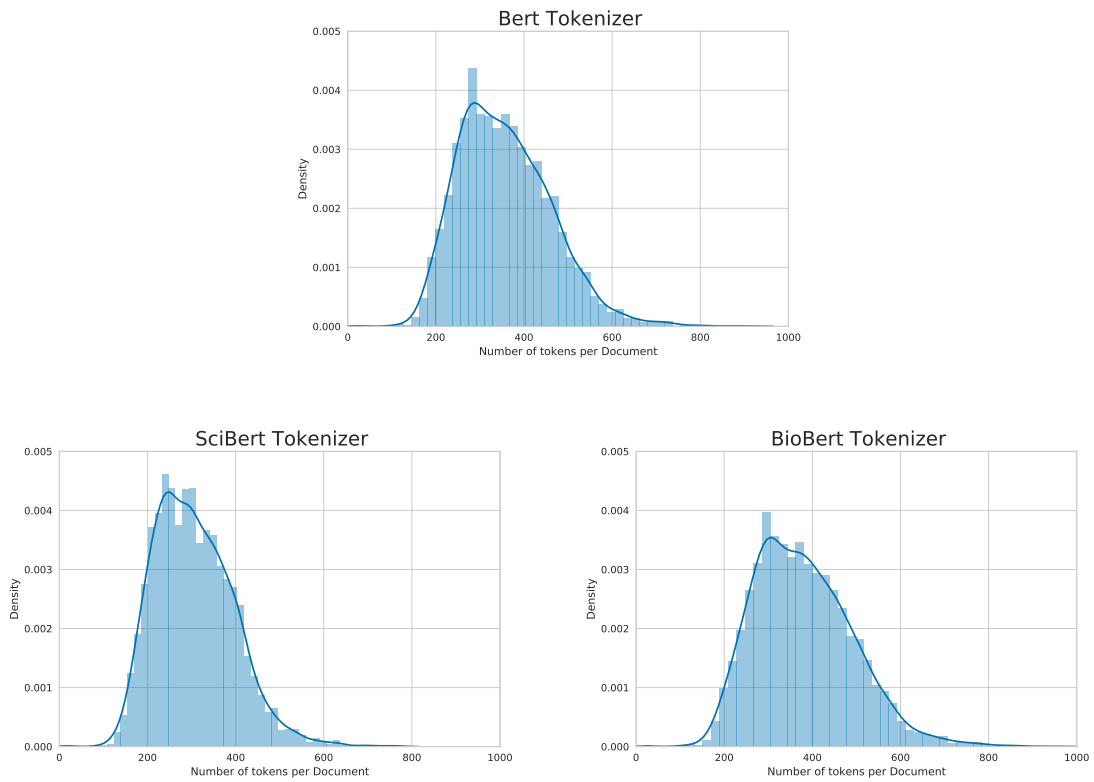


Fig. 27 – Number of tokens per document using different BERT Tokenizers.

### Training Set - Top 100 Words (without stopwords)

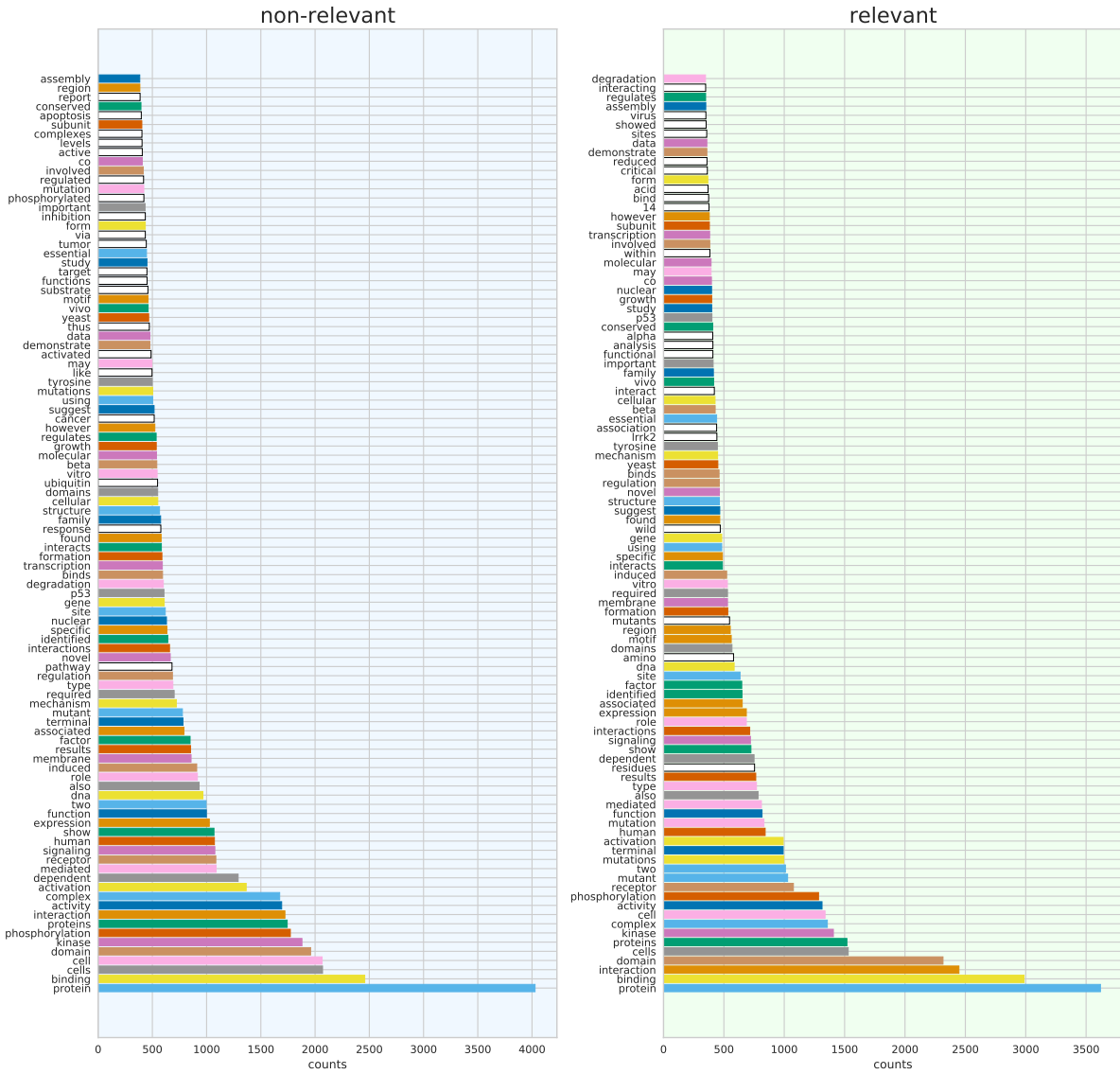


Fig. 28 – Top 100 words of Training set without stop words.

Test Set - Top 100 Words (without stopwords)

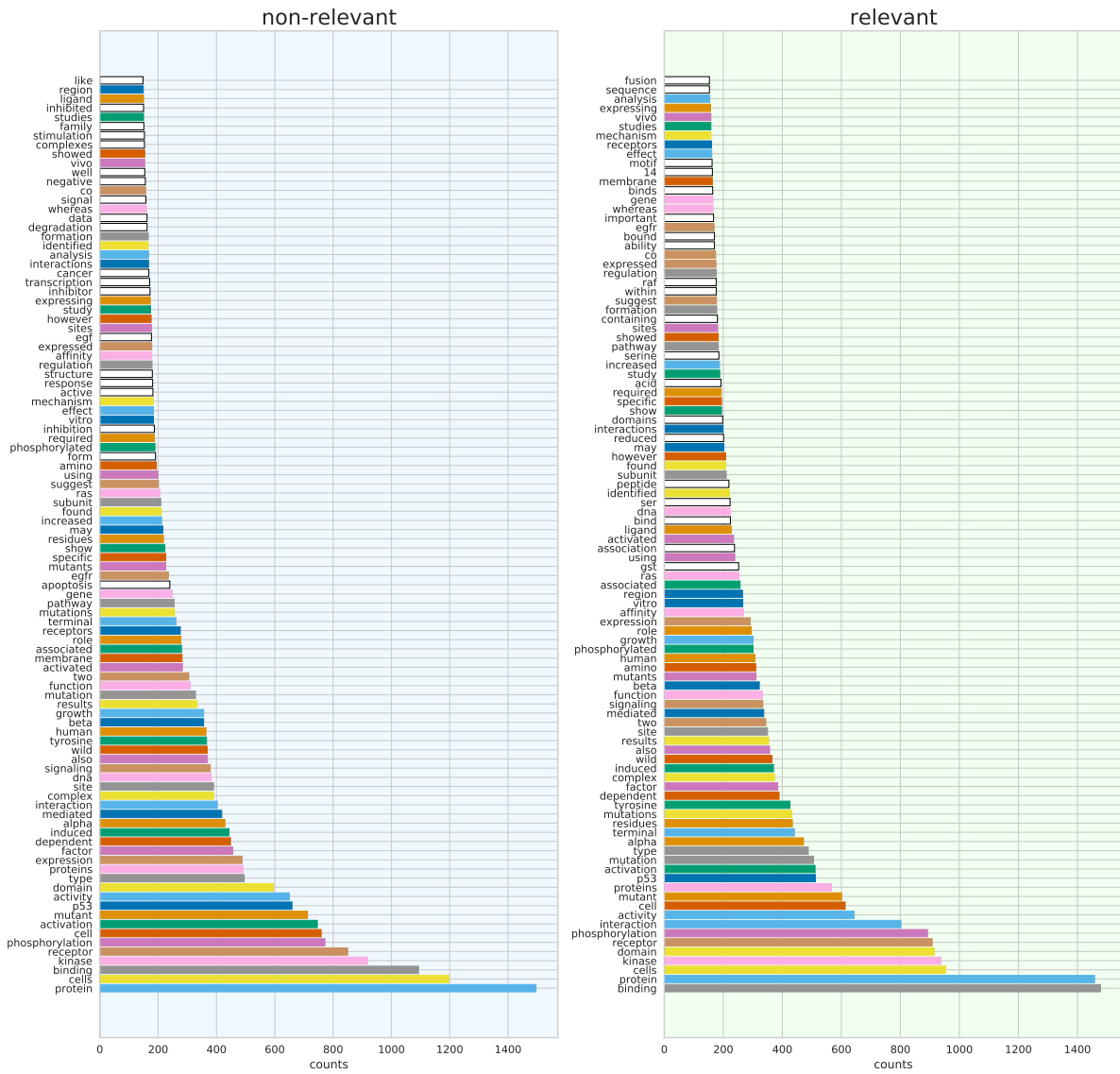


Fig. 29 – Top 100 words of Test set without stop words.



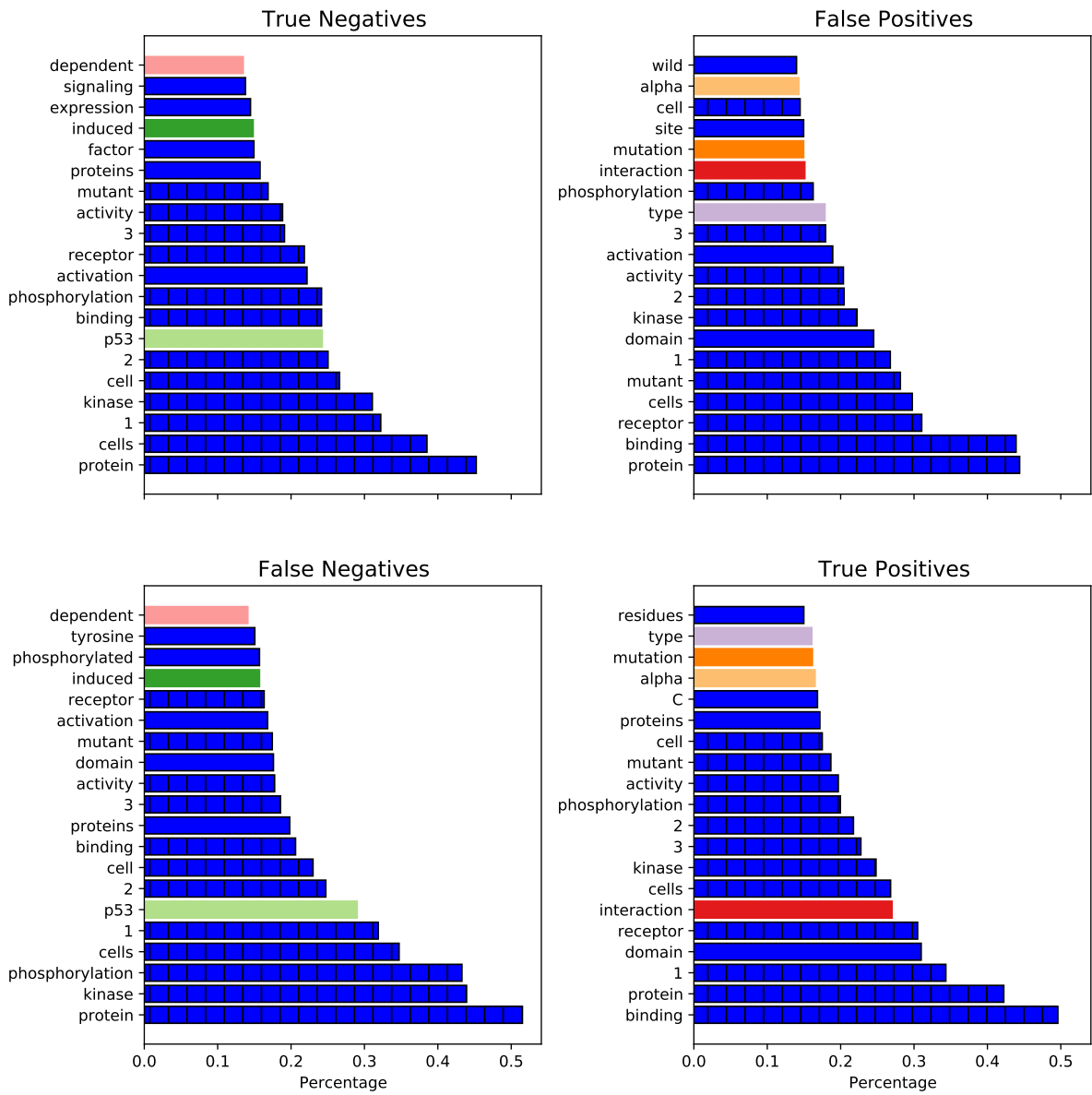


Fig. 30 – Top 20 number of words present on subsets of the test set based on our best model predictions.