Universidade do Minho
Escola de Engenharia
Departamento de Informática

Tiago Rafael Ferreira Miranda da Silva

Development of a recommendation system for scientific literature based on deep learning

October 2022

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Tiago Rafael Ferreira Miranda da Silva

Development of a recommendation system for scientific literature based on deep learning

Master dissertation
Master Degree in Bioinformatics

Dissertation supervised by
Miguel Francisco Almeida Pereira da Rocha
Vítor Manuel Sá Pereira

October 2022

ACKNOWLEDGEMENTS

Firstly, I would like to appreciate Dr. Miguel Rocha and Dr. Vítor Pereira for the opportunity of working with them. All the expertise and guidance were highly motivating and this thesis would not have been possible without them. It was their help and motivation that grew my interest in the Machine and Deep Learning fields.

Secondly, I would like to thank the opportunity given by OmniumAI for allowing me to develop this thesis in collaboration with the company. Also, all the collaborators at OmniumAI for their help and knowledge provided in this collaboration, with a special acknowledgment to Fernando Cruz for his help with the work performed in the company. A special thanks to Nuno Alves, a PhD student and collaborator at OmniumAI, for his tireless guidance, availability, knowledge, and experience. Without him, this thesis would not have been possible.

Additionally, I would like to express my most sincere gratitude to my family for their support all my life, especially to my parents. It was their encouragement and support that allowed me to reach this stage in my studies and in my life. For that, my deep gratitude.

Finally, I would like to thank all the friends I made on this path. From the beginning of my studies right to this point. To the friends I have kept in contact with since first grade. To all the amazing people that I have met while studying at the University of Minho, with a special thanks to Miguel Barros, Mónica Fernandes, and Tiago Machado for the amazing adventure it was working with them. A special thanks to my girlfriend, Mónica Fernandes for all the emotional support and company over the past year.

Again, to all the people that supported me in this journey: Thank You!

STATEMENT OF INTEGRITY

ABSTRACT

The previous few decades have seen an enormous volume of articles from the scientific community on the most diverse biomedical topics, making it extremely challenging for researchers to find relevant information. Methods like Machine Learning (ML) and Deep Learning (DL) have been used to create tools that can speed up this process. In that context, this work focuses on examining the performance of different ML and DL techniques when classifying biomedical documents, mainly regarding their relevance to given topics. To evaluate the different techniques, the dataset from the BioCreative VI Track 4 challenge was used. The objective of the challenge was to identify documents related to protein-protein interactions altered by mutations, a topic extremely important in precision medicine. Protein-protein interactions play a crucial role in the cellular mechanisms of all living organisms, and mutations in these interaction sites could be indicative of diseases.

To handle the data to be used in training, some text processing methods were implemented in the Omnia package from OmniumAI, the host company of this work. Several preprocessing and feature extraction methods were implemented, such as removing stopwords and TF-IDF, which may be used in other case studies. They can be used either with generic text or biomedical text. These methods, in conjunction with ML pipelines already developed by the Omnia team, allowed the training of several traditional ML models.

We were able to achieve a small improvement on performance, compared to the challenge baseline, when applying these traditional ML models on the same dataset. Regarding DL, testing with a CNN model, it was clear that the BioWordVec pre-trained embedding achieved the best performance of all pre-trained embeddings. Additionally, we explored the application of more complex DL models. These models achieved a better performance than the best challenge submission. BioLinkBERT managed an improvement of 0.4 percent points on precision, 4.9 percent points on recall, and 2.2 percent points on F1.

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

RESUMO

As décadas anteriores assistiram a um enorme aumento no volume de artigos da comunidade científica sobre os mais diversos tópicos biomédicos, tornando extremamente difícil para os investigadores encontrar informação relevante. Métodos como Aprendizagem Máquina (AM) e Aprendizagem Profunda (AP) têm sido utilizados para criar ferramentas que podem acelerar este processo. Neste contexto, este trabalho centra-se na avaliação do desempenho de diferentes técnicas de AM e AP na classificação de documentos biomédicos, principalmente no que diz respeito à sua relevância para determinados tópicos. Para avaliar as diferentes técnicas, foi utilizado o conjunto de dados do desafio BioCreative VI Track 4. O objectivo do desafio era identificar documentos relacionados com as interacções proteína-proteína alteradas por mutações, um tópico extremamente importante na medicina de precisão. As interacções proteína-proteína desempenham um papel crucial nos mecanismos celulares de todos os organismos vivos, e as mutações nestes locais de interacção podem ser indicativas de doenças.

Para tratar os dados a utilizar no treino, alguns métodos de processamento de texto foram implementados no pacote Omnia da OmniumAI, a empresa anfitriã deste trabalho. Foram implementados vários métodos de pré-processamento e extracção de características, tais como a remoção de palavras irrelevantes e TF-IDF, que podem ser utilizados em outros casos de estudos, tanto com texto genérico quer com texto biomédico. Estes métodos, em conjunto com as *pipelines* de AM já desenvolvidas pela equipa da Omnia, permitiram o treino de vários modelos tradicionais de AM.

Conseguimos alcançar uma pequena melhoria no desempenho, em comparação com a linha de referência do desafio, ao aplicar estes modelos tradicionais de AM no mesmo conjunto de dados. Relativamente à AP, testando com um modelo CNN, ficou claro que o *embedding* pré-treinado BioWordVec alcançou o melhor desempenho de todos os *embeddings* pré-treinados. Adicionalmente, explorámos a aplicação de modelos de AP mais complexos. Estes modelos alcançaram um melhor desempenho do que a melhor submissão do desafio. BioLinkBERT conseguiu uma melhoria de 0,4 pontos percentuais na precisão, 4,9 pontos percentuais no *recall*, e 2,2 pontos percentuais em F1.

Palavras-Chave: Aprendizagem Profunda; Classificação de Documentos; Aprendizagem Máquina; Mineração de Texto Biomédico; Mineração de Texto.

CONTENTS

ACRONYMS

AI	Artificial Intelligence
ANN	Artificial Neural Networks
API	Application Programming Interface
AUC	Area Under Curve
AutoML	Automated Machine Learning
BERT	Bidirectional Encoder Representations from Transformers
BioTM	Biomedical Text Mining
BPTT	Back Propagation Through Time
CBOW	Continuous Bag-Of-Words
CNN	Convolutional Neural Networks
DL	Deep Learning
DNN	Dense Neural Network
FN	False Negatives
FP	False Positives
FPR	False Positive Rate
GPU	Graphics Processing Unit
GRU	Gated Recurrent Units
HPO	Hyperparameter Optimization
KNN	K-Nearest Neighbors
LC-MS	Liquid-Chromatography Mass Spectrometry
LGBM	Light Gradient Boosting Machine
LogR	Logistic Regression

LSTM	Long Short-Term Memory
MCC	Mathews Correlation Coefficient
MeSH	Medical Subject Headings
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NER	Named Entity Recognition
NLP	Natural Language Processing
NN	Neural Network
PCA	Principal Component Analysis
PPI	Protein-Protein Interaction
PR	Precision-Recall
ReLu	Rectified Linear Unit
RF	Random Forest
RNN	Recurrent Neural Networks
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SNE	Stochastic Neighbor Embedding
SVM	Support Vector Machines
Tanh	Hyperbolic Tangent
TF-IDF	Term Frequency-Inverse Document Frequency
TLU	Threshold Logic Unit
TM	Text Mining
TN	True Negatives
TP	True Positives
t-SNE	t-distributed Stochastic Neighbor Embedding

XGBoost	eXtreme Gradient Boosting
XT	Extra Trees

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Context and Motivation

Over the years, and as technology grew, science produced and still keeps producing large amounts of data. A significant part of these data is condensed in the form of articles. Following this trend, the number of published literature keeps getting bigger and bigger as the years pass [1]. Platforms such as PubMed, one of the most popular scientific literature databases, contains as many as 31 million published articles. Considering this number of documents, it would be deeply time-consuming for a researcher to read and analyze all the information about a given topic. To help with this nearly impossible task, Text Mining (TM) technologies can be of help.

The main objective of TM is to discover relevant knowledge within textual data, a process that can be achieved using information retrieval, information extraction and document classification tasks [2, 3]. Applying TM on the biomedical field produced a new sub-field of text mining: Biomedical Text Mining (BioTM).

Document classification tasks are an important part of TM that aims to classify documents according to their content. This classification can have two alternatives: binary, in which documents are labelled as one of two labels, and multi-class, in which the documents are classified as one of the available three or more classes in question. As an example of a binary task, documents are usually classified between relevant and non-relevant, relative to the topic at hand. These tasks are usually performed using machine/deep learning [4, 5].

Indeed, Machine Learning (ML) is well established on document classification. Models such as Support Vector Machines (SVM), Naïve Bayes, and Decision Trees have already proven to be competent in these tasks [6, 7]. Recently, Deep Learning (DL) application to TM tasks has grown considerably. DL models, like Bidirectional Encoder Representations from Transformers (BERT) [8] and XLNET [9], have been proven to be quite effective in TM. Some BERT adaptations have been able to achieve good results and originated some variations like BioBERT [10], which was pre-trained with biomedical texts, thus obtaining better performance than BERT, and DocBERT [11], which originated from the need for a better performing model for document classification, when applied to BioTM tasks.

There are already some Python packages developed within the BioSystems research group at the University of Minho that perform text/document classification successfully. These packages, however, can be updated with newer deep learning models to improve their efficiency, which would open the possibility to develop a new platform.

1.2 Objectives

The main goal of this work is to develop ML and DL models to provide better literature recommendations, by improving its prediction abilities regarding document relevance tasks. This work will address the following objectives:

- Review important literature regarding text/document classification.
- Explore different ML and DL techniques for text/document classification, comparing their performance.
- Develop a platform around existing packages created within the host group and OmniumAI company for document relevance assessment.
- Apply the developed models and pipelines to a case study in document relevance.
- Writing the thesis exposing the results.

1.3 Text Organization

This dissertation is organized in 5 chapters. A concise description of the chapters follows:

- **Chapter 2** focuses on describing the BioTM field and Natural Language Processing (NLP). It also describes useful ML techniques, both supervised and unsupervised, focusing on

DL. Additionally, it introduces some techniques such as transfer learning, Hyperparameter Optimization (HPO), and Automated Machine Learning (AutoML).

- **Chapter 3** will focus on exposing the methodology used in this work. It exposes the work performed under OmniumAI, the ML models trained, and parameters used.
- **Chapter 4** will expose and discuss the results obtained in each step of this work. It explains the dataset used to train the ML models, it discusses the results obtained with traditional ML models, the performance of different types of pre-trained embeddings, the difference between training biomedical and generic models, and the effects of performing AutoML.
- **Chapter 5** depicts some final conclusions of this project coupled with future work.

2

STATE OF THE ART

2.1 Biomedical Text Mining

Over the last years, the number of published scientific literature has been growing at an incredible rate. PubMed alone has more than 31 million articles on its database, with 3,300 million searches on its platform in 2020[1]. All this knowledge is easily available and covers a vast amount of topics. However, finding the information pursued by end users is still an arduous task. The huge amount of available data compel researchers to spend a long time performing the tedious task of classifying articles according to their interest without any warranty of success. As such, computers took a crucial place in mitigating this problem given their ability to process large amounts of data quickly [12]. Consequently, researchers can find the information they need in a short period with a minimum effort. Even though computers can process data quickly, interpreting the article's content is not as easy of a job as for humans. Since these articles' texts are in an unstructured free-way, there was a need to find approaches to convert them, so that computer algorithms could understand them. The answer to this problem appeared as BioTM, a subfield of TM [3]. TM, a type of data mining, can be described as an aggregation of techniques with the main purpose of analyzing, processing, and discovering information within the unstructured text [3, 13].

Given the wide array of possible tasks to perform with TM, they often superimpose each other. As such, there is no clear way to divide TM into subfields. Zweigenbaum *et al.* [14] divided TM into four different tasks: Named Entity Recognition (NER), relation extraction, document summarization, and question answering. Later, Miner *et al.* [3] expanded this sub-division into seven fields:

1 Retrieved from: https://www.nlm.nih.gov/bsd/medline_pubmed_production_stats.html

- **Information Retrieval**: searching and retrieving documents from a database using keywords or queries. This technique is widely used by search engines.
- **Document Clustering**: grouping similar documents into clusters using clustering algorithms like k-means, a task that is part of unsupervised learning.
- **Document Classification**: classifying unlabeled documents using a model that learned using labelled ones. This technique is the main focus of this work, being part of supervised learning.
- **Web Mining**: differs from the rest because, usually, the text is presented in a structured way with hyperlinks between web pages.
- **Information Extraction**: extracts information from the text and possible correlation within that information. Includes NER and relation extraction.
- **Natural Language Processing**: combines computation with linguistics. Using computational tools to manipulate human language.
- **Concept Extraction**: identifying concepts of interest from parts of the text.

Even though TM has been applied for a long time and is well studied, there were issues with biomedical text in general. Since these texts have a more precise and concise language than normal text and have been steadily increasing over the last years, it quickly became a challenge to surpass. That way, BioTM appeared.

BioTM is an application of TM on biomedical text to extract relevant biomedical information. Some examples of its application are protein-protein interactions, drug-target discovery, and gene expression annotation [1, 15, 16].

2.1.1 Document Classification

Document Classification is a TM task that focuses on classifying new text documents with a class from a set of pre-defined labels. This classification is attributed in accordance to the information built-in the document [17]. This task can take the form of a binary classification when there are only two labels, but it can also be multi-class when there are more than two labels or multi-label if each document can receive more than one label simultaneously. Document Classification can also be applied to document segments like paragraphs and sentences.

Document Classification can be employed in several ways. Hart *et al.*, as a means of data loss prevention, developed a ML algorithm to learn and classify documents as private or public [18]. Sulea *et al.* applied document classification to the legal domain to classify the law area to which a case belonged [19]. Kang *et al.* classified text based on the opinion behind the text, a field known as opinion mining [20].

The document classification problem is old. There always was a need to classify all the documents we create. The first solution was plain and simple: classifying manually. However, with the Internet's appearance and with its vast amounts of text and information, there was an urgency to improve traditional methods. Therefore, Artificial Intelligence (AI) was introduced in this process. In a first effort, ML algorithms were considered (Figure 1), facilitating the opportunity to classify a large number of documents quickly using a model that had been trained previously on a set of already classified documents. More recently, DL has been making its way into this field.

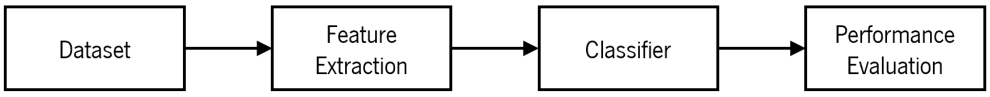

Fig. 1 – Generic document classification pipeline. Adapted from [21].

2.1.2 NLP/Text Preprocessing

NLP is a computer science subfield that handles human language. Hirschberg and Manning further define it as being concerned with using computational techniques to learn, understand, and produce human language content [22]. NLP has been gaining more and more notoriety because of some of its applications, for example, speech recognition, dialogue systems, information retrieval, question answering, machine reading, and machine translation [22–24].

Indurkhya and Damerau break down the process of natural language analysis into five steps: (1) tokenization, which is the task of segmenting text into words; (2) lexical analysis, which is the process of relating morphological variants of words; (3) syntactic analysis, the process of defining the structural description of a string of words (i.e., sentence) according to a formal grammar; (4) semantic analysis, is the process of understanding the contents of texts (e.g., information retrieval, information extraction, text summarization); (5) pragmatic analysis, is the process of understanding the context of words or sentences. The tokenization step is an essential part of the text preprocessing task in TM. Text preprocessing is the task responsible for processing the text for downstream machine applications. This task encompasses processes like tokenization, stopword removal, and vectorization [23].

Since computer algorithms do not understand words the same way we do, all the text needs to be converted into numeric vectors. This process is called vectorization. Term Frequency-Inverse Document Frequency (TF-IDF) is often used as a vectorization technique. TF-IDF gives weight to the words of a document based on the frequency by which that word appears on the document (TF), inversely proportional to its frequency in the whole corpus (IDF). It is given by the following expression:

$$\text{TF-IDF}_{w,d} = \frac{\text{Number of } w \text{ in } d}{\text{Number of words in } d} \times log \frac{N}{\text{Number of } d \text{ where } w \text{ appears}} \tag{1}$$

where w is the word in question, d is the document and N is the size of the corpus.

TF-IDF follows two main ideas: the more often a word exists in a document, the more it represents the content of the text; the more documents the word appears in, the less informative it is [25, 26]. Basically, it means that a word is more meaningful if it appears more times in a document and fewer times on the whole dataset. Another vectorization technique is one-hot encoding. It consists of attributing a unique index to each word and then transforming that index into a vector with the size of the vocabulary. Each vector is filled with zeros except for the index of that word, which becomes a one [27].

However, before the vectorization, other preprocessing techniques are usually applied, like tokenization, stopword removal, lowercase or capitalization, punctuation removal, stemming, lemmatization, and *n*-grams. The following description of these techniques was adapted from Sarkar [17], Allahyari *et al.* [5], and Kowsari *et al.* [6]:

- **Tokenization**: breaking text into words/phrases called tokens.
- **Stopword removal**: removing certain words that appear on the text. Stopwords are frequent words that appear in texts that usually present little to no meaning to the topic, words such as “a”, “and”, “the”, “of” and “what”.
- **Lowercase or Capitalization**: converting all words to their lower or uppercase format. Since some programming languages are case sensitive, this task ensures that the same word, but with different capitalization, is counted as the same word.
- **Punctuation removal**: removing all punctuation from the text.
- **Stemming**: obtaining the root (stem) of words. Stemming is a way of aggregating different forms of words into a single one. However, the stem does not need to be a word. For example, the stem of the words “argues” and “arguing” is “argu”.
- **Lemmatization**: obtaining the basic word form (lemma). Lemmatization considers the morphology of the word. It is very similar to stemming as it gets the root of the word, however, the lemma needs to be a word. Using the same example as in stemming, the lemma of the words “argues” and “arguing” is “argue”.
- ***N*-grams**: technique that groups *n*-words into a token. The word tokenization task described before is an example of a 1-gram.

Following these steps allows us to make text intelligible and workable with machine and deep learning algorithms, which is nowadays the foundation of TM.

2.1.3 Word Embeddings

Word vectors or word embeddings are an alternative to the vectorization processes explained in the previous subchapter. Word embeddings are dense, low-dimensional vectors that represent words. These embeddings capture semantic and syntactic information of those words, and their relationship is characterized by a vector offset [27–29]. This means that the vectors of closely related words, like *man* and *woman*, would be very close to each other. For example, the subtraction of the vector for *Man* to the vector for *King* and then sum the vector for *Woman* results in a vector roughly similar to the vector for *Queen* [29, 30].

Word embeddings can be trained using texts from given corpora or used in a pre-trained format by including them into the DL algorithms. Usually, pre-trained embeddings are more utilized since they require no prior computation, also, since they were trained on massive datasets, reaching billion of words, they offer good representations. However, they have the disadvantage of keeping the context of the subject on which they were firstly trained. For example, a word embedding model trained from the biomedical domain would be better used on biomedical texts [31]. A list of some of the existing word embeddings are shown in table 1.

Table 1 – List of embeddings. Medical Subject Headings (MeSH) is a vocabulary that gives uniformity and consistency to the indexing and cataloging of biomedical literature

Embeddings	Year	Description
Word2vec [30]	2013	Word embeddings learned from CBOW and Skip-gram models
GloVe [32]	2014	Embeddings based on aggregated global word-word co-occurrence statistics
fastText [33]	2018	Embeddings trained with focus on unseen words
ELMo [34]	2018	Embeddings are functions of the entire input sentence
BERT [8]	2018	Contextualized word representation model based on bidirectional transformers
BioWordVec [35]	2019	Based on fastText. Combines subword information from biomedical text together with the MeSH vocabulary
GPT-2 [36]	2019	Transformer-based language model with 1.5 billion parameters and focus on predicting the next word based on all previous words
SciBERT [37]	2019	BERT applied on documents from the Semantic Scholar[2] database
BioBERT [10]	2020	BERT applied on documents from the PubMed[3] and PMC[4] databases

2.1.3.1 *Word2vec*

Word2vec is a group of neural network-based models which are applied to generate word embeddings. These models are Continuous Bag-Of-Words (CBOW) and Continuous Skip-gram. Even though these models are similar, the way they function is different. While CBOW tries to

2 https://www.semanticscholar.org/
3 https://pubmed.ncbi.nlm.nih.gov/
4 https://www.ncbi.nlm.nih.gov/pmc/

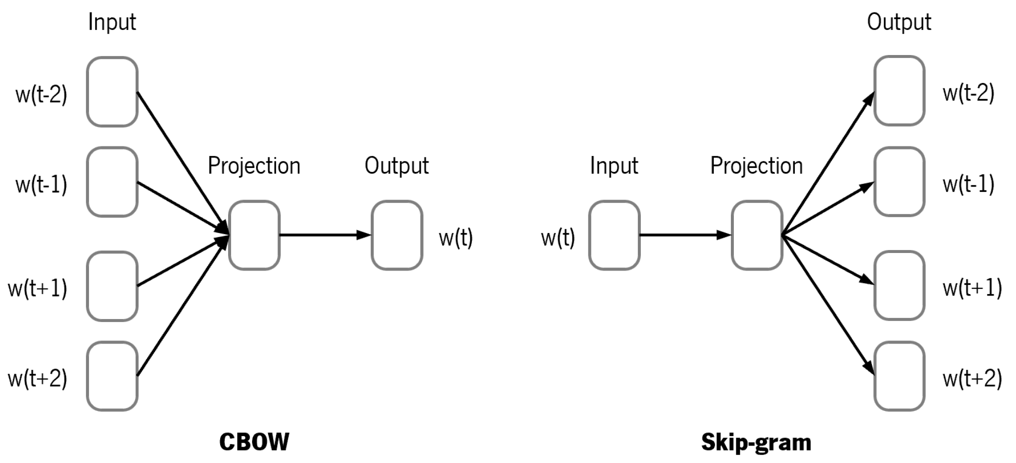

Fig. 2 – CBOW and skip-gram model's architectures. CBOW predicts the current word (w(t)) using adjacent words (w(t-1), w(t-2), w(t+1), w(t+2)), while skip-gram predicts the adjacent words using the current words. Adapted from [30].

predict a word based on its context (or adjacent words), skip-gram uses that word to predict its adjacent words (Figure 2) [30].

2.1.3.2 *GloVe*

Like Word2vec, GloVe also uses local context windows methods to train the model. However, it also applies global matrix factorization methods that improve global statistical information. These methods, combined with a specific weighted least squares model, are trained on global word-word co-occurrence counts, which results in a word vector space with a meaningful sub-structure [32].

2.1.3.3 *BERT*

BERT is a contextualized word representation model. It was pre-trained with the two objectives of predicting masked tokens in texts and determining if one text passage is likely to follow another. These are known as masked language modeling and next sentence prediction, respectively. The goal of next sentence prediction is for the model to predict from the previous sentence whether a given sentence was the successive sentence. The masked language model randomly masks (hides) some of the words from the input text, and its objective is to predict the actual word based only on its context. This allows the model to fuse the left and right context, which in turn enables bidirectional training. As a result of the whole pre-training process, the model is able to learn contextual embeddings for words [8].

2.2 Machine Learning

ML is a subfield of AI that first appeared in the 1950s with the objective to automate intellectual tasks usually performed by humans. In a sense, AI serves as a human substitute in some tasks. This field embraces ML and DL (Figure 3).

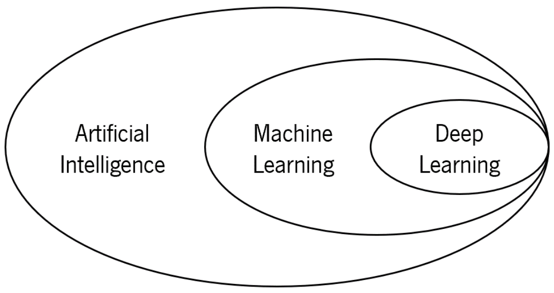

Fig. 3 – AI and its subfields. Adapted from [27].

The main objective of ML is to learn from data by extracting relationships between variables, whether, for example, by recognizing patterns or a set of rules from the given data so that it can produce a result when new data is given. These ML systems are provided with examples that the algorithm learns from, and then it tries to find statistical structure from these examples so that it can produce a set of rules that best automate the task. This means that a ML algorithm needs to be trained in order to produce results. ML methods can be divided into four separate types: Supervised Learning, Unsupervised Learning, Self-supervised Learning, and Reinforcement Learning [27]. Supervised learning consists of learning patterns on the data while knowing the labels of the data. In contrast, unsupervised learning does not have labelled data, so it is restricted to clustering, data visualization, and dimensionality reduction. Self-supervised learning is a specific instance of supervised learning, but it is different enough to have its own category. It is similar to supervised learning because it still uses labels in the training process, but these labels are generated from the unlabeled input data [27, 38, 39]. Reinforcement Learning algorithms map all possible actions in an environment with the intent to find the best possible one. Each action will provide a different reward to the algorithm, so the main objective is to find the actions that yield the most reward [27, 40].

2.2.1 Unsupervised Learning

As exposed before, unsupervised learning techniques use unlabeled data as input to discover hidden structures or data groupings in the training data. Since the data are unlabeled, there is no direct way to evaluate the performance of the algorithms. Therefore, these techniques are more employed as a data exploration method. These techniques can be divided into three categories: clustering (e.g., *k*-means), dimension reduction (e.g., PCA), and data visualization (e.g., t-SNE). These techniques are very useful in text mining as there are, usually, a large number of features. This way, unsupervised learning has become a useful tool in TM [6, 39].

2.2.1.1 *k-means*

The term "*k*-means" was first used by MacQueen in 1967 [41]. However, the algorithm behind it was introduced a full decade before in 1957, as stated by Lloyd [42]. MacQueen described it as a process for partitioning an *N*-dimensional population into *k* sets on the basis of a sample [41]. In simpler words, this algorithm tries to attribute each sample into a cluster to arrange the data. The clustering procedure has three general steps. First, the algorithm chooses *k* cluster centroids at random. Second, it appoints each sample to the closest centroid. Third, it recalculates the centroids with the samples from the previous step. After these three initial steps, the algorithm enters a loop where it goes back to the second step to reassign the samples with new clusters. This loop continues until the cluster centroids do not change between iterations of the loop [43]. Regarding the second step, each sample is assigned to a cluster based on its distance to the centroids. This distance is often calculated using the euclidean distance, and the cluster is chosen based on the smallest distance between the sample and all the cluster centroids [44].

2.2.1.2 *Principal Component Analysis*

An early form of Principal Component Analysis (PCA) was first described by Pearson in 1901 [45]. Later, Hotelling developed it even further, even claiming the term “principal component”, which would become the standard for PCA [46, 47]. The main goal of PCA is to reduce the dimensionality of data, while retaining most of the variation in the dataset [48]. This technique is often used as a first step, reducing dimensionality before performing supervised learning tasks [6, 49]. PCA works by creating new uncorrelated variables, named principal components, obtained from the singular value decomposition of the original data, which explain correlations between the original variables. These components are ordered so that the first principal component is required to explain the most possible variance in the data [47, 50].

2.2.1.3 *t-distributed Stochastic Neighbor Embedding*

t-distributed Stochastic Neighbor Embedding (t-SNE) is a variation of an older technique called Stochastic Neighbor Embedding (SNE) and is widely used as a data visualization technique, but it can also be used in dimensionality reduction. The main goal of SNE is to represent data in a low-dimensional space so that it preserves neighbour identities. SNE works by computing the euclidean distances between data points and then converting them into conditional probabilities representing similarities. The conditional probabilities are calculated using a Gaussian distribution [51–53]. The main difference t-SNE has from SNE is that it uses a Student-t distribution rather than a Gaussian distribution to compute the similarity between the points in the low-dimensional space. This solves the main problem of SNE, which Maaten and Hinton described as a “crowding problem” [51, 52].

2.2.2 Supervised Learning

As opposed to unsupervised learning, supervised learning uses labelled data to train the algorithms (Figure 4). That way, each training sample is a pair of values composed of input and output [39]. After the training is performed, a new set of data can be supplied to the model to predict the output for each sample. The type of prediction is different according to the type of output expected. If the output variables are continuous (e.g., predicting house prices), the model is a regression model, and if the output is discrete variables (e.g., classifying documents), the model is a classification model. The latter, is the main focus of this work. Some of the classification models used in Document Classification are Decision Trees, Naïve Bayes, SVM, Logistic Regression (LogR), K-Nearest Neighbors (KNN), Random Forest (RF), and Neural Network (NN) (further discussed in section 2.3) [3, 7, 17].

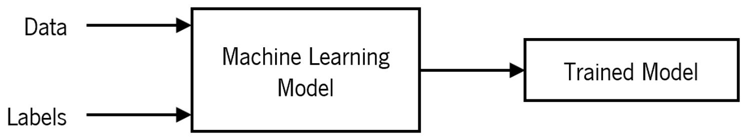

Fig. 4 – Supervised machine learning structure. Adapted from [27].

2.2.3 Evaluation Metrics

Supervised learning algorithms are evaluated by comparing the predicted labels to the real ones. This comparison produces a Confusion Matrix table (Table 2) which is a representation of the number of cases for True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). In a binary Document Classification task, the positive label corresponds to the relevant documents, and the negative one corresponds to the non-relevant documents. In this case, a TP case is when the model correctly predicts a document as being relevant [54–56].

Table 2 – Confusion matrix for binary classification

Predicted / **Actual**	**Negative Class**	**Positive Class**
Negative Class	True Negative (TN)	False Positive (FP)
Positive Class	False Negative (FN)	True Positive (TP)

Using the values obtained in the Confusion Matrix, a plethora of evaluation metrics can be computed. Some of the most commonly used are Accuracy, Precision, Recall, F1, and Mathews Correlation Coefficient (MCC) [55–57].

- **Accuracy**: measures the ratio of true instances predicted, both negative and positive, among the total number of instances (Equation 2). It measures how well the model performs overall. The values for the Accuracy score range from 0 to 1, the latter being a perfect model.
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

- **Precision and Recall**: both metrics are based on the positive class. Precision measures the fraction of true positive predicted instances from all instances predicted as positive (Equation 3). Recall measures the correctly predicted positive instances among all actual positive instances (Equation 4). The values for these metrics range from 0 to 1, with 1 being the best model.
$$Precision = \frac{TP}{TP + FP} \tag{3}$$
$$Recall = \frac{TP}{TP + FN} \tag{4}$$

- **F1**: this metric depicts the harmonic mean between Recall and Precision (Equation 5). Like all metrics before, the values range from 0 to 1, with 1 being the best result.
$$F1 = 2 \times \frac{(Precision \times Recall)}{(Precision + Recall)} \tag{5}$$

- **MCC**: this metric is a correlation coefficient between the observed and predicted binary classifications. It considers the four types of results (TN, FN, FP, TP) and is regarded as a balanced metric that can be used even if classes are unbalanced. (Equation 6). The values range between -1 and 1. A coefficient of 1 is a perfect prediction (all examples correctly predicted), 0 is no best than a random prediction, and -1 is an opposite prediction (all examples incorrect).
$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{6}$$

These metrics are often complemented by the Receiver Operating Characteristic (ROC) curve and the Precision-Recall (PR) curve. ROC plots Recall against False Positive Rate (FPR)

(Equation 7), whereas PR plots Precision against Recall. Both ROC and PR have an Area Under Curve (AUC) value which ranges from 0 to 1, being 1 the perfect classifier algorithm [54].

$$FPR = \frac{FP}{FP + TN} \tag{7}$$

The purpose of all these metrics is to evaluate the performance of trained models. It is best practice to evaluate models using a new, *unseen* dataset. To that extent, it is common to divide the initial dataset into two smaller subsets. Usually, a bigger subset is used to train the model, while the other is reserved for performance evaluation only. That way, we obtain an unbiased evaluation of the model's performance. Sometimes, the training subset is divided further to obtain a small validation subset. This subset can be used when performing HPO or to evaluate the training performance as the model is being trained (often used in DL). A more common validation technique among ML tasks is *k*-fold Cross-Validation. This technique partitions the training set into *k* subsets of approximately equal size (Figure 5). Then, the model is trained using $k-1$ subsets, and the remaining one is used as a test set. This procedure is repeated until each *k* subset has been used as a test subset [58].

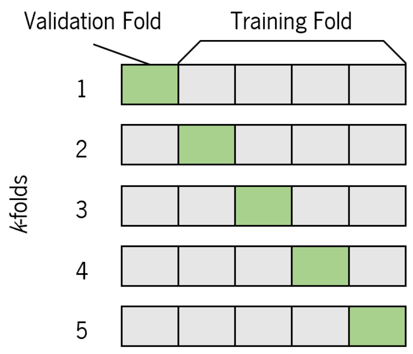

Fig. 5 – *k*-fold cross validation structure.

2.3 Deep Learning

As previously explained (Section 2.2), DL is a subfield of AI. Furthermore, it is considered a subfield of ML (Figure 3). Chollet describes DL as a new way of learning representations from data that focuses on learning successive layers of increasingly meaningful representation [27].

The *deep* from DL comes from the successive layers of representations, sometimes reaching tens or hundreds of layers. In contrast, ML and some simpler NN are often called *shallow* learning since they possess only one or two layers [24, 27]. This subfield improved image classification, speech recognition, machine translation, text-to-speech conversion, search engine results, and natural language question answering [27]. These are just some examples, but it is visible that DL improved a lot in the NLP field.

DL advanced the ML field by excluding the need to hand-design features, which is time-consuming and of human expertise. As stated before, DL uses representation learning, which is a way to discover effective features and their mappings from the given data. This means that DL can use raw data and learn from it, whereas traditional ML would have a hard time [27, 59, 60].

2.3.1 Neural Networks

Artificial Neural Networks (ANN) is a group of connected nodes or neurons. In some sense, these NN try to replicate how our brain works. The idea of creating an "artificial brain" first appeared in the 1940s [61]. Later, in 1957, Rosenblatt created an artificial neuron, which he called *Perceptron* [62]. The *Perceptron* is one of the simplest ANN, and it was based on a (Threshold Logic Unit (TLU)).

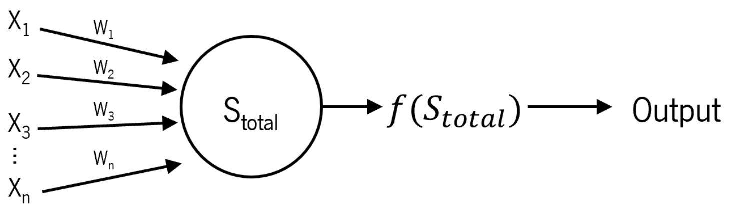

Fig. 6 – Perceptron. Adapted from [63].

Essentially, a Perceptron way of working is straightforward (Figure 6). It is composed of a node that receives input from N external sources, numbered 1 to N. Each input (i) is called X_i and its associated weight is called W_i. Therefore, the total input that a node receives is the weighted sum of all inputs S_{total}, given by equation 8a. This sum becomes the input to the step function (f) which results in the output (y) (Equation 8b).

$$S_{total} = \sum_{i=1}^{N} W_i X_i = W_1 X_1 + W_2 X_2 + ... + W_N X_N \tag{8a}$$

$$y = f(S_{total}) \tag{8b}$$

The most common step function (f) used in Perceptrons is the *Heaviside* step function. It basically functions as a switch. If S_{total} is equal to or greater than 0 the output value is 1, otherwise, the output is 0 (Equation 9).

$$Heaviside(S_{total}) = \begin{cases} 0 & \text{if } S_{total} < 0 \\ 1 & \text{if } S_{total} \geqslant 0 \end{cases} \tag{9}$$

In 1969, Minsky and Papert pointed out some problems with Perceptrons. The main problem was that they were not capable of solving nonlinear problems [64]. This difficulty was quickly surpassed by stacking multiple Perceptrons, which resulted in an ANN called Multi-Layer Perceptron (MLP) [65].

MLP is an ANN constituted by an *input layer*, which serves just as a passthrough for the input data, one or more layers of Perceptrons, which are called *hidden layers*, and finally, one more layer of Perceptrons which functions as the *output layer*. Except for the output layer, all layers receive an additional *bias neuron*, which always outputs 1 and is fully connected to the next layer (Figure 7). If the ANN has many hidden layers, it then can be called Dense Neural Network (DNN) [65].

In 1986, Rumelhart *et al.* introduced an enduring training algorithm called *Backpropagation* [66]. This algorithm is executed in two distinct phases: one forward pass and one backward pass. In the forward phase, the algorithm calculates the error (e.g., Mean Squared Error (MSE)) of the NN by comparing its predicted output to the expected output. After the error is obtained, the backward phase runs through each layer of the NN in reverse to quantify the error contribution of each layer connection. Lastly, once the algorithm has these errors, it uses an optimization algorithm to adjust the weights to reduce these errors. This training process is done repeatedly

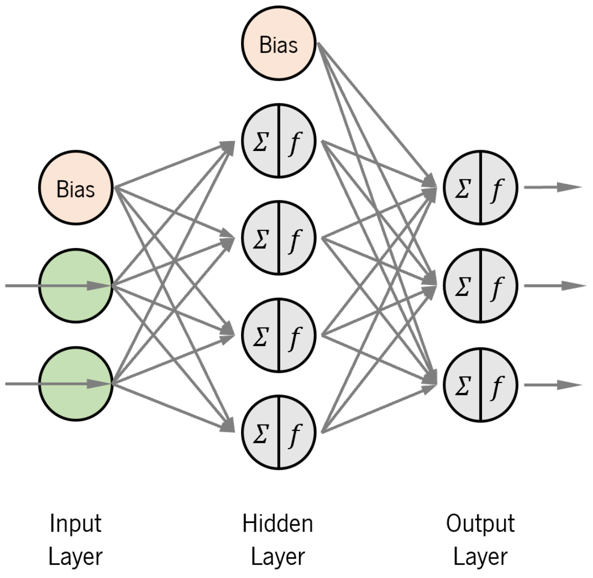

Fig. 7 – Multi-Layer Perceptron. Adapted from [65].

until the optimization algorithm stops, either by converging or reaching the maximum number of epochs [65]. Among the vast number of optimization algorithms, two remain the most utilized. They are Stochastic Gradient Descent (SGD) and Adam [67, 68]. Since optimization algorithms were applied, there was a need to change the step functions into activation functions. The most used activation function is the logistic function (e.g., Sigmoid), but later other functions appeared, like Hyperbolic Tangent (Tanh) or Rectified Linear Unit (ReLu) [65].

- **Sigmoid**: the sigmoid curve has an S-like shape (Figure 8). Its values range from 0 to 1. Because of its properties, the output is very sensitive to small changes in the input when it is close to 0. However, when the input is closer to either end, the output's sensibility becomes nearly null, being the main problem with this function. This function receives any real values and outputs a value between 0 and 1 [69]. It is given by equation 10.

$$\sigma(S_{total}) = \frac{1}{1 + e^{-S_{total}}} \tag{10}$$

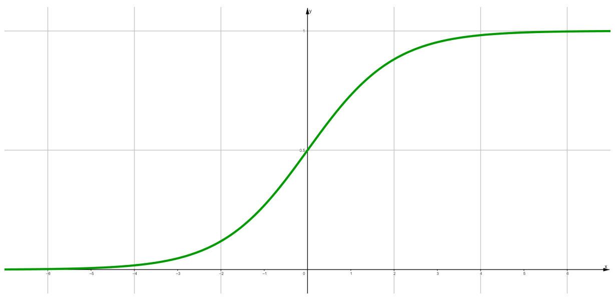

Fig. 8 – Sigmoid curve.

- **Tanh**: the Tanh curve (Figure 9) is very similar to the Sigmoid curve, sharing the same problem. However, its values range from -1 to 1. It is usually used as an activation function on hidden layers [69]. This function is given by equation 11.

$$tanh(S_{total}) = 2\sigma(2S_{total}) - 1 \tag{11}$$

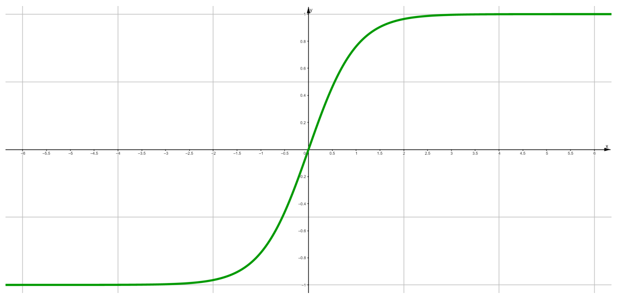

Fig. 9 – Tanh curve.

- **ReLu**: even though ReLu looks like a linear function, it is nonlinear because the output is 0 when $X_i < 0$ (Figure 10). The main advantages of ReLu are that it does not have the problem the other two have, and it also is less resource-heavy. However, since the output is 0 for all negative inputs, it can make the neurons enter a perpetually inactive state. This problem is called “Dying ReLu”. It is still prevalent in hidden layers, especially in Convolutional Neural Networks (CNN) [69]. It is given by equation 12.

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

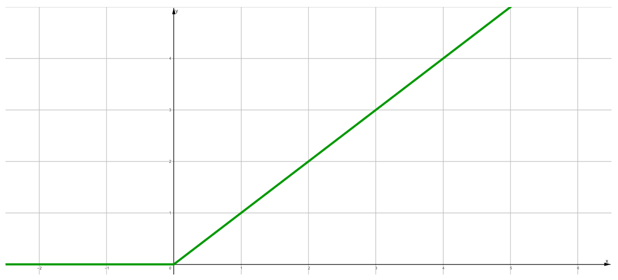

Fig. 10 – ReLu curve.

2.3.2 Convolutional Neural Networks

Since the adoption of ANN, there was an immediate interest in applying that new technology into images. In 1980, Fukushima introduced the *neocognitron*, a NN inspired by studies of the visual cortex [70]. This NN steadily evolved into what we now call CNN. The main focus of CNN is pattern recognition within images, and by 1998, LeCun *et al.* introduced the *LeNet-5* architecture and it quickly became widely used to recognize handwritten numbers [65, 71]. Later, new architectures appeared such as AlexNet, VGGNet, GoogLeNet, and ResNet [72–75]. Nowadays, CNNs are also used in text classification and other NLP tasks [76, 77]. The main difference between a CNN and a DNN is the *Convolutional* and *Pooling* layers that a CNN possesses. Usually, a CNN is built in the following order: Convolutional layer, Pooling layer, and one or more Dense layers.

- **Convolutional layer**: this is the most important part of a CNN. Its name comes from the use of a linear operation called Convolution. This layer uses the Convolution operation, where a small array of numbers, also known as *kernel* or *filter*, is applied over the input's array of numbers, also known as a *tensor*. The output tensor, also known as *feature map*, is obtained by the sum of the values of an element-wise product between the kernel and input tensor (Figure 11). The key hyperparameter in Convolution is the kernel size, which typically is 3×3. In essence, the filter is applied over a part of the data of the same size, it calculates the output tensor of that part of the data, and then the filter moves. The number of steps the filter moves is known as *stride*. This process occurs until all data has been covered. Other parameters of this process can be changed, such as *padding*. *Padding* consists of adding a layer around the input tensor so that the generated feature map is of the same size as the input tensor [27, 65, 78, 79].

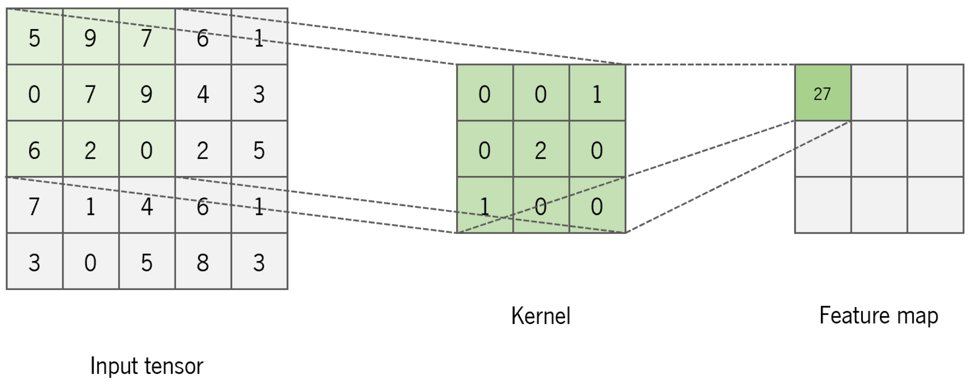

Fig. 11 – Convolution process. The output value exemplified corresponds to $(5 \times 0) + (9 \times 0) + (7 \times 1) + (0 \times 0) + (7 \times 2) + (9 \times 0) + (6 \times 1) + (2 \times 0) + (0 \times 0) = 27$.

- **Pooling layer**: in general, the pooling layer serves as a downsampling technique. In other words, it is used with the intent to reduce the dimensionality of the feature map created by the convolution layer. This downsampling technique is usually done with a 2×2 filter and stride 2, in order to downsample the feature map by a factor of 2. Similar to the convolutional operation, this process is performed until the feature map has been totally covered. The most common forms of pooling are *max pooling* and *average pooling*. Max pooling selects the maximum value among the values present in the zone of the feature map that the pooling filter is currently on, and discards the others (Figure 12). Average pooling follows the same principle but instead, it calculates the average value [27, 78].

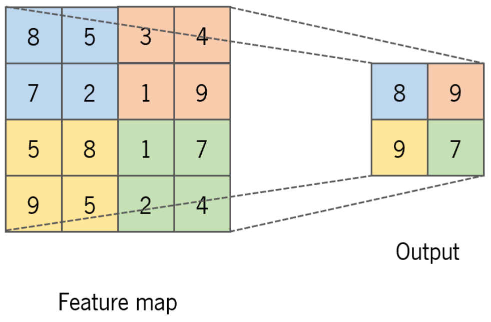

Fig. 12 – Max pooling process. Adapted from [78].

2.3.3 Recurrent Neural Networks

As seen previously, DNNs and CNNs process one input at a time, without previous recollection of what came before. This, however, is not an ideal situation when the data is sequential such as text. For that reason, these two models can lose information when training with such type of data. To tackle this issue, Recurrent Neural Networks (RNN) appeared [27]. The concept of RNN was first introduced in 1986 by Rumelhart *et al.* [66].

In essence, RNN is a type of NN that has an internal loop. This loop is what allows the model to process sequential data, while maintaining a memory, or *state*, containing information of what it has already processed (Figure 13(A)). Clarifying its working, the RNN model receives a sequence of vectors as input, then it iterates over that sequence. The output comes from an activation function (f), which uses the current iteration (t) of the input and its weights (W_i), the current state of the model and its weights (W_s), and a bias (b). This output becomes the new state in $t+1$ (Figure 13(B)) [27, 80]. This model is usually trained using Back Propagation Through Time (BPTT), which is the Back Propagation algorithm adapted for sequences [81].

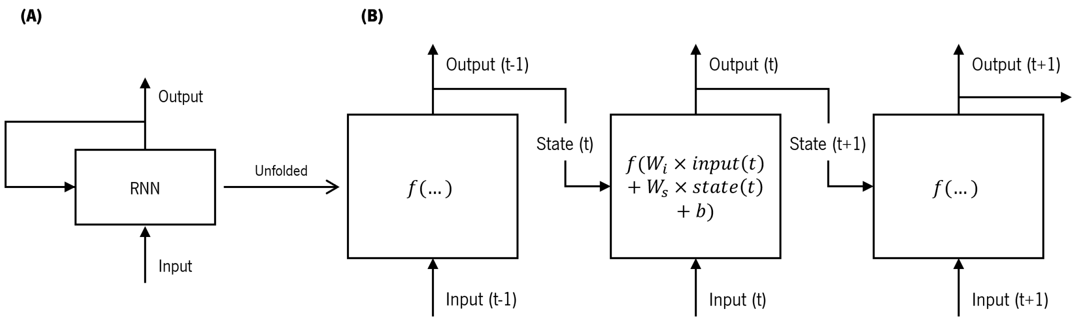

Fig. 13 – (A) Simple RNN structure representation. (B) RNN unfolded over time. Adapted from [27].

RNNs have become the staple model for sequential data, but their true value is in the realm of NLP. However, since nothing is perfect, this model has its drawbacks, being the main the one Hochreiter *et al.* described in their paper [82]. The model has a problem when dealing with long-term sequences, as a result of vanishing and exploding gradients during the training process [83, 84]. To take on this problem, some RNN variations appeared like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU).

2.3.3.1 *Long Short-Term Memory*

The LSTM model was created with the intent to solve the vanishing gradient problem the RNN model has [85]. The LSTM architecture consists of memory blocks (Figure 14). Each block has one or more memory cells and three gates: the input, output, and forget gates. These gates are what grants LSTM memory cells the ability to store and access information for a long time, and thus, alleviate the vanishing gradients problem [86].

The first step in the memory block structure is the combination of the current input ($Input(t)$) data and the output from the previous memory block ($Output(t-1)$). This results in a newly calculated input which is fed into the activation function of the forget gate (F). The forget gate (F) will decide which information should be discarded from the previous cell states ($S(t-1)$) through a multiplication of the previous cell state ($S(t-1)$) and the forget gate (F) output. The input is also fed into the activation function of the input gate (I), and its output is then multiplied with the output of an activation function using the same input. The output of this multiplication is then summed with the result of the multiplication of the previous cell state ($S(t-1)$) and forget gate (F) output, creating the new cell state ($S(t)$). The current cell state ($S(t)$) is then used to calculate the output ($Output(t)$), while also being passed to the next memory block.

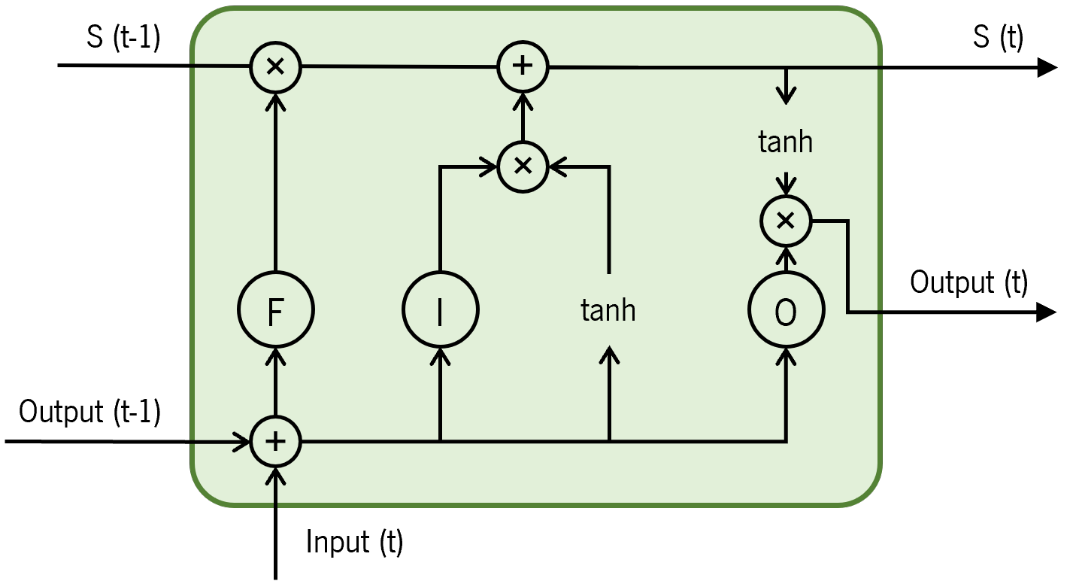

Fig. 14 – LSTM structure. ¨F¨ denotes the forget gate, ¨I¨ the input gate, ¨O¨ the output gate, and the $\times$ and $+$ symbols denote multiplication and sum, respectively. Adapted from [87].

Lastly, the input is fed into the output gate (O). In parallel, the current cell state ($S(t)$) passes through an activation function. The resulting outputs are then multiplied to produce the final output ($Output(t)$) of the memory block, which will be used in the next memory block [87, 88].

2.3.3.2 *Gated Recurrent Unit*

GRU first appeared in 2014 by the hands of Cho *et al.*. They state that it was encouraged by the LSTM memory block, but being simpler [89]. It is composed of just two gates: a reset and an update gates (Figure 15). The reset gate (R) receives the sum of the current input ($Input(t)$) and the previous state ($S(t-1)$) and performs the sigmoid function. The output of this gate will then determine how much information is passed to the current input. The update gate (U) works in the exact same way as the reset gate. The main difference is the way its output is used. The update gate (U) output is used in the computation of the current state ($S(t)$) and in the provisional state ($S_p(t)$). The provisional state ($S_p(t)$) is calculated by performing the Tanh activation function on the sum of the current input and the previous state ($S(t-1)$). The output from this function is then multiplied with the result from the update gate (U). The current state ($S(t)$) is then obtained by summing the provisional state ($S_p(t)$) with the result of the multiplication of the previous state ($S(t-1)$) with the reversed output ($1-U$) of the update gate (U) [89, 90].

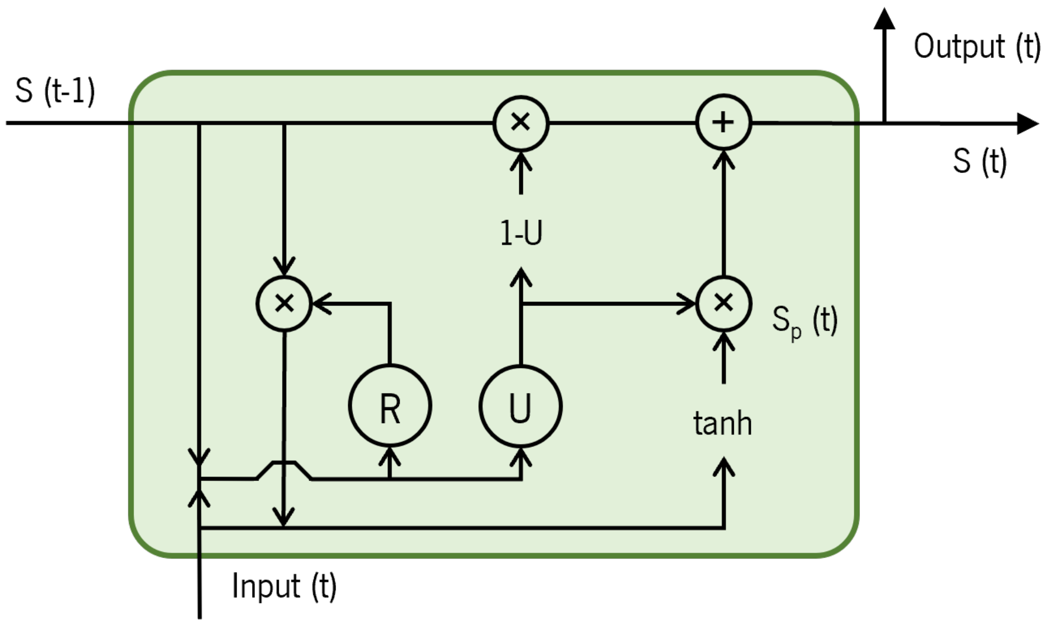

Fig. 15 – GRU structure. ¨R¨ denotes the reset gate, ¨U¨ the update gate, and the $\times$ and $+$ symbols denote multiplication and sum, respectively. Adapted from [91].

As stated by Chung *et al.*, GRU was able to outperform LSTM, on the majority of datasets, on audio related tasks. They even concluded that GRU was faster than other recurrent models [90].

2.3.4 Transformers

Before the appearance of transformers, state-of-the-art in NLP were recurrent architectures such as LSTM and GRU. As explained beforehand, these architectures possess a loop in the network connections that allows information to pass from one step to another. This peculiarity makes these types of architectures optimal for sequential data like text. However, these architectures have difficulties when dealing with long sequences of data [92]. To tackle this issue, in 2017, Vaswani *et al.* introduced a new simple architecture called Transformer [93].

The transformer is an attention-based architecture composed of encoder and decoder sections. The encoder-decoder sections work, in general, like RNNs but instead of outputting the *hidden state* at each step, the encoder component outputs only the *last hidden state* which corresponds to the encoded information from the whole input sequence. This encoded information takes the shape of a numerical vector. The job of the decoder is to decode the numerical representation, given by the encoder, back to the original information. However, if the sequence is long, a problem emerges. Since the encoder has to encode the entire sequence, some information might be lost when compressing everything into a unique and fixed representation [92]. To solve this problem, the attention mechanism part of the architecture comes into play.

The main concept behind attention is that instead of outputting a single *hidden state*, the encoder produces a *hidden state* at each step that the decoder can access. However, using all the *hidden states* would create an enormous input for the decoder. To prevent this from happening, the attention mechanism lets the decoder choose the weight for each of the encoder states. That way, the decoder assigns more importance to certain states of the encoder [92]. Nowadays, there are 3 types of transformers:

- **Encoder-only**: use the output of the encoder to provide a representation of the input sequence. They use Masked Language Modeling as a method of training. This involves the model corrupting (i.e., masking) certain parts of sentences and assigning the model to find the initial sentence. This type of model is well suited for tasks like text classification and NER. Models like BERT [8], RoBERTa [94], and DistilBERT [95] are examples of this type of transformer [92, 96].
- **Decoder-only**: predict the next word in the sentence. This type of model is usually used for text generation. Models like GPT [97] and CTRL [98] are examples of this type of transformer [92, 96].
- **Encoder-decoder**: adapt complex mappings from one sequence to another. This type of model is best applied for translation, summarization, and question-answering. Models like BART [99] and T5 [100] are examples of this type of transformer [92, 96].

2.4 Transfer Learning

Transfer Learning is a technique used in DL that focuses on transferring knowledge from one learner to another. The knowledge is transferred within related domains and it can also span across different tasks [101]. This transfer enables the possibility of utilizing knowledge from enormous datasets without the hassle of having to train a new learner with those datasets.

A common use within NLP is what Zhuang *et al.* call Parameter Sharing. It consists in using a pre-trained model and fine-tuning it to the desired domain and/or task. This involves using the base model architecture and only tuning the last layers [102]. This helps reduce the time needed to train a model for a specific task while using the knowledge obtained from big datasets. For biomedical NLP tasks, this technique quickly produced state-of-the-art results such as the ones obtained by Giorgi and Bader [103]. Some models, such as BioBERT [10], SciBERT [37], and BlueBERT [104] also managed to obtain state-of-the-art performance by transferring the knowledge that BERT [8] learned on the generic text together with training on biomedical text.

2.5 Hyperparameter Optimization

There are two types of parameters when training ML models: model parameters and hyperparameters. The model parameters are the values that the model uses and updates using backpropagation when training (e.g., the weights of neurons and biases in NN). Hyperparameters are the parameters that define the model configuration (e.g., learning rate, weight decay, optimizer, etc.). These parameters are part of the model core and are fixed values (i.e. cannot be updated via backpropagation). Therefore, these parameters are tuned in advance so that the model better fits the problem since they can have a high influence on the learning process. The process of adjusting the hyperparameters is called HPO [27, 105]. This task can be executed either by searching manually or automatically. The manual method implies finding the best set of hyperparameters by hand. However, this method is very cumbersome, even for an expert. To mitigate this problem, automatic methods were developed [106]. Nowadays there is a plethora of automatic search algorithms available:

- **Grid Search**: the principle of grid search is to train a model with each of the possible combinations of hyperparameter values. Exhausting all combinations, it outputs the best combination. This algorithm works better when dealing with fewer hyperparameters since the number of combinations increases exponentially with the number of hyperparameters [107].

- **Random Search**: similar to grid search, but instead of training each combination the algorithm selects random values from the hyperparameters search space. This makes this algorithm much more efficient and it can even obtain better results, as proved by Bergstra and Bengio [107]. This algorithm also needs to be given the number of combinations to try so that it can stop the optimization.
- **Bayesian Optimization**: the main difference from the algorithms talked about before is that bayesian optimization uses the results obtained before to find the next group of hyperparameters. It uses a surrogate model (i.e. statistical model) that intends to fit all observations into the target function. Then, an acquisition function, which uses the predicted distribution of the surrogate model, determines the usefulness of different points. This is performed by trading off between searching for areas that have not been explored (called exploration) and searching for better points within the explored area (called exploitation). This makes it more efficient than random search since it can find the optimal number of hyperparameter combinations [105, 106, 108].
- **Successive Halving**: it is similar to random search in the way of obtaining different combinations of hyperparameters values. This algorithm allocates a budget for all combinations, therefore, each combination is evaluated with a uniformly allocated budget. After evaluating the combinations, the algorithm discards half of the low-performing combinations, while the remaining half is evaluated again but doubles the budget. This evaluation continues until there is only one combination left. It is more efficient than random search, but it needs to adjust between the number of combinations and the budget [105, 109].
- **Hyperband**: it addresses the trade-off problem successive halving has by choosing the number of combinations in a dynamic way. The number of combinations is calculated based on the total number of points, the minimum number of instances required to train a model, and the available budget [108, 109].

2.6 AutoML

AutoML is a technique to automate the process of building a ML pipeline. Usually, to build a ML pipeline, experts need to manually prepare all the data, and features and fine-tune the model to achieve good results, in a trial-and-error manner. However, as the tasks and models increase in complexity, it has become an arduous task to perform all these steps manually [110]. He *et al.* splits AutoML into four methods [111]:

- **Data Preparation**: where new data is collected to create and/or augment a dataset and where data is cleaned to remove noise.
- **Feature Engineering**: where feature extraction is used to transform existing features, feature construction is used to create new features from data, and feature selection is used to reduce dimensionality by selecting the most important features.
- **Model Generation**: where a search space defines the design of neural architectures, an architecture optimization method defines the best model architecture, and the model's hyperparameters are tuned using the principles explained beforehand.
- **Model Evaluation**: where a model evaluation method assesses the performance of each model.

Some tools that perform AutoML are stated in table 3.

Table 3 – List of AutoML frameworks

Framework	**Description**
Auto-PyTorch [112]	Optimizes the network architecture and the training hyperparameters to enable fully automated DL
Neural Network Intelligence [113]	Open-source AutoML toolkit for feature engineering, neural architecture search, HPO, and model compression
PyCaret [114]	Open-source library that automates ML workflows
Auto-SKLearn [115]	Performs algorithm selection and hyperparameter tuning
AutoGluon [116]	Automates ML tasks on image, text, time series, and tabular data
Auto-Keras [117]	AutoML system based on Keras
TPOT [118]	AutoML tool that optimizes ML pipelines using genetic programming
H2O [119]	Automates the ML workflow, which includes automatic training and tuning of various models

2.7 Related Work

The majority of work in document classification uses DL architectures. Even though there are newer architectures like transformers (Section 2.3.4), older architectures (e.g., CNN) are still prevalent in document classification. Authors that use these type of architectures, often use them in conjunction with another type of architecture e.g., Ibrahim *et al.* used a CNN-BiLSTM. However, there are some authors that used traditional ML as the classifier, like Chen *et al.*. As for feature extraction, from all the authors that used DL models, only one did not use any type of pre-trained word embeddings. Jiang *et al.* even used embeddings from previous pre-trained models (i.e., BERT, XLNet, and RoBERTa). Table 4 presents previous work, performed by various authors, and the methodologies used by them.

Table 4 – Different document classification literature and their used methods. The prefix Bi and Hie stand for bidirectional and hierarchical, respectively. PoS and BoW stand for Part-of-speech and Bag-of-words, respectively. CCRCNN stands for Context-relevant Concept Recurrent CNN

Author	Architecture	Feature Extraction
Baker *et al.* [120]	CNN	Word2vec, BioNLP, BioNLP-2016[5]
Chen *et al.* [121]	LogR, SVM, RF	PoS, BoW
Fergadis *et al.* [122]	HieBiRNN	Word2vec
Luo *et al.* [123]	Ensemble(LSTM, CNN, BiLSTM-CNN, Recurrent CNN, HieLSTM)	fastText, PoS, NER
Abdulkadhar *et al.* [124]	Recurrent CNN	Word2vec
Baker and Korhonen [125]	CNN	BioNLP-2016[5]
Du *et al.* [126]	BiRNN	ELMo embeddings
Dollah *et al.* [127]	CNN	Word2vec, PoS, NER
Yang *et al.* [9]	XLNet	XLNet embeddings
Xu and Cai [128]	CCRCNN	Word2vec
Zhang and Zhang [129]	Text Graph Transformer	TF-IDF
Ibrahim *et al.* [130]	CNN-BiLSTM	BioNLP, BioWordVec
Jiang *et al.* [131]	LightXML	BERT, XLNet, RoBERTa embeddings

5 https://github.com/cambridgeltl/BioNLP-2016

3

METHODOLOGY

3.1 Development over the Omnia plataform

3.1.1 Overall description of the Omnia plataform

Omnia is an AutoML platform for bioinformatics. The platform was created for commercial use by the team at OmniumAI[1]. OmniumAI is a company that provides consulting, software development, and tailored training in the fields of AI and Data Sciences, boosted by the team's prowess in ML, DL, NLP, and optimization. With AI as its core, OmniumAI focuses on bioinformatics and biomedical data sciences, with enriched solutions for biological and biomedical data processing, analysis, mining, and integration. OmniumAI is a spin-off of the University of Minho created in 2021 by members of the Centre of Biological Engineering.

The Omnia platform contains a set of tools for the analysis of biological data. Currently, it has tools for the analysis of the following data and areas:

- **Generics**: subpackage that contains basic and universal data handling and transformation methods. It provides the ability to load data, transform data, and train ML models. All sub-packages listed below depend on this subpackage.
- **Compounds**: this subpackage contains methods to process compound data (e.g., smiles). It is capable of performing compound feature extraction, compound standardization, and molecular splitting.

1 https://omniumai.com/

- **Proteins**: this subpackage contains methods to process protein sequences. It provides the capacity to perform protein standardization, protein description, and protein-encoding.
- **Genes**: this subpackage contains methods to process DNA sequences. It is capable of performing DNA feature extraction.
- **Metabolomics**: this subpackage contains methods to process Liquid-Chromatography Mass Spectrometry (LC-MS) data. It is capable of processing LC-MS data and perform data reduction.
- **Transcriptomics**: this subpackage contains methods to process RNA-seq data. It is capable of performing RNA-seq data parsing, RNA-seq data processing, and RNA-seq data feature selection.
- **Single-cell transcriptomics**: this subpackage contains methods to process single cell (scRNA-seq) data. It provides the capacity to perform scRNA-seq data parsing, scRNA-seq data scaling, scRNA-seq data feature selection, and scRNA-seq data instance selection.
- **Text mining**: this subpackage contains methods to process text data. It is capable of performing text feature extraction and text processing.

3.1.2 Improving the omnia-text-mining package

The omnia-text-mining subpackage contains all the modules needed to process text. The subpackage is divided into two different modules, according to the type of process: Processing and Feature Extraction. Processing designates all the methods responsible for simple text processing (e.g., lowercase). Feature Extraction encompasses all methods responsible for extracting useful information from the available text. These two modules are important when trying to train ML models, especially traditional ML since these require the text to be converted *a priori* to vectors.

Below, we list the methods developed in the context of the present work for the Omnia package. For processing there are:

- **Lowercase**: transforms all text to its lowercase form. This method receives a pandas Dataframe, transforms the text, and returns a new pandas Dataframe with lowercase text.

- **Remove Punctuation**: removes all punctuation from the text. This method receives a pandas Dataframe, removes all punctuation from the text, and returns a new pandas Dataframe with the transformed text.
- **Remove Stopwords**: removes all stopwords from the text (as explained in section 2.1.2). This method receives a pandas Dataframe, removes stopwords from the text, and returns a new pandas Dataframe with the new text.

For feature extraction there are:

- **Named Entity Recognition**: extracts all the entities from text. This method receives a pandas Dataframe, extracts all available entities from the text, and returns a new pandas Dataframe with the entities and counts for each document.
- **Part of Speech**: extracts the part of speech (i.e., grammatical class) counts from the text. This method receives a pandas Dataframe, counts the number of parts of speech from the text, and returns a new pandas Dataframe with the number of words for each part of speech for each document.
- **Word Counts**: counts all the words in the text. This method receives a pandas Dataframe, counts the number of words in each document, and returns a new pandas Dataframe with the words present and the number of times they appear in each document.
- **TF-IDF**: applies the TF-IDF method (as explained in section 2.1.2). This method receives a pandas Dataframe with word counts, calculates the TF-IDF, and returns a new pandas Dataframe with the TF-IDF values for each word in each document.

3.1.3 Packages used in the development

Below, are some of the most important packages used in the omnia-text-mining subpackage.

- **Numpy**: package for scientific computing in Python. It provides a multidimensional array object, masked arrays, matrices, and an assortment of routines for fast operations on arrays [132]. This package was used globally for its' ease of use, the ability to create arrays, and the ability to perform quick computational calculations.

- **Pandas**: Python package providing fast and flexible data structures. The two primary data structures of pandas are Series (1-dimensional) and DataFrame (2-dimensional) [133]. This package was chosen for its ability to easily structure and handle data.
- **Scikit-learn**: open source ML library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model evaluation, and many other utilities [134]. This package was used for feature extraction methods, i.e., TF-IDF.
- **NLTK**: platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources, along with a suite of text-processing libraries for classification, tokenization, stemming, tagging, and parsing [135]. This package was chosen mainly for its' list of stopwords to use on the Remove Stopwords method explained above.
- **spaCy**: library for advanced NLP in Python. spaCy comes with pre-trained pipelines and supports tokenization and training for more than 60 languages. It features state-of-the-art NN models for tagging, parsing, NER, text classification, and more [136]. This package was used for its' ability to perform NER.
- **ScispaCy**: Python package containing spaCy models for processing biomedical, scientific or clinical text [137]. This package was used for its' models for biomedical text.
- **Pytorch**: Python package that contains data structures for multi-dimensional tensors and defines mathematical operations over these tensors. It has the ability to perform those operations with strong Graphics Processing Unit (GPU) acceleration. It also has the building blocks needed to construct NN models [138]. This package was used for its' ability to use GPUs to perform mathematical calculations and build NN models.
- **AutoGluon**: enables easy-to-use and easy-to-extend AutoML tool with a focus on automated stack ensembling, DL, and real-world applications spanning image, text, and tabular data [116]. This package was used for its' ability to perform AutoML (i.e., HPO) on DL models.
- **Transformers**: Hugging Face Transformers provides Application Programming Interface (API)s and tools to easily download and train state-of-the-art pre-trained models that

support tasks in different modalities such as NLP, Computer Vision, Audio, and Multimodal [139]. This package was chosen for its' capacity to easily load and train a DL model.

3.2 Machine Learning Models

3.2.1 Traditional machine learning models

Several ML models were trained to compare the performance between well-known traditional ML and DL models, and state-of-the-art biomedical DL models. Some word embeddings (Section 2.1.3) were also compared using a custom CNN model. These models were trained with the dataset explained beforehand. Some of the DL models were also used on an AutoML pipeline with the main objective being HPO.

The training of traditional ML was performed using the models established within the Omnia package. The training procedure is done using the pipelines built on the AutoGluon package. The models were trained using AutoGluon's *TabularPredictor* function. This function trains a set of ML models and provides their performance. In total, 8 traditional ML models were trained:

- **RF**: constructed by training a predetermined number of decision trees and combining their results to create a final estimator. Each tree in the ensemble is constructed using a sample taken from the training set. In addition, the optimal split between each node of the tree is determined using either all of the input data or a randomly selected subset of features. These two randomization sources are intended to reduce the variance of the forest estimator.
- **SVM**: employs an ideal hyperplane to divide observations into different classes depending on the information patterns in the data. This hyperplane allows the classification of new data.
- **KNN**: non-generalizing learning since it merely saves examples of the training data rather than making an effort to build a broad internal model. The classification is determined by a simple majority vote of each point's closest neighbors.

- **Gradient Boosting**: it is an ensemble of weak prediction models, typically, decision trees. The objective of this model is to find a function that best fits the input variables to the output variables. This is done by using a loss function and minimizing it. In total, 3 variants of gradient boosting were trained, more precisely eXtreme Gradient Boosting (XGBoost)[2], Light Gradient Boosting Machine (LGBM)[3], and CatBoost[4].
- **Extra Trees (XT)**: a random subset of candidate features is used, as in RF, but instead of looking for the most discriminative thresholds, the thresholds are randomly generated for each candidate feature, and the best of those are chosen as the splitting rule. This typically enables a slight reduction in the model's variance at the expense of a slight increase in bias.
- **Linear**: is a linear model for classification. In this model, a logistic function is used to simulate the probabilities describing the potential outcomes of a single trial.

3.2.2 Deep learning models

For DL, most of the models trained were variants of the BERT model introduced by Devlin *et al.* [8]. BERT is a technique for pre-training language representations. It was trained on a massive text corpus (nearly 3,300 million words) creating a general-purpose "language understanding" model. This pre-trained model can then be used for downstream NLP tasks (e.g. document classification).

Except for the CNN architecture, all models were implemented using the Transformers package[139] from the Hugging Face platform. The models can be divided into two distinct areas: models trained on generic text and models trained on biomedical text. Regarding the former, 4 models of the BERT structure were trained:

- **BERT**: presented in 2018, this model was trained on a 3,300 million word corpus, containing text from english Wikipedia and BooksCorpus, totaling 16GB of uncompressed text. It is composed of 12 layers of transformer blocks with a hidden size of 768 and 12 attention heads. This produces a 110 million parameter model. It was pre-trained with

2 https://github.com/dmlc/xgboost
3 https://github.com/microsoft/LightGBM
4 https://github.com/catboost/catboost

masked language modeling and next sentence prediction objectives. This level of training made it achieve state-of-the-art performance, at the time, in multiple NLP tasks [8].

- **RoBERTa**: presented in 2019, this model refines BERT's pre-training process through iterations that include training the model for longer periods of time with larger batches of data, eliminating the next sentence prediction objective, training on longer sequences, and dynamically altering the masking pattern used on the training data. It also adds more data for the training procedure, on top of the data used by the original BERT, totaling over 160GB of uncompressed text. This new training allowed this model to achieve state-of-the-art results, at the time, in various NLP tasks [94].
- **StructBERT**: presented in 2019, this model extends BERT by incorporating language structures into pre-training. Specifically, this model was pre-trained with two additional tasks to make the most of language structures at the word and sentence levels. The incorporation of dependency between words as well as sentences improves the model's ability to generalize and adapt. The two new objectives in pre-training made the model achieve state-of-the-art results, at the time, in some NLP tasks [140].
- **DeBERTa**: presented in 2020, by combining two unique methods, DeBERTa enhances both the BERT and RoBERTa models. The first is the disentangled attention mechanism, in which each word is represented by two vectors that convey its content and location, respectively, and the attention weights among words are determined using disentangled matrices on those vectors. Second, the output softmax layer is replaced with an enhanced mask decoder in order to predict the masked tokens for model pre-training. These two methods greatly enhance the effectiveness of model pre-training and the performance of downstream tasks. This model surpassed human performance on the SuperGLUE benchmark [141, 142].

In the matter of biomedical models, seven models of the BERT architecture were trained:

- **BioBERT**: presented in 2019, this model increments the data used on the pre-training of BERT. It adds biomedical text from PubMed Abstracts and PMC Full-text articles, bringing the total words used for training to 21,300 million words. Of these, 18,000 million of them are from biomedical text. This makes the BERT model focus more on biomedical documents. The training on biomedical data made this model perform better than BERT on 3 biomedical TM tasks [10].

- **SciBERT**: presented in 2019, this model was trained on a corpus of 3,100 million words obtained from articles from the Semantic Scholar[5] corpus. This model also has a vocabulary built from scientific text, improving its performance in biomedical NLP tasks [143].
- **BlueBERT**: presented in 2019, this model was pre-trained with abstracts from PubMed (4,000 million words) and clinical notes (MIMIC-III) (500 million words) over the base BERT. The new data adds more than 4,500 million words to the training corpus, from biomedical and clinical texts. At the time, this model achieved state-of-the-art on the BLUE benchmark [144].
- **BiomedRoBERTa**: presented in 2020, this model expanded the pre-training of the base RoBERTa model. It was pre-trained on 7,550 million words from full-text scientific papers from the Semantic Scholar[5] corpus [145].
- **PubMedBERT**: presented in 2020, this model was created to be domain-specific. This means that this model was trained from scratch on a corpus solely composed of biomedical text. The text used was obtained from abstracts from PubMed and full-text articles from PubMedCentral, resulting in a 3,100 million word corpus. At the time, this model achieved state-of-the-art performance in various biomedical NLP tasks [146].
- **ProcBERT**: presented in 2021, this model uses the BERT architecture to train from scratch on a corpus. The corpus was made by up-sampling experimental procedures extracted from articles, full-text articles, and chemical patents, resulting in 12,000 million words [147].
- **BioLinkBERT**: presented in 2022, this model uses LinkBERT as base. LinkBERT is a pre-trained model that uses document links (i.e., hyperlinks and citations) to access knowledge that spreads across multiple documents. BioLinkBERT is pre-trained from LinkBERT on PubMed with citation links. It achieved state-of-the-art performance in most of the tasks in the BLURB[6] benchmark [148].

5 https://www.semanticscholar.org/
6 https://microsoft.github.io/BLURB/

The CNN model was used to compare the performance of different pre-trained embeddings. The fastText and GloVe pre-trained embeddings are part of the generic text domain. BioWordVec, BioNLP, and BioNLP+Wiki are part of the biomedical text domain. Highlight to the latter, which was pre-trained on both generic and biomedical texts. In total, 5 pre-trained embeddings were tested:

- **fastText**: pre-trained on the Common Crawl, totalling 600 billion words. This pre-trained embedding is available in 300 dimensional vectors [33].
- **GloVe**: pre-trained on uncased text from Wikipedia available in 2014 and Gigaworld fifth edition, totalling in 6 billion words. It is available in 50, 100, 200, and 300 dimensional vectors [32]. The 300 dimensional vectors were used in this work.
- **BioWordVec**: based on the word2vec model. It was pre-trained on the PubMed text corpus and MeSH, totalling in 2.3 billion words. It is available as 200 dimensional vectors [35].
- **BioNLP**: based on the word2vec model. It was pre-trained from a combination of the texts from PubMed and PMC, totalling 5.5 billion words. It is available as 200 dimensional vectors [149].
- **BioNLP+Wiki**: it is the same as BioNLP but it was also pre-trained with text from Wikipedia [149].

Training

Figure 16 represents the structure used in the CNN model. The CNN begins with an embedding layer. This layer receives the pre-trained word embeddings listed before and vectorizes the input text to the corresponding embedding representations. Proceeding from this layer, the transformed data is then provided to 3 parallel convolutional layers (Conv1D). These 3 layers have each 100 neurons, a stride of 1, and a padding of 0. The main difference between them is the kernel size. One has a kernel of 3, another has a kernel of 4, and the last has a kernel of 5. This means that the same input will have 3 different feature maps. The output from each Conv1D layer passes through a ReLu activation function (Figure 10) and it is then forwarded to a Max Pooling layer. The output of all 3 Max Pooling layers is then concatenated. This vector passes through a Dropout layer with a dropout value of 0.5, and finally, the remaining features are then given to a fully connected layer that will act as an output layer. This last layer will decide if the input text is classified as relevant or non-relevant.

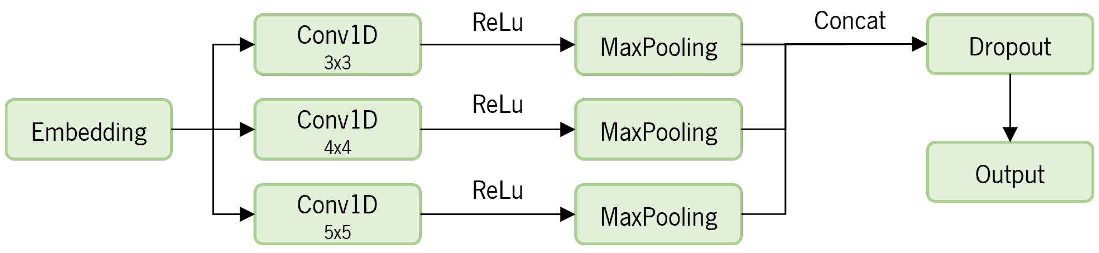

Fig. 16 – CNN structure utilized in this work. The values inside the Conv1D layers refer to the kernel size of that layer.

The text that this model receives as input receives a small preprocessing. Before being passed to the model, the documents are processed to remove their stopwords and to lowercase the text. This model was trained for 50 epochs with a learning rate of 0.25 and a batch size of 256. To evaluate the performance of each epoch, an evaluation set was created by selecting 15% of the training set.

The models based on the BERT structure were trained using the Transformers package [139]. Since most BERT models were not built to perform text/document classification, the package has a built-in function that makes it possible (*BertForSequenceClassification*). This function takes the model and adds an additional final layer that performs binary classification.

The text only suffered preprocessing on select cases. Since the vast majority of the models were trained with lowercase text, they received the text in lowercase. As some models were trained with uppercase text (i.e., text with both lower and uppercase letters), they received the raw text. There is no need to vectorize the text like it needs to be done for traditional ML (Section 2.1.2). The BERT-based models handle text in a specific way. As BERT is an encoder-only transformer, it creates representations of the text. Basically, as stated previously, it generally receives an uncased text.

The hyperparameters for each BERT-based model were chosen based on the parameters suggested by the authors or the parameters used to train said models. The number of training epochs was tested both on 3 and 4 epochs, the suggested value by the original authors of BERT. The learning rate was set to 0.00002, the warmup ratio was set to 0.1, weight decay was set to 0.01, the loss function was Cross-Entropy, and the optimizer function was set to AdamW. The batch size was set to 16 by default, however, for certain more complex models, the batch size was reduced to fit the memory available on the hardware used. Essentially, the batch size was chosen based on the available memory. To validate the performance of the training procedures and hyperparameters used, a validation set was created. This set was obtained by extracting 15% of the training set. To make the whole training and prediction reproducible, a seed was set to 16.

AutoML

The AutoML pipeline used in this work was obtained from the AutoGluon package. This pipeline was designed to perform HPO. The models used in this step were the BERT based biomedical models described previously, with the exception of BioLinkBERT and the addition of DeBERTa. For text preprocessing, as stated before, it was only transformed to lowercase as it is standard for BERT.

This pipeline performs HPO and trains the model with the best performing combination of hyperparameters. The HPO is performed by defining the lower and upper bounds (i.e., search space) for each hyperparameter. The hyperparameters tested were batch size, learning rate, learning decay, learning rate schedule, and warmup ratio.

4 RESULTS AND DISCUSSION

4.1 DATASET

The first step when training a ML model is to find or create a dataset that fills the requisites of the work. In the case of this work, the main focus was to use DL models to classify the relevance of biomedical documents. Therefore, a dataset for binary classification was needed. Consequently, the chosen one was a dataset used in Track 4 ("Mining protein interactions and mutations for precision medicine") from the BioCreative VI challenge, which took place in 2017. The main focus of this Track is Protein-Protein Interaction (PPI). PPI plays a crucial role in the cellular mechanisms of all living organisms and also contributes to predicting protein function and drug ability. In humans, changes in PPI can be indicative of disease [150, 151]. Therefore, it is important to uncover mutations that alter PPI. Because of this, the BioCreative team challenged the biomedical TM community with this task.

In this track, the challenge was divided into two different sub-tasks: Document Triage, which consists in identifying relevant PubMed literature describing genetic mutations affecting PPI, and Relation Extraction, which consists in extracting experimentally verified PPI affected by a genetic mutation. For this work, the dataset for the document triage task was chosen, since the main purpose of this sub-task is to perform binary classification. The data from this dataset is branched into a train set and a test set, comprised of 4082 and 1427 abstracts respectively. The statistics of the dataset are presented in table 5. The documents in this dataset were manually labeled as relevant/non-relevant by the curators of the BioGRID database [152].

Table 5 – Size of the two datasets from BioCreative VI Track 4

Dataset	Abstracts	Relevant	Non-Relevant
Train	4082	1729	2353
Test	1427	704	723

4.1.1 Challenge Overview

In total, 11 teams participated in the BioCreative VI Track 4 challenge. From these, 10 teams participated in the document triage task. Each team was allowed to submit 3 different runs. They could be 3 runs of the same method or different methods. In the end, there were 22 submissions for the document triage task. To compare the results from the submissions, the BioCreative team put up a baseline obtained using a SVM classifier.

The best average precision obtained was 72.5%, the best precision was 62.9%, the best recall was 98.0%, and the best F1 was 69.1% (Table 6). Team 414, which obtained the highest recall value, used a CNN which included several layers such as embedding, convolution, max pooling, dropout, and softmax [153]. Team 418, which obtained the best precision and F1 scores, used a Hierarchical Bidirectional Attention-Based RNN. It uses 2 bidirectional GRU, one as a sentence encoder and the other as a document encoder [122]. Finally, team 421, which obtained the best average precision, used an ensemble of NN (i.e. LSTM, CNN, LSTM-CNN, recurrent CNN, and hierarchical LSTM) [154].

Table 6 – Best results were submitted to the document triage task and the baseline. The values in bold represent the best result for each metric

Team	Average Precision	Precision	Recall	F1
414	0.508	0.502	**0.980**	0.664
418	0.716	**0.629**	0.766	**0.691**
421	**0.725**	0.607	0.800	0.690
Baseline	0.652	0.612	0.644	0.627

4.2 Traditional ML

In order to assess the performance of the DL models, some traditional ML models were trained first to establish a baseline. The training can be separated into 2 methodologies, explained by a defined minimum word frequency (cut-off) across all documents (i.e., a certain word needs to appear a defined minimum number of times across all documents) when performing TF-IDF. The main objective of these methodologies was to verify the behavior of the training process when more relevant words are used. Each of these methodologies was also evaluated either with all features or with 10% of the features. This method of dimensionality reduction should reduce the training resources while keeping near the same performance. The preprocessing methods performed before calculating the TF-IDF were lowercasing, removing newlines, removing stopwords, removing punctuation, removing URL, and lemmatization. Tables 7 and 8 details the results of a cut-off value of 10 and 5 words, respectively.

From the results expressed in table 7, it is noticeable that the majority of models performed worse when applying feature selection. In detail, the LGBM model performed worse in all metrics, in spite of the changes being small. The RF, XGBoost, XT, and SVM models performed worse on accuracy, precision, and MCC. The SVM also worsens on F1. On the other hand, CatBoost and Linear models performed better on all metrics. KNN performed better on accuracy, precision, and MCC. Special note to the Linear model, which got a substantial 7.7% improvement from feature selection. This result proves that feature selection, in certain cases, is a good way to improve the model's performance. This improvement was obtained while reducing the number of features from 3326 to 333 and also the time required to train the model from 183 seconds to 12 seconds. In general, the models obtained better results on recall. The increase on recall can be explained by the higher importance, to the relevant documents, of the features used in training. In that sense, the models would be better at classifying relevant documents. Otherwise, the worst performance, with feature selection, could be related to the lower number of features provided for training. Overall, the best model would be the CatBoost model with feature selection. It got the best performance in accuracy, F1, and MCC and still obtained the second-best performance in recall.

The results from table 8 show a generalized improvement when performing feature selection. Apart from the LGBM and XGBoost models, all models improved on 3 or more metrics. CatBoost,

Table 7 – Traditional ML performance from the Omnia package, with the TF-IDF values calculated with a cut-off value of 10. FS stands for Feature Selection. The values in bold represent the best result for each metric

Model	FS	Accuracy	Precision	Recall	F1	MCC
CatBoost	No	0.633	0.619	0.665	0.641	0.267
	10%	**0.637**	0.621	0.678	**0.648**	**0.276**
KNN	No	0.539	0.535	0.494	0.514	0.077
	10%	0.559	0.593	0.341	0.433	0.125
Linear	No	0.585	0.575	0.608	0.591	0.171
	10%	0.612	0.595	0.665	0.628	0.226
LGBM	No	0.628	0.612	0.672	0.640	0.258
	10%	0.626	0.611	0.666	0.637	0.253
RF	No	**0.637**	**0.632**	0.632	0.632	0.274
	10%	0.625	0.605	**0.690**	0.645	0.254
SVM	No	0.629	0.627	0.612	0.620	0.258
	10%	0.607	0.596	0.632	0.613	0.215
XGBoost	No	0.629	0.615	0.659	0.636	0.258
	10%	0.625	0.609	0.672	0.639	0.252
XT	No	0.628	0.629	0.599	0.614	0.255
	10%	0.624	0.608	0.668	0.636	0.249

Linear, and RF improved on all metrics. Like the results of the cut-off value of 10, the Linear model got a good improvement in its performance when using feature selection. In this case, it got a smaller, but still good improvement of 5.9% over the results without feature selection. SVM improved on all metrics but precision, however, with a negligible difference (i.e., 0.001). KNN and XT models got a small decrease in performance when focusing on recall and F1. In contrast, LGBM and XGBoost models achieved worse results on all metrics. Once again, CatBoost model with 10% feature selection was considered the best model overall. It got the best results on accuracy and MCC. It also got the second-best result in precision, the third-best F1, and the fourth-best recall.

Table 9 shows a comparison between the results of the chosen best models from tables 7 and 8 and the baseline results provided by the challenge team (Section 4.1.1). Firstly, it shows that the model trained with a TF-IDF cut-off value of 5 performed better than the model trained

Table 8 – Traditional ML performance from the Omnia package, with the TF-IDF values calculated with a cut-off value of 5. FS stands for Feature Selection. The values in bold represent the best result for each metric

Model	FS	Accuracy	Precision	Recall	F1	MCC
CatBoost	No	0.637	0.620	0.682	0.650	0.276
	10%	**0.642**	0.624	0.690	0.655	**0.286**
KNN	No	0.531	0.526	0.510	0.518	0.062
	10%	0.574	0.602	0.402	0.482	0.152
Linear	No	0.596	0.588	0.605	0.597	0.193
	10%	0.615	0.601	0.656	0.627	0.232
LGBM	No	0.636	0.614	0.709	**0.658**	0.277
	10%	0.622	0.606	0.670	0.637	0.247
RF	No	0.622	0.603	0.685	0.641	0.247
	10%	0.631	0.612	0.690	0.649	0.266
SVM	No	0.609	0.601	0.616	0.609	0.218
	10%	0.614	0.600	0.651	0.624	0.229
XGBoost	No	0.629	0.607	0.700	0.650	0.262
	10%	0.623	0.604	0.686	0.642	0.249
XT	No	0.633	0.609	**0.713**	0.657	0.271
	10%	0.640	**0.625**	0.675	0.649	0.281

with a TF-IDF cut-off value of 10. Most importantly, the model trained in this work performed better than the model used by the challenge team. The baseline model was a SVM trained with unigram and bigram features created from the titles and abstracts of the documents. In contrast, this works' model was trained using TF-IDF. Since both models were trained on the same dataset, this proves that TF-IDF is a better feature extraction method than unigram and bigram counts.

4.3 Deep Learning Models

Following the results obtained with traditional ML, the logical step was to train a simple DL model to compare the performance. The main objective was to see how a simple DL model would perform compared to traditional ML. For that reason, the model chosen was a simple CNN with

Table 9 – Comparison between the two best-performing models trained in this work and the BioCreative VI Track 4 challenge baseline. The number in CatBoost refers to the cut-off value used. The values in bold represent the best result for each metric

Model	Precision	Recall	F1
CatBoost 10	0.621	0.678	0.648
CatBoost 5	**0.624**	**0.690**	**0.655**
Challenge	0.612	0.644	0.627

3 parallel convolutional layers, as shown in figure 16. Using the same CNN architecture, different embeddings were also tested in order to see how they would change the performance of the model.

As stated in section 3.2, 5 different pre-trained embeddings were trained. They can be differentiated by the type of text they were trained with. fastText and GloVe were pre-trained with generic text, while BioWordVec, BioNLP, and BioNLP+Wiki were pre-trained with biomedical text. The latter was trained with a mix of biomedical and generic text. A quick analysis that can be done, for each pre-trained embedding, is to compare the number of tokens that the dataset has present in the embedding vocabulary. The dataset used (Section 4.1) contains a total of 57,038 different tokens. Knowing this, the number of known words on the pre-trained embeddings can be calculated. From the embeddings trained with generic text, fastText knew 21,437 words from the entire dataset, and GloVe knew only 16,719 words. These values correspond to 37,6% and 29.3% of the total number of words, respectively. From the biomedical embeddings, BioWordVec identified 42,544 words, BioNLP got 30,560, and BioNLP+Wiki got 30,767 words. These values correspond to 74.6%, 53.6%, and 53.9%, respectively, of the dataset's total number of words. The considerable difference between the generic and biomedical embeddings is obvious, as expected. This type of biomedical text possesses various specific terms that are not usually used in the generic text. This fact explains why these types of embeddings got a better coverage of the dataset vocabulary. Following this thought, it would be expected that the biomedical embeddings would result in a better performance when training the CNN model.

Table 10 presents the performance metrics obtained from training the CNN model with the different embeddings. fastText got the best results on recall and F1, while BioWordVec got the best results on accuracy, precision, and MCC. Overall, the best embedding, for this study case and model, would be the BioWordVec. It did not also get the highest word coverage, the best accuracy, the best precision, and the best MCC, but the other 2 metrics were not far from the best result. However, if the main objective was to obtain the maximum number of relevant documents, fastText embedding would be the ideal choice.

Table 10 – Performance comparison between the pre-trained embeddings. The values in bold represent the best result for each metric

Embedding	Accuracy	Precision	Recall	F1	MCC
fastText	0.631	0.600	**0.761**	**0.671**	0.275
GloVe	0.630	0.613	0.676	0.643	0.262
BioWordVec	**0.640**	**0.619**	0.702	0.658	**0.283**
BioNLP	0.637	0.618	0.693	0.653	0.277
BioNLP+Wiki	0.633	0.610	0.707	0.655	0.270

Comparing the BioWordVec embedding to the best traditional ML model discussed previously, CatBoost, it is noticeable that the results are quite proximate all around (Table 11). However, it is unexpected that BioWordVec got worse results in the majority of the metrics. This can be explained by the usage of TF-IDF on the CatBoost model since this method focuses only on the dataset vocabulary, i.e. the importance of each word is determined uniquely within the dataset. This technique would produce a better representation of the words present in the dataset, while the embedding would not have that word importance factor.

Table 11 – Performance comparison between the best-performing traditional ML model and the best-performing embedding. The values in bold represent the best result for each metric

Embedding	Accuracy	Precision	Recall	F1	MCC
CatBoost	**0.642**	**0.624**	0.690	0.655	**0.286**
BioWordVec	0.640	0.619	**0.702**	**0.658**	0.283

With regards to the unexpected results from the CNN model, the next step was to test the performance of more complex, and in theory better, DL models. The BERT model was chosen as a start. At the time it was presented, this model achieved state-of-the-art performance in multiple NLP tasks. From that point, several variations of BERT appeared. In this work, multiple of these variations were trained and tested (Section 3.2). To train these models, the text suffered no major preprocessing. As some models were pre-trained with lowercase text, these models were trained with lowercase text also. All the other models were trained with the raw text. The reason these models need no preprocessing is because of the way BERT handles text. The way BERT tokenizes the text is more complex than a simple tokenization method. It breaks down words that it does not know into smaller parts. To put it more accurately, BERT tokenizers are capable of breaking down a word that is not already part of its' vocabulary into words that are already part of the vocabulary. Because of that, it is recommended to use the raw text when training BERT models.

The models trained in this step are separated by the type of data they were pre-trained with. Like the embeddings, some models were pre-trained with generic text, while others were pre-trained with biomedical text. Table 12 contains the performance values of the models pre-trained with generic text. DeBERTa stands out from the other models, since it got the best result on accuracy, precision, F1, and MCC. On the other hand, StructBERT got the best result on recall, making it the best model to obtain the maximum number of relevant documents. Overall, the DeBERTa model would be the best, since it has the best performance on all metrics but recall. Even though it has the worst result in recall, it is still a very balanced model, so it would have a great performance globally. This result is somewhat expected, since DeBERTa is the newest model in the list and is based on BERT and RoBERTa. So, in theory, it would have gathered the best parts of both models it was based on, and it would have access to newer training techniques and data.

Table 12 – Performance of the models pre-trained with generic text. These models were trained with the Transformers package. The values in bold represent the best result for each metric

Model	Accuracy	Precision	Recall	F1	MCC
BERT	0.622	0.582	0.832	0.685	0.274
RoBERTa	0.623	0.586	0.807	0.679	0.269
StructBERT	0.597	0.557	**0.889**	0.685	0.246
DeBERTa	**0.674**	**0.631**	0.817	**0.712**	**0.366**

Table 13 presents the performance values of biomedical models. These models were initially pre-trained with biomedical text, in order to become more capable at dealing with this domain. From the results, BioLinkBERT got the best result on accuracy, precision, F1, and

MCC, while BioBERT got the best performance on recall. Overall, BioLinkBERT got the best performance, since it got the best result in 4 of 5 metrics. However, the other metric got a result within a respectable distance from the maximum value. Additionally, it should be noted that BioBERT, one of the first biomedical BERT models, got better performance than most of the models.

Table 13 – Performance of the models pre-trained with biomedical text. These models were trained with the Transformers package. The values in bold represent the best result for each metric

Model	Accuracy	Precision	Recall	F1	MCC
BioBERT	0.649	0.603	**0.841**	0.703	0.327
SciBERT	0.637	0.596	0.821	0.691	0.299
BlueBERT	0.632	0.595	0.798	0.682	0.284
BiomedRoBERTa	0.638	0.597	0.822	0.692	0.302
PubMedBERT	0.658	0.617	0.807	0.700	0.334
ProcBERT	0.647	0.605	0.815	0.695	0.316
BioLinkBERT	**0.676**	**0.633**	0.815	**0.713**	**0.369**

To compare the performance of generic and biomedical models, table 14 shows the performance of the best models from each domain. As expected, the BioLinkBERT model got an overall better performance than the generic ones. As said before, this result is expected, since the main purpose of the biomedical models is to perform biomedical NLP tasks. However, one unexpected result was the recall performance of StructBERT, since it got the best recall of all trained models. This would make it the best model to correctly classify the relevant biomedical documents, despite the fact that it was pre-trained with generic text.

Table 14 – Performance comparison between the best-performing generic models and the best-performing biomedical models. The values in bold represent the best result for each metric

Model	Accuracy	Precision	Recall	F1	MCC
StructBERT	0.597	0.557	**0.889**	0.685	0.246
DeBERTa	0.674	0.631	0.817	0.712	0.366
BioBERT	0.649	0.603	0.841	0.703	0.327
BioLinkBERT	**0.676**	**0.633**	0.815	**0.713**	**0.369**

4.4 AutoML

The following step was to perform AutoML, with an emphasis on HPO. The main goal was to test the effect of HPO on the models' performance, using the AutoGluon package. The main focus was to test, essentially, biomedical models as it was expected for them to perform better, as explained previously. However, with the performance obtained by the DeBERTa model, it was decided to include it in this step.

Table 15 shows the results obtained when training and performing HPO on several models. As seen, PubMedBERT got the best results on accuracy, precision, F1, and MCC. The best recall score was obtained by the BlueBERT model.

Contrary to expectations, the majority of the models got worse performance when using HPO. One way to explain these results is to analyze the way HPO works. As explained in section 2.5, HPO tries to find the best-performing hyperparameters within the given limits. Since the Bayesian Optimization method is used in this step to search for the best combination of hyperparameters, it is expected that different trials on the same model produce different hyperparameter combinations. It is this randomness that could explain the results obtained here.

Another explanation could be the way the models are trained. In this task, the package used for training and HPO is different from the package used in the previous steps. Even though AutoGluon uses the package Transformers to load the desired model, it uses its own method to train the model. This difference could be a reason that explains the lack of performance improvement since the HPO initial hyperparameters were the same as the ones used when training the models with the Transformers package.

Still another important factor is probably the number of trials to try new hyperparameter combinations. Theoretically, a higher number of trials would result in better performance since the optimization function would have more time to converge to a better hyperparameter combination. One complication found, when increasing the number of trials was that the software would, eventually, run into a problem and not recognize the hardware (GPU) needed to perform the training and HPO. This problem was a major hindrance when performing AutoML.

Table 15 – Results obtained when performing HPO using the AutoGluon package. Loss was used to evaluate the performance of the models. The values in bold represent the best result for each metric

Model	Accuracy	Precision	Recall	F1	MCC
BioBERT	0.645	0.604	0.817	0.694	0.313
SciBERT	0.627	0.586	0.831	0.687	0.283
BlueBERT	0.615	0.574	**0.847**	0.684	0.264
BiomedRoBERTa	0.629	0.596	0.767	0.671	0.271
PubMedBERT	**0.676**	**0.638**	0.790	**0.706**	**0.363**
ProcBERT	0.634	0.601	0.771	0.675	0.282
DeBERTa	0.644	0.601	0.831	0.697	0.315

4.5 Final Evaluation

To perform a final evaluation on the best performing model, trained in this work, a new PPI literature dataset was created. This dataset was constructed by extracting documents from the PubMed database using an API. This API was created by Rúben Rodrigues, a student at the BioSystems research group. It works by receiving queries and it searches the PubMed database for documents containing those queries. In this case, the queries searched were "protein-protein interactions" and "protein-protein interactions mutations". In total, 1,997 documents were obtained.

The model chosen for this task was BioLinkBERT. It was chosen because it was the best overall performing model for the case study. The preprocessing method, that was applied to this new dataset, was lowercasing, so it is the same as the documents used for training the model. After processing the text, it was classified using the trained model. From the 1,997 documents retrieved, the model classified 1,308 as relevant and the remaining 689 as non-relevant. From the performance on the test set, it is expected that from the 1,308 relevant documents, the model should have correctly classified about 1,066 documents as relevant.

To verify this hypothesis and these results, an analysis was performed to a reduced set of documents that were manually verified. In total, 30 documents were analysed to verify their contents and if the classification given by the model was correct. None of these documents was present in the datasets used for training and testing. From these 30 documents, 10 of them were the documents which the model classified with the highest probability of being relevant (top 10), 10 of which the model classified with the lowest probability of being relevant (bottom 10), and 10 of them with nearly a 50/50 change of being relevant (middle 10).

The top 10 documents classified as relevant by the model were all correctly classified, since these documents' content was about PPI and mutations in these sites. The bottom 10 documents also proved to be correctly classified by the model as non relevant. From the middle 10, half of them were correctly classified. The other half were wrongly classified as relevant. Even though these documents contain keywords related to PPI mutations articles, none of them actually contained information about the topic. Keywords like "protein", "mutations", and "interactions" could be the reason why the model wrongly classified these documents. On a closer analysis, some of these documents focus on the suppression or enhancement of gene silencing within structural proteins, or changes in proteins' terminal region caused by mutations. Furthermore, some of these, focus on the exploration of "sensitive" regions, important to protein aggregation, however this "sensitivity" can be caused by mutations. Finally, others focus on the change of protein flexibility in case of an immune response. However, it was somewhat expected for the model to wrongly classify the middle documents, since the model classifies them with a 50% probability of being relevant.

This analysis is another evidence that BioLinkBERT can be successfully used to classify relevant literature regarding PPI mutations.

5

CONCLUSION

One of the objectives of this dissertation was to explore and evaluate different ML techniques to perform document classification on biomedical literature. To that intent, several traditional ML and DL techniques were used in this work. It was possible to compare the impact made by different cut-off values and the different number of features when training traditional ML models. It was also possible to compare different types of pre-trained embeddings, from embeddings pre-trained exclusively on generic text to embeddings pre-trained on biomedical text and also a mix of both. The last step was to train several DL models to assess the performance gained from utilizing more complex models.

The traditional ML models were able to achieve better performance than the baseline defined by the challenge from where the dataset was extracted. The CNN model trained with the different pre-trained embeddings produced better results with the BioWordVec embedding. Despite the small differences, it still managed to outperform the other 4 pre-trained embeddings in 3 out of 5 metrics. Additionally, this model also surpassed the baseline results. The next step was to see how more complex DL models, i.e. transformers, performed with the same dataset. Several variants of the BERT structure were trained. From these, BioLinkBERT managed to obtain the best results. When compared to the challenge best submission, it managed to improve by 0.4 percent points on precision, 4.9 percent points on recall, and 2.2 percent points on F1. The main focus was to analyze the impact of HPO in the models' performance. To do that, the AutoGluon package was used because of its AutoML capabilities. Some of the models tested were trained using this package and the results proved that, in some cases, performing AutoML improved the performance.

In future work, it would be a good idea to train different DL structures. Some of the most commonly used models to perform document classification are a combination of CNNs and

LSTMs. Mixing transformers with these “basic” DL architectures could produce better results. One of the biggest issues found in this work was the lack of hardware power. Performing any type of NLP task requires very powerful hardware, especially when dealing with this type of models. Most of the encountered problems were in the AutoML task, since for the most part, the training process would fail, likely due to the lack of hardware power. This problem was the reason why HPO was not performed on the BioLinkBERT model. Still, regarding the AutoML task, it would be advisable to run the HPO task for longer periods of time. The AutoML results shown in this work were obtained with the HPO task running for 20 to 30 trials.

The overall results, obtained in this work, prove that there is room to improve current document classification techniques, being them traditional ML or DL.

BIBLIOGRAPHY

[1] Fleuren, W. W., & Alkema, W. (2015). Application of text mining in the biomedical domain. *Methods*, *74*, 97–106.

[2] Dang, S. (2014). Text mining : Techniques and its application. *International Journal of Enginerring & Technology Innnovation*, *1*, 22–25.

[3] Miner, G., Elder, J., Fast, A., Hill, T., Nisbet, R., & Delen, D. (2012). The Seven Practice Areas of Text Analytics. *Practical text mining and statistical analysis for non-structured text data applications* (pp. 29–41). Elsevier.

[4] Mowafy, M., Rezk, A., & El-Bakry, H. (2018). An efficient classification model for unstructured text document. *American Journal of Computer Science and Information Technology*, *06*.

[5] Allahyari, M., Pouriyeh, S., Assefi, M., Safaei, S., Trippe, E. D., Gutierrez, J. B., & Kochut, K. (2017). A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques.

[6] Kowsari, K., Meimandi, K. J., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text classification algorithms: A survey. *Information (Switzerland)*, *10*(4), 1–68.

[7] Kadhim, A. I. (2019). Survey on supervised machine learning techniques for automatic text classification. *Artificial Intelligence Review*, *52*(1), 273–292.

[8] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, *1*(1950), 4171–4186.

[9] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding.

[10] Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., & Kang, J. (2020). BioBERT: A pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, *36*(4), 1234–1240.

[11] Adhikari, A., Ram, A., Tang, R., & Lin, J. (2019). DocBERT: BERT for Document Classification.

[12] Bornmann, L., & Mutz, R. (2015). Growth Rates of Modern Science : A Bibliometric Analysis Based on the Number of Publications and. *Journal of the Association for Information Science and Technology*, *66*(11), 2215–2222.

[13] Saffer, J. D., & Burnett, V. L. (2014). Introduction to biomedical literature text mining: Context and objectives. *Methods in Molecular Biology*, *1159*, 1–7.

[14] Zweigenbaum, P., Demner-fushman, D., Yu, H., & Cohen, K. B. (2007). Frontiers of biomedical text mining: Current progress. *Briefings in Bioinformatics*, *8*(5), 358–375.

[15] Shatkay, H., & Craven, M. (2012). Mining the Biomedical Literature. *Mining the biomedical literature*.

[16] Holzinger, A., Schantl, J., Schroettner, M., Seifert, C., & Verspoor, K. (2014). Biomedical text mining: State-of-the-art, open problems and future challenges. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8401*, 271–300.

[17] Sarkar, D. (2019). Text Classification. *Text analytics with python* (pp. 275–342). Apress.

[18] Hart, M., Manadhata, P., & Johnson, R. (2011). Text classification for data loss prevention. *HP Laboratories Technical Report*, (114), 1–21.

[19] Sulea, O. M., Zampieri, M., Malmasi, S., Vela, M., Dinu, L. P., & Van Genabith, J. (2017). Exploring the use of text classification in the legal domain. *CEUR Workshop Proceedings*, *2143*.

[20] Kang, M., Ahn, J., & Lee, K. (2018). Opinion mining using ensemble text hidden Markov models for text classification. *Expert Systems with Applications*, *94*, 218–227.

[21] 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.

[22] Hirschberg, J., & Manning, C. D. (2015). Advances in natural language processing. *Science*, *349*(6245), 261–266.

[23] Indurkhya, N., & Damerau, F. J. (2010). *Handbook of Natural Language Processing* (2nd). Chapman & Hall/CRC.

[24] Deng, L., & Liu, Y. (2018). *Deep Learning in Natural Language Processing* (L. Deng & Y. Liu, Eds.). Springer Singapore.

[25] Sebastiani, F. (2002). Machine Learning in Automated Text Categorization. *ACM Computing Surveys*, *34*(1), 1–47.

[26] Yun-tao, Z., Ling, G., & Yong-cheng, W. (2005). An improved TF-IDF approach for text classification. *Journal of Zhejiang University-SCIENCE A (Applied Physics & Engineering)*, *6*(1), 49–55.

[27] Chollet, F. (2018). *Deep Learning with Python*. Manning Publications Co.

[28] Liu, Y., Liu, Z., Chua, T.-s., & Sun, M. (2015). Topical Word Embeddings. *AAAI Press*, 2418–2424.

[29] Mikolov, T., Yih, W.-t., & Zweig, G. (2013). Linguistic Regularities in Continuous Space Word Representations. *Proceedings of NAACL-HLT*, 746–751.

[30] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, 1–12.

[31] Parwez, M. A., Abulaish, M., & Jahiruddin. (2019). Multi-Label Classification of Microblogging Texts Using Convolution Neural Network. *IEEE Access*, *7*, 68678–68691.

[32] Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, *31*(6), 1532–1543.

[33] Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., & Joulin, A. (2018). Advances in pre-training distributed word representations. *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

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

[35] 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), 1–9.

[36] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners.

[37] Beltagy, I., Lo, K., & Cohan, A. (2019a). Scibert: Pretrained language model for scientific text. *EMNLP*.

[38] Brownlee, J. (2016). *Master Machine Learning Algorithms: discover how they work and implement them from scratch*. Machine Learning Mastery.

[39] Baştanlar, Y., & Özuysal, M. (2014). Introduction to Machine Learning. In M. Yousef & J. Allmer (Eds.), *Mirnomics: Microrna biology and computational analysis* (pp. 105–128).

[40] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (Second). The MIT Press.

[41] MacQueen, J. (1967). Some Methods for classification and Analysis of Multivariate Observations. *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, *1*, 281–297.

[42] Lloyd, S. P. (1982). Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, *28*(2), 129–137.

[43] Zhao, W. L., Deng, C. H., & Ngo, C. W. (2018). k-means: A revisit. *Neurocomputing*, *291*, 195–206.

[44] Shi, N., Liu, X., & Guan, Y. (2010). Research on k-means clustering algorithm: An improved k-means clustering algorithm. *3rd International Symposium on Intelligent Information Technology and Security Informatics, IITSI 2010*, 63–67.

[45] Pearson, K. (1901). LIII. 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.

[46] Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, *24*(6), 417.

[47] Abdi, H., & Williams, L. J. (2010). Principal component analysis. *Wiley Interdisplinary Reviews: Computational Statistics*, *2*, 1–47.

[48] Ringnér, M. (2008). What is principal component analysis? *Nature Biotechnology*, *26*(3), 303–304.

[49] Jolliffe, I. (2005). Principal Component Analysis. *Encyclopedia of statistics in behavioral science* (pp. 1215–1220). John Wiley & Sons, Ltd.

[50] 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*, *12*(6), 1.

[51] van der Maaten, L., & Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, *9*(86), 2579–2605.

[52] Hinton, G., & Roweis, S. (2003). Stochastic neighbor embedding. *Advances in Neural Information Processing Systems*.

[53] van der Maaten, L. (2014). Accelerating t-SNE using Tree-Based Algorithms. *Journal of Machine Learning Research*, *15*(93), 3221–3245.

[54] Liu, Y., Zhou, Y., Wen, S., & Tang, C. (2014). A Strategy on Selecting Performance Metrics for Classifier Evaluation. *International Journal of Mobile Computing and Multimedia Communications*, *6*(4), 20–35.

[55] Hossin, M., & Sulaiman, M. (2015). A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process*, *5*(2), 01–11.

[56] Dalianis, H. (2018). Evaluation Metrics and Evaluation. *Clinical Text Mining*, (1967), 45–53.

[57] Boughorbel, S., Jarray, F., & El-Anbari, M. (2017). Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric. *PLoS ONE*, *12*(6), 1–17.

[58] Berrar, D. (2019). Cross-validation.

[59] Min, S., Lee, B., & Yoon, S. (2016). Deep learning in bioinformatics. *Briefings in Bioinformatics*, *18*(5), bbw068.

[60] Wani, M. A., Bhat, F. A., Afzal, S., & Khan, A. I. (2020). *Advances in Deep Learning* (Vol. 57). Springer Singapore.

[61] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*, 115–133.

[62] Rosenblatt, F. (1957). *The Perceptron, a Perceiving and Recognizing Automaton* (tech. rep.). Cornell Aeronautical Laboratory. Buffalo, N.Y.

[63] Krogh, A. (2008). What are artificial neural networks? *Nature Biotechnology*, *26*(2), 195–197.

[64] Minsky, M., & Papert, S. (1969). *Perceptrons.* M.I.T. Press.

[65] Géron, A. (2019). *Hands-on Machine Learning whith Scikit-Learing, Keras and Tensorfow*.

[66] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

[67] Darken, C., Chang, J., & Moody, J. (1992). Learning rate schedules for faster Stochastic gradient search. *Neural Networks for Signal Processing - Proceedings of the IEEE Workshop*, (August), 3–12.

[68] Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization, 1–15.

[69] Feng, J., & Lu, S. (2019). Performance Analysis of Various Activation Functions in Artificial Neural Networks. *Journal of Physics: Conference Series*, *1237*(2), 022030.

[70] Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, *36*(4), 193–202.

[71] Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

[72] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc.

[73] Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition.

[74] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–9.

[75] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition.

[76] Wang, P., Xu, J., Xu, B., Liu, C.-L., Zhang, H., Wang, F., & Hao, H. (2015). Semantic clustering and convolutional neural network for short text categorization. *ACL*.

[77] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, *12*, 2493–2537.

[78] Yamashita, R., Nishio, M., Do, K. R. G., & Togashi, K. (2018). Convolutional neural networks : an overview and application in radiology. *Insights into Imaging*, *9*, 611–629.

[79] O'Shea, K., & Nash, R. (2015). An Introduction to Convolutional Neural Networks, 1–11.

[80] Goldberg, Y. (2017). Neural Network Methods for Natural Language Processing. *Synthesis Lectures on Human Language Technologies*, *10*(1), 1–309.

[81] Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.

[82] Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J. et al. (2001). Gradient flow in recurrent nets: The difficulty of learning long-term dependencies.

[83] Bengio, Y., Frasconi, P., & Simard, P. (1993). The problem of learning long-term dependencies in recurrent networks. *IEEE International Conference on Neural Networks*, 1183–1188 vol.3.

[84] Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, III–1310–III–1318.

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

[86] Graves, A. (2012). Long Short-Term Memory. *Supervised sequence labelling with recurrent neural networks* (pp. 37–45). Springer.

[87] Van Houdt, G., Mosquera, C., & Nápoles, G. (2020). A review on the long short-term memory model. *Artificial Intelligence Review*, *53*(8), 5929–5955.

[88] Hua, Y., Zhao, Z., Li, R., Chen, X., Liu, Z., & Zhang, H. (2019). Deep Learning with Long Short-Term Memory for Time Series Prediction. *IEEE Communications Magazine*, *57*(6), 114–119.

[89] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1724–1734.

[90] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, 1–9.

[91] Wang, Y., Liao, W., & Chang, Y. (2018). Gated recurrent unit network-based short-term photovoltaic forecasting. *Energies*, *11*(8).

[92] Tunstall, L., von Werra, L., & Wolf, T. (2022). *Natural Language Processing with Transformers*. O'Reilly Media.

[93] 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-December*, 5999–6009.

[94] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. (1).

[95] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.

[96] Lin, T., Wang, Y., Liu, X., & Qiu, X. (2021). A Survey of Transformers.

[97] Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training.

[98] Shirish Keskar, N., Mccann, B., Varshney, L. R., Xiong, C., Socher, R., & Research, S. (2019). CTRL: A Conditional Transformer Language Model for Controllable Generation.

[99] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2019). BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension, 7871–7880.

[100] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2019). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, *21*, 1–67.

[101] Weiss, K., Khoshgoftaar, T. M., & Wang, D. D. (2016). A survey of transfer learning. *Journal of Big Data*, *3*(1), 1–40.

[102] Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., & He, Q. (2021). A Comprehensive Survey on Transfer Learning. *Proceedings of the IEEE*, *109*(1), 43–76.

[103] Giorgi, J. M., & Bader, G. D. (2018). Transfer learning for biomedical named entity recognition with neural networks. *Bioinformatics (Oxford, England)*, *34*(23), 4087–4094.

[104] Peng, Y., Yan, S., & Lu, Z. (2019a). Transfer Learning in Biomedical Natural Language Processing: An Evaluation of BERT and ELMo on Ten Benchmarking Datasets. *BioNLP 2019 - SIGBioMed Workshop on Biomedical Natural Language Processing, Proceedings of the 18th BioNLP Workshop and Shared Task*, 58–65.

[105] Yang, L., & Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, *415*, 295–316.

[106] Wu, J., Chen, X. Y., Zhang, H., Xiong, L. D., Lei, H., & Deng, S. H. (2019). Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization. *Journal of Electronic Science and Technology*, *17*(1), 26–40.

[107] Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, *13*(10), 281–305.

[108] Feurer, M., & Hutter, F. (2019). Hyperparameter Optimization, 3–33.

[109] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2016). Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research*, *18*, 1–52.

[110] Zöller, M. A., & Huber, M. F. (2021). Benchmark and Survey of Automated Machine Learning Frameworks. *Journal of Artificial Intelligence Research*, *70*, 409–472.

[111] He, X., Zhao, K., & Chu, X. (2021). AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, *212*, 106622.

[112] Zimmer, L., Lindauer, M., & Hutter, F. (2021). Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl [also available under https://arxiv.org/abs/2006.13799]. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3079–3090.

[113] Microsoft. (2021). *Neural Network Intelligence* (Version 2.0).

[114] Ali, M. (2020). *Pycaret: An open source, low-code machine learning library in python* [PyCaret version 1.0].

[115] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and robust automated machine learning. *Advances in Neural Information Processing Systems 28 (2015)*, 2962–2970.

[116] Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M., & Smola, A. (2020). Autogluon-tabular: Robust and accurate automl for structured data.

[117] Jin, H., Song, Q., & Hu, X. (2019). Auto-keras: An efficient neural architecture search system. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1946–1956.

[118] Le, T. T., Fu, W., & Moore, J. H. (2020). Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics*, *36*(1), 250–256.

[119] LeDell, E., & Poirier, S. (2020). H2O AutoML: Scalable automatic machine learning. *7th ICML Workshop on Automated Machine Learning (AutoML)*.

[120] Baker, S., Korhonen, A., & Pyysalo, S. (2016). Cancer hallmark text classification using convolutional neural networks. *Proceedings of the Fifth Workshop on Building and Evaluating Resources for Biomedical Text Mining (BioTxtM2016)*, 1–9.

[121] Chen, Q., Chandrasekarasastry, N., Elangovan, A., Davis, M., & Verspoor, K. (2017). Document triage and relation extraction for protein-protein interactions affected by mutations.

[122] Aris, F., Baziotis, C., Pappas, D., Papageorgiou, H., & Potamianos, A. (2017). Hierarchical bidirectional attention-based rnn in biocreative vi precision medicine track, document triage task.

[123] Luo, L., Yang, Z., Lin, H., & Wang, J. (2017). Dutir at the biocreative vi precision medicine track: Document triage for identifying ppis affected by genetic mutations.

[124] Abdulkadhar, S., Murugesan, G., & Natarajan, J. (2017). Recurrent convolution neural networks for classification of protein-protein interaction articles from biomedical literature. *2017 Third International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, 192–197.

[125] Baker, S., & Korhonen, A. (2017). Initializing neural networks for hierarchical multi-label text classification. *BioNLP 2017*, 307–315.

[126] Du, J., Chen, Q., Peng, Y., Xiang, Y., Tao, C., & Lu, Z. (2019). ML-Net: multi-label classification of biomedical texts with deep neural networks. *Journal of the American Medical Informatics Association*, *26*(11), 1279–1285.

[127] 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).

[128] Xu, J., & Cai, Y. (2019). Incorporating context-relevant knowledge into convolutional neural networks for short text classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*(01), 10067–10068.

[129] Zhang, H., & Zhang, J. (2020). Text graph transformer for document classification. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 8322–8327.

[130] Ibrahim, M. A., Ghani Khan, M. U., Mehmood, F., Asim, M. N., & Mahmood, W. (2021). GHS-NET a generic hybridized shallow neural network for multi-label biomedical text classification. *Journal of Biomedical Informatics*, *116*, 103699.

[131] Jiang, T., Wang, D., Sun, L., Yang, H., Zhao, Z., & Zhuang, F. (2021). Lightxml: Transformer with dynamic negative sampling for high-performance extreme multi-label text classification.

[132] 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., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., . . . Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362.

[133] pandas development team, T. (2020). *Pandas-dev/pandas: Pandas* (Version latest). Zenodo.

[134] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

[135] Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with python: Analyzing text with the natural language toolkit*. " O'Reilly Media, Inc."

[136] Honnibal, M., & Montani, I. (2017). *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing* [To appear].

[137] Neumann, M., King, D., Beltagy, I., & Ammar, W. (2019). ScispaCy: Fast and robust models for biomedical natural language processing. *Proceedings of the 18th BioNLP Workshop and Shared Task*.

[138] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., . . . Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc.

[139] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., . . . Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45.

[140] Wang, W., Bi, B., Yan, M., Wu, C., Bao, Z., Xia, J., Peng, L., & Si, L. (2019). Structbert: Incorporating language structures into pre-training for deep language understanding.

[141] He, P., Liu, X., Gao, J., & Chen, W. (2020). Deberta: Decoding-enhanced bert with disentangled attention.

[142] Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2019). Superglue: A stickier benchmark for general-purpose language understanding systems.

[143] Beltagy, I., Lo, K., & Cohan, A. (2019b). Scibert: A pretrained language model for scientific text.

[144] Peng, Y., Yan, S., & Lu, Z. (2019b). Transfer learning in biomedical natural language processing: An evaluation of bert and elmo on ten benchmarking datasets.

[145] Gururangan, S., Marasović, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D., & Smith, N. A. (2020). Don't stop pretraining: Adapt language models to domains and tasks. *ArXiv*, *abs/2004.10964*.

[146] Gu, Y., Tinn, R., Cheng, H., Lucas, M., Usuyama, N., Liu, X., Naumann, T., Gao, J., & Poon, H. (2022). Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare*, *3*(1), 1–23.

[147] Bai, F., Ritter, A., & Xu, W. (2021). Pre-train or annotate? domain adaptation with a constrained budget.

[148] Yasunaga, M., Leskovec, J., & Liang, P. (2022). Linkbert: Pretraining language models with document links.

[149] Pyysalo, S., Ginter, F., Moen, H., Salakoski, T., & Ananiadou, S. (2013). Distributional semantics resources for biomedical text processing. *Proceedings of LBM 2013*, 39–44.

[150] Rao, V. S., Srinivas, K., Sujini, G. N., & Kumar, G. N. S. (2014). Protein-Protein Interaction Detection: Methods and Analysis. *International Journal of Proteomics*, *2014*, 1–12.

[151] Kuzmanov, U., & Emili, A. (2013). Protein-protein interaction networks: Probing disease mechanisms using model systems. *Genome Medicine*, *5*(4), 1–12.

[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., Altlnel, 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. *Database*, *2019*, 147.

[153] Altınel, B., Hüsünbeyi, Z. M., & Özgür, A. (2017). Classification using ontology and semantic values of terms for mining protein interactions and mutations.

[154] 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: The Journal of Biological Databases and Curation*, *2018*(2018), 1–12.