BIOMEDICAL ENGINEERING MSc PROGRAM
UNIVERSITY OF CRETE
TECHNICAL UNIVERSITY OF CRETE
FOUNDATION FOR RESEARCH AND TECHNOLOGY-HELLAS

Application Grade Thesis

EMPLOYING A BERT DEEP LEARNING MODEL TO DISTINGUISH CLASSES OF CLINICAL AND GENOMIC ENTITIES IN BIOMEDICAL TEXT

ΔΙΑΚΡΙΣΗ ΚΛΙΝΙΚΩΝ ΚΑΙ ΓΟΝΙΔΙΩΜΑΤΙΚΩΝ ΟΝΤΟΤΗΤΩΝ ΠΟΥ ΕΜΠΕΡΙΕΧΟΝΤΑΙ ΣΕ ΒΙΟΪΑΤΡΙΚΑ ΚΕΙΜΕΝΑ ΜΕ ΤΗ ΧΡΗΣΗ ΤΟΥ ΜΟΝΤΕΛΟΥ ΒΑΘΙΑΣ ΜΑΘΗΣΗΣ BERT

Giannakoula Amalia

Kanterakis Alexandros

23/05/2022

Acknowledgements

I would like to express my gratitude to my supervisor, Kanterakis Alexandros, who guided me throughout this project. I would also like to thank my friends and family who supported me during this period.

Abstract

Fast-paced changes occur in biomedical literature on a daily basis, leading to an increase of the extraction of essential information from different text corpora. Biomedical named entity recognition tasks require the use of deep neural network, in order to identify these entities and extract the requested features automatically, yielding state of the art performance. Even though the existence of text mining techniques using neural networks is widespread, it is not clear yet if in the comparison of biomedical and clinical entities, exist entities that can be strongly distinguished by the model.

The aim of this dissertation is to extract sentences containing specific classes of biomedical entities like Disease, SNP, Gene and Chemical, encrypted in excluded archives, compare them with each other using Bidirectional Encoder Representation for Transformers and examine if the model can distinguish the entities of each pair successfully.

The results showed that the model is capable of accomplishing great performance with adequate training, without extensive changes on its basic built-in core code. However, new challenges emerge daily, requiring greater optimization and fine-tuning techniques.

Περίληψη

Η ζήτηση για την απόκτηση κρίσιμων βιοϊατρικών δεδομένων και πληροφοριών μέσα από μια πληθώρα διαφορετικών κειμένων εντείνεται, καθώς η βιβλιογραφία που αφορά τη βιοϊατρική αυξάνεται καθημερινά. Οι μέθοδοι αναγνώρισης βιοϊατρικών οντοτήτων απαιτούν τη χρήση μοντέλων τεχνητών νευρωνικών δικτύων, τα οποία μπορούν να εντοπίσουν και να εξάγουν αυτόματα τα ζητούμενα χαρακτηριστικά, με τη μέγιστη δυνατή απόδοση. Ωστόσο, παρόλο που η ύπαρξη τεχνικών εξαγωγής κειμένου που χρησιμοποιούν νευρωνικά δίκτυα είναι ευρέως διαδεδομένη, δεν είναι ακόμη σαφές εάν κατά τη σύγκριση μεταξύ βιοϊατρικών και κλινικών οντοτήτων, υπάρχουν οντότητες που εντοπίζονται εντονότερα από το εκάστοτε μοντέλο.

Ο στόχος της παρούσας διατριβής είναι να εξαγάγει προτάσεις που εμπεριέχουν συγκεκριμένες κατηγορίες βιοϊατρικών οντοτήτων όπως Ασθένειες, Πολυμορφισμός ενός νουκλεοτιδίου, Γονίδια και Χημικές ενώσεις, οι οποίες βρίσκονται κρυπτογραφημένες σε ειδικά διαμορφωμένα αρχεία, να τις συγκρίνει μεταξύ τους χρησιμοποιώντας μια τεχνική μηχανικής εκμάθησης βασισμένη σε μετασχηματιστές (Bidirectional Encoder Representation for Transformers) και να εξετάσει εάν το μοντέλο μπορεί να διακρίνει με επιτυχία αυτές τις οντότητες σε κάθε ζεύγος σύγκρισης.

Τα αποτελέσματα των πειραμάτων έδειξαν πως το μοντέλο αποδίδει εξαιρετικά μόνο με την βασική εκπαίδευση, χωρίς να έχουν εφαρμοστεί κρίσιμες αλλαγές στον βασικό κορμό του κώδικα του. Ωστόσο, οι νέες προκλήσεις που εμφανίζονται καθημερινά, απαιτούν επιπλέον τη χρήση τεχνικών βελτιστοποίησης του κώδικα για μέγιστη απόδοση.

Table of contents

Acknowledgements 2
Abstract 3
Περίληψη 4
1. Introduction 10
2. State of the art 11
2.1 Machine learning 11
2.1.1 Encoder – Decoder architecture 13
2.2 Natural language processing (NLP) 13
2.2.1 Categories of NLP tasks 14
2.3 Biomedical natural language processing 14
2.3.1 Challenges of clinical and biological language processing 15
2.4 BioC format 15
2.4.1 BioC workflow 16
2.4.2 Understanding of BioC structure 17
2.5 Machine learning and Natural language processing 18
2.5.1 Attention mechanism 20
2.5.2 Transformer architecture 20
2.5.3 Bidirectional Encoder Representations from Transformers (BERT) 28
2.5.4 DistilBERT 31
2.6 Evaluating machine learning model performance 32
2.6.1 Useful terms 32
2.6.2 Model classification metrics 32
2.7 Jupyter Notebook 33
2.8 Colaboratory (Colab) 33
3. Research methodology 35
3.1 The problem 35
3.2 Acquisition of the dataset 35
3.3 Processing phase 41
3.3.1 Explaining of the codes 43
4. Presenting and analyzing the findings 46
4.1 SNP vs Disease, Gene and Chemical 46
4.2 Disease vs Gene 52
4.3 Disease vs Chemical 54
4.4 Gene vs Chemical 56
4.5 Execution time 58
5. Conclusions and recommendations 59
References 61

List of figures

Figure 1 - Importance of gradient 12
Figure 2 - Feed forward neural networks 12
Figure 3 - Encoder/Decoder Architecture 13
Figure 4 - BioC workflow 16
Figure 5 - Example of BioC file 17
Figure 6 - Example of BioC file 18
Figure 7 – Polysemy for the word “crane” 18
Figure 8 - LSTM model 19
Figure 9 - Attention mechanism 20
Figure 10 - Transformer architecture 20
Figure 11 - Encoder and Decoder components 21
Figure 12 - Encoder design 21
Figure 13 - Decoder design 22
Figure 14 - Word embedding 22
Figure 15 - Embedding layer 23
Figure 16 - Parallel execution 23
Figure 17 - Positional vector 24
Figure 18 - Query/Key/Value 24
Figure 19 - Importance of each word of the input sentence 25
Figure 20 - Query/Key/Value matrices 25
Figure 21 - Z matrix calculation 26
Figure 22 - Multi-head attention mechanism 26
Figure 23 - Normalization step 27
Figure 24 - Transformer architecture 27
Figure 25 - Computational power needs 28
Figure 26 - BERT masked language modelling architecture 29
Figure 27 – Superiority of bidirectionality 30
Figure 28 - Next sentence prediction model 30
Figure 29 - Example of a confusion matrix table 33
Figure 30 - Pip command 34
Figure 31 - Colab's layout 34
Figure 32 - Selection of runtime environment based on user's preference 34
Figure 33 - Specifications 36
Figure 34 - Dataset layout 41
Figure 35 - Specifications 42
Figure 36 – Shuffled sample 43
Figure 37 - Tokenization of the sentences 44
Figure 38 - SNP vs Disease Evaluation diagram 46
Figure 39 - SNP vs Gene Evaluation diagram 46
Figure 40 - SNP vs Chemical Evaluation diagram 47
Figure 41 - SNP vs Disease Accuracy diagram 48
Figure 42 - SNP vs Disease Precision diagram 48
Figure 43 - SNP vs Disease Recall diagram 49
Figure 44 - SNP vs Gene Accuracy diagram 49
Figure 45 - SNP vs Gene Precision diagram 50
Figure 46 - SNP vs Gene Recall diagram 50
Figure 47 - SNP vs Chemical Accuracy diagram 51
Figure 48 - SNP vs Chemical Precision diagram 51
Figure 49 - SNP vs Chemical Recall diagram 52
Figure 50 - Evaluation diagram 52
Figure 51 - Accuracy diagram 53
Figure 52 - Precision diagram 53
Figure 53 - Recall diagram 54
Figure 54 - Evaluation diagram 54

Figure 55 - Accuracy diagram 55
Figure 56 - Precision diagram 55
Figure 57 - Recall diagram 56
Figure 58 - Evaluation diagram 56
Figure 59 - Accuracy diagram 57
Figure 60 - Precision diagram 57
Figure 61 - Recall diagram 58
Figure 62 - Mean execution time 58
Figure 63 - Summary of total evaluation 59

List of tables

Table 1 - BioC file layout 17
Table 2 - Comparison between BERT/DistilBERT 31
Table 3 - Accuracy/Precision/Recall at the maximum number of sentences for the three cases 47
Table 4 - Comparison between the 3 performance scores 55

1. Introduction

Biomedical named entity recognition (BioNER) intents to systematically identify biomedical entities (some of which are diseases and chemicals) in given sequences. Successful recognition of biomedical entities is considered as a prerequisite of extracting biomedical knowledge, hidden in unstructured texts, since it helps into converting them to structured compositions. This ability indicates the importance of the research value of BioNER methods.

BioNER tasks require pioneering feature engineering, like tools and knowledge of many natural languages processing (NLP). However, over the last few years, the use of neural networks is closely connected with named entity recognition (NER) tasks. In comparison with feature engineering methods, neural networks are able to automatically find out relationships and patterns between the words and thus can accomplish more competitive performance.

The existing techniques describe the BioNER task as a sequence of labeling problem, where a model is trained for assigning a label to each token in a given text. There are many models (like BioALBERT or Spacy) that can achieve name entity recognition tasks, however the scope of this approach is to distinguish between all the examined entities and locate the easiest to find.

The methodology of the aforementioned approach includes the following steps. Firstly, the creation of a code using Python in Jupyter, in order to decipher the text, since the majority of biomedical entities are stored into specific archives, called BioC files. Second, the use of BERT code in order to verify in what extent these entities can be recognized by the trained neural network. Lastly, the comparison between the examined entities will be evaluated using performance scores.

The dissertation is divided into five chapters, where each one of them presents a thorough analysis of the approach.

Chapter 1: Introduction, where the reader understands the basic axis of the document
Chapter 2: State of the art, where the literature review is presented
Chapter 3: Research methodology, where the developed methodology is demonstrated
Chapter 4: Research results, where the results are presented, discussed and analyzed
Chapter 5: Conclusions and recommendations

2. State of the art

2.1 Machine learning

Machine learning is a sub-field of computer science and artificial intelligence (AI), imitates humans' learning ability by using a variety of data and algorithms [20]. Through the use of statistical methods, its algorithms are trained in order to gradually improve the accuracy of the model.

Machine learning is divided into **three sub-categories**:

Supervised learning: *the algorithms are trained using labeled datasets. The model adjusts its weights, in order to fit properly in the input data. Meantime, the prevention of overfitting or under-fitting is ensured through the cross-validation process.*

The typical supervised machine learning algorithm consists of the following three main components:

- A decision process: *calculates the input data to return an assumption over the kind of the expected pattern, that the algorithm is looking for.*
- An error function: *measures how accurate "the guessing" was by comparing it to known examples.*
- An optimization process: *updates the final decision of the decision process so as to achieve a smaller loss.*

Unsupervised learning: *as an input it takes unlabeled data, which are analyzed and clustered. The algorithm discovers hidden patterns, similarities and differences without human intervention.*

Semi-supervised learning: *uses a smaller labeled dataset to guide feature extraction and classification.*

Deep learning is a sub-field of machine learning, which uses automations to extract features, eliminating the human factor. It can use larger unstructured datasets in their raw form and distinguish them into categories.
Neural networks use series of algorithms, in order to mimic the behavior of the human brain. This ability allows to computer programs, the recognition of relationships and patterns between vast amounts of data and at the same time, the ability to solve many common problems in the field of AI.
An input layer, one or more hidden layers and an output layer are the cornerstones of every neural network. For every defined input, the corresponding weights are assigned. These weights are critical, because they determine the significance of any given variable, meaning that the greater the value, the more it contributes to the output. At the end, all inputs are multiplied by their weights, giving the following formula:

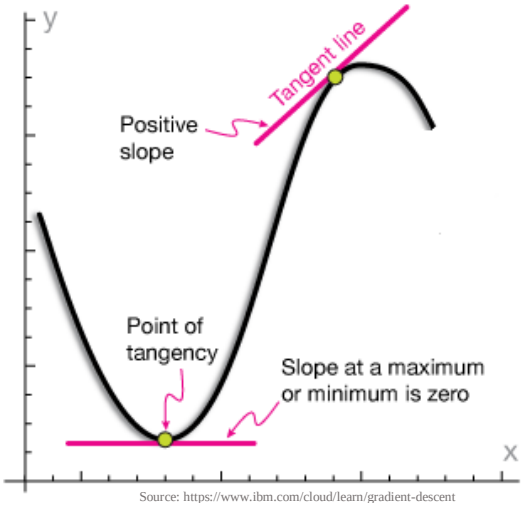
$$\sum_{i=1}^{m} w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias \qquad Eq.1$$

where,
x, *the input variables*
w, *the weights*
m, *the total number of the input nodes*
bias, *a systemically prejudiced result due to inaccurate assumptions in the machine learning process*

Each neuron, with its associated weight, is connected with another node. The activation of the node is interwoven with the output. If the output exceeds the pre-defined threshold value, then the neuron sends data to the next layer of the network, whereas if the output does not exceed the pre-defined threshold value, no information is passed along to the next layer.

Feed forward neural networks took their names from the direction they channel information. In a feed-forward neural network, the information can only flow in one way - starting from the input layers, through the hidden layers, at the output layer. In this kind of network the information never reach the same node twice. The feed-forward neural network appoints a weight matrix to its inputs and then generates the output (see *Figure 2*). A drawback of this network is the lack of memory of the input, which prevent the network to execute accurate predictions.
Before moving on, it is crucial to explain the importance of the meaning of gradient to a machine learning model.

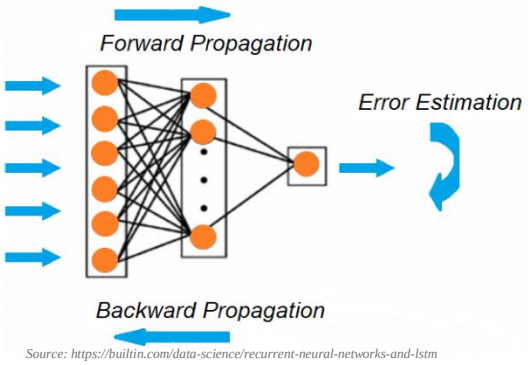
Gradient: *A partial derivative that expresses how much a little change in the input can affect the output. In a diagram the gradient depicts the slope of the function.*



Source: https://www.ibm.com/cloud/learn/gradient-descent

Figure 1 - Importance of gradient

The steeper the slope the faster the learning process of the model is. However, zero slope means that the model stops learning (see *Figure 1*). In a machine learning problem, the gradient measures the change in the weights with respect to the change in error.
The gradient of an error function with respect to the weights of the neural network is calculated using backpropagation. This algorithm is looking for the partial derivative of the errors with respect to the neural networks' weights, by going backwards through the various layers of gradients. Then it uses them to decrease the error margins during the training phase.
Backward propagation is an algorithm used for the training of feedforward neural networks. Backpropagation computes the gradient of the loss function with respect to the weights of the network. Back propagation through time, mainly means that the error is back-propagated from the last to the first time step. This technique allows the calculation of the error of each time-step, leading to new, updated weights. However, this algorithm can be proven computationally expensive, as the number of time steps increases.

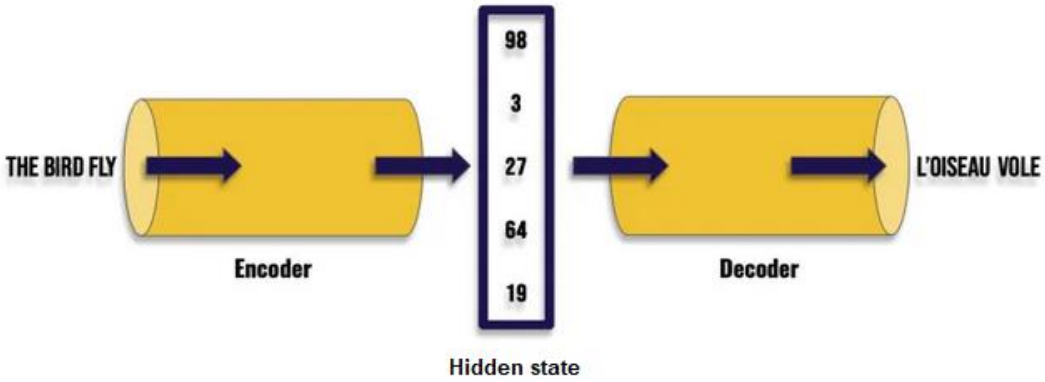


Source: https://builtin.com/data-science/recurrent-neural-networks-and-lstm

Figure 2 - Feed forward neural networks

2.1.1 Encoder – Decoder architecture

An **encoder** is a neural network that takes the input and provides as an output a feature map/vector/tensor. These vectors hold the information ·the features· that represents the input.
A **decoder** is again a network (usually has the same network structure as encoder but it works in the opposite orientation) that takes the feature vector from the encoder, and gives the best closest match to the actual input or intended output.
The encoders are trained with the decoders. The loss function is based on computing the delta between the actual and reconstructed input. The optimizer will try to train both encoder and decoder to lower this reconstruction loss. Once trained, the encoder will give feature vector for input. Then, these vectors can be used by decoder to construct the input with the features that matter the most, so as to make the reconstructed input recognizable as the actual input.
There is an intermediate plane between encoder and decoder that is called hidden state. Hidden state is basically the output of the encoder ·a two-dimensional vector that encapsulates the whole meaning of the input sequence. Its length depends only on the number of cells.
The two architectures are depicted in *Figure 3*.



Source: *https://www.researchgate.net*

Figure 3 - Encoder/Decoder Architecture

2.2 Natural language processing (NLP)

Natural language processing is a branch of computer science that began in the 1950s. NLP aims to create algorithms with the ability to understand and respond to text or voice data, similarly with a human being. NLP combines statistics and deep learning models with computational linguistics - rule - based modeling human language, in order to process and "understand" the full meaning of human language, when it is in the form of written text or voice data [16].
Writing software for determining the expected meaning of text or voice data accurately is quite difficult, since human language is filled with ambiguities (e.g. sarcasms, idioms, metaphors etc.). Therefore, several NLP task have been created, in order to break them down into simpler forms that can be easily understandable from computers.
The first approaches into the development of these tasks were simplistic so soon they failed, due to homographs. Also, same other NLP limitations described below lead to the rise of statistical NLP.

- Handwritten rules can become unmanageably numerous, leading to unpredictable interaction after multiple parses.
- Handwritten rules cannot ·handle with a high success· rate ungrammatical spoken prose, even if it is human-comprehensible.
- Each NLP task requires different types of data, leading to a variety of challenges.

Klein made a fundamental reorientation in this field in 1980 [7], having as a result the use of statistical methods in NLP. The use of probabilities leads to better results, since statistical approaches, by learning with a mass of real data, can utilize the most common cases. Statistical approaches utilize

the fact that some choices in the grammar are more favorable than other. Moreover, they degrade smoothly if the input is erroneous or unfamiliar.
Statistical NLP governs by a well-known principle: ***"The more abundant and representative the data, the better they get"*** [1].

2.2.1 Categories of NLP tasks

The NLP tasks are divided into two categories: low-level and high level tasks.

The following list includes low-level NLP tasks:

- Detection of the boundaries of the sentences
- Part of speech toggling
- Tokenization: *refers to the separation of the sentences into smaller entities called tokens*.
- Segmentation of text into smaller and meaningful classes/bodies
- Morphological decomposition of compound words
- Identification of patterns into constituent part-of-speech (e.g. an adjective sequence is followed by a noun.)

The second category is listed below:

- Identification and reconstruction of spelling or grammatical error
- Name entity recognition(NER): *the task of detecting instances of mentions of some semantic classes in text*
- Word sense disambiguation: *verifying the correct meaning of a homograph*
- Relationship extraction: *detecting connections between events or entities. The task of extracting information about a specified type of relationship from free text, is called relation extraction. An example of relation extraction is protein-protein interactions, since almost all proteins act as members of a network.*

Text mining applications are used as a tool in NLP applications, because they can perform tasks in order to meet information needs (e.g. extracting statements by labelling parts of speech). Processing information, writing the appropriate software for each application and using algorithms in order to perform tasks on datasets of various types are some of the tasks of text mining [5].

2.3 Biomedical natural language processing

When NLP deals with biomedical text, then it is called biomedical NLP.

Clinical documents and scientific journals are the main document structures of interest in biomedical NLP. However, these document types are more complicated in form, in contrast with the newswire text. An article body and an abstract are the two main sub-categories that a scientific journal is typically separated into. The differences in synthesis, content and structure of these two categories lead to a difference in performance of the extracting tools. On the other hand, there is variability in the type of clinical documents, making them difficult to be easily parsed. Finding the boundaries between sentences constitutes an even more difficult task in clinical texts, since the boundaries of the sentences are not clearly defined. Taking into consideration all the above, NLP application is crucial to have the ability to appropriately segment these documents.
NER in biomedical NLP differs, since there is diversity in the semantic types that need to be analyzed.

The most common categories of name entities are presented in the list below.

- Cell lines
- Cell types
- Chemicals
- Drugs
- Genes/proteins

- Malignances
- Disorders
- Mutations
- Diseases

NLP applications, in biomedical domain open up a new era of functionalities and abilities for health care domain, helping in the management of texts of large volumes [9].
The following list presents NLP applications in biomedicine:

- Information extraction: *detecting the critical information of the text, even without executing a thorough analysis*
- Information retrieval: *scientific literature usually inhabits inside large collections, which are quite difficult to be parsed*. This task is crucial in biomedicine, since it tries to match a user's query against the collection of documents and return the most akin.
- Text generation: *natural language sentences are formulated, leading to generation of text from structure data(e.g. compiling patterns or/and trends)*
- User interfaces: *tools that enable humans to communicate more adequately with computer systems*
- Machine translation: *the task, with which one language is converted into another*.

2.3.1 Challenges of clinical and biological language processing

NLP applications in biomedical domain face some challenges, since the complexity of the text that need to be procced is greater [9]. These challenges are presented below:

- Good performance: *every output of NLP applications used in biomedicine should be characterized by accuracy, sensitivity and specificity. However, each clinical application requires varying levels of performance.*
- Regaining implicit information
- Intraoperability: *the NLP system has to have dual role, handling a variety of different interchange formats associated with the different kinds of reports, generating outputs that can be easily stored in a currently existed clinical achieve.*
- Interoperability: *NLP applications are used by a variety of institutions, each one of them using its own vocabulary. So, if the needed vocabulary for an application exceeds the "standard", then an extra vocabulary is needed.*
- Development of NLP systems with the help of training sets
- Evaluation process is crucial, however a gold standard is difficult to be obtained, since it requires the exchange of data and information across institutions
- Heterogeneous forms: *no standardized format/structure within the reports*
- Considerable amount of different clinical domains
- Manifestation of spelling or /and typographic errors
- Rare events
- Biomolecular names with dubious nature
- The number of biomolecular entities is quite large
- Variability in names: *names are created within an institute, so they differ in different databases*
- Long names containing substrings of other names
- Difference in schemas or/and taxonomies of each community
- Heterogeneity of the source of the text
- Complexity of each texts/reports language

2.4 BioC format

As was mentioned in chapter *2.3.1 Challenges of clinical and biological language processing* , one of the challenges of biomedical NLP is the heterogeneity in formats of the reports and texts. Due to this situation, the interchange of data across different NLP applications is obstructed. For this reason,

some tools have to be developed, in order to facilitate the seamless integration of the data. These tools should operate from just a token to even the whole text. These tools should also have a common format that will be flexible enough to allow the integration of annotations and at the same time the extension of text mining infrastructure.

The goal of this common format is based on the following principles:

- Interoperability
- Simplicity
- Broad use
- Reuse

A common evaluation framework was developed based on a collaborative initiative, called BioCreative [13]. This new text-mining tool serves a dual purpose:

- can be used by general biology researchers and more specialized end users and
- generates shared gold standard datasets that can be used for training and/or testing of text-mining applications

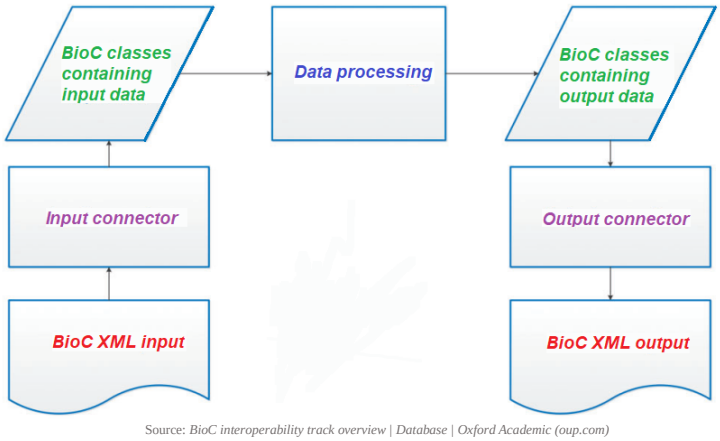
BioC is a tool for sharing biomedical text, annotations and libraries in a simple XML format. More specifically, BioC is a text format that contains relations between annotations [8]. In this tool there are also some libraries included. These libraries help users to achieve their goal, by reading and writing data into common programming languages. The choice of XML format was based on the simplicity and good attestation of it. One other advantage of this format is that it can handle both encodings and character sets, existed in a biomedical text.

2.4.1 BioC workflow

BioC design is analyzed in a simple workflow, which contains 3 basic components [7]:

- Data input
- Data processing
- Data output

In
Figure 4, the workflow of the BioC design is depicted.



Source: *BioC interoperability track overview | Database | Oxford Academic (oup.com)*

Figure 4 - BioC workflow

First, the input data that are written in BioC format are passing into the phase of BioC data classes via an Input Connector. The input data can be either a file or a network stream written in XML format. Then, the Data Processing module executes the desired NLP or text mining process. When the final desired output is ready, it is passed by the BioC classes containing the output module to the end user via the output connector module.

2.4.2 Understanding of BioC structure

BioC is considered as a flexible NLP tool since it fulfills the following 3 main requirements:

- Easily represented in common programming languages
- Easily recorder in a file format
- Portability between different operating environments

Although, is quite easy to grasp the idea of an XML file, someone needs additional information in order to fully understand it. A BioC file uses key files, composed by the author, in order to communicate its information and specify details on how to interpret this file. A key file explains the meaning of tags and at the same time includes important information about the text, like the included character set and encoding or the entities annotated [6] (see *Table 1*).

Table 1 - BioC file layout

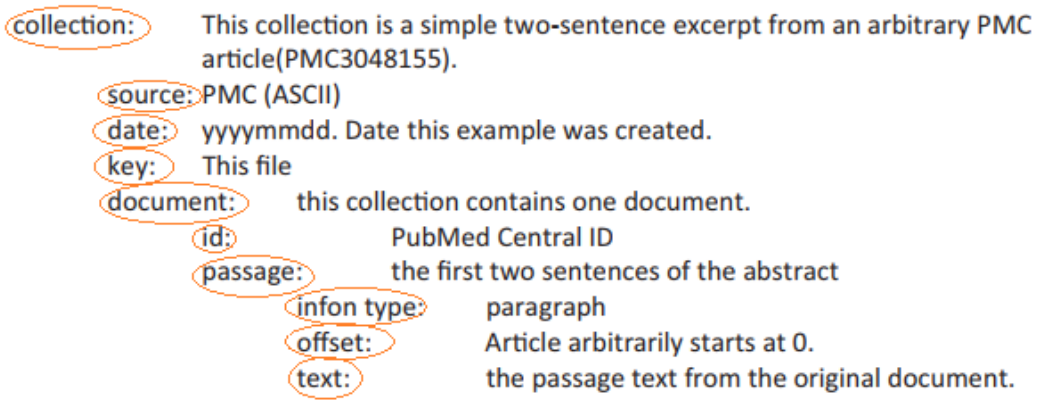
Element	Description
Collection	A group of archives or papers, extracted by a source
Source	The name of the corpus or the source, from where the documents were retrieved
Key	A separate plain text describing and explaining the details of the BioC XML file
Date	The date that the documents were obtained from their original source. It may be in every acceptable format, since it is described in key file
Infon	Key-value pairs that encompass essential information. Attribute: Key: a unique characteristic of each element
Document	A single stand-alone document inside the collection
id	Identification of each document, connecting it with the parent source
Passage	A part of the whole document
Offset	The position of the element in the document
Text	The text of each element
Sentence	A sentence of a document
Annotation	Stand-off annotation, referred to by relations
Location	It is the location of the annotated text.

An example of A BioC file appears on *.Figure 5 and* Figure 6 .

```
<!DOCTYPE collection SYSTEM "BioC.dtd">
<collection>
 <source>PubMed Central</source>
 <date>20130123</date>
 <key>exampleCollection.key</key>
 <document>
  <id>PMC3048155</id>
  <passage>
   <infon key="type">paragraph</infon>
   <offset>0</offset>
   <text>The efficacy of computed tomography (CT) screening for early lung cancer detection in
heavy smokers is currently being tested by a number of randomized trials. Critical issues remain
the frequency of unnecessary treatments and impact on mortality, indicating the need for
biomarkers of aggressive disease.</text>
  </passage>
 </document>
</collection>
```

Source: *BioC interoperability track overview | Database | Oxford Academic (oup.com)*

Figure 5 - Example of BioC file



Source: *BioC interoperability track overview | Database | Oxford Academic (oup.com)*

Figure 6 - Example of BioC file

2.5 Machine learning and Natural language processing

The use of a pre-trained model is a solution that can be applied to accomplish a variety of NLP tasks like sentence prediction or text summarization. Training a separate model exclusively for each NLP task every time, is highly time consuming and equipment cost consuming. So, the general idea is to develop a model that is trained in a vast amount of data. Later, this pre-trained model will be used for each NLP task after fine tuning. However, the development of such models requires the conversion of the words into numbers that can be easily understood by the computer system. So, many algorithms were created in order to covert words into vectors. This process is generally called "word embeddings".Word2vec is one of the earliest algorithms created for this purpose [15]. The problem of this algorithm was that it was context-free, meaning that it cannot handle polysemy. For example, the word "crane" has multiple meaning according to the context. It can mean "lifting machine" or "bird" (see *Figure 7*). However, in Word2vec model the word "crane" would only relate to one meaning returning the same embedding.

Figure 7 – Polysemy for the word "crane"

To overcome the aforementioned problem, context-based models were created. One such was created using Convolutional Neural Networks (CNN). CNNs are usually used in image processing. In Image processing, there is a relation between nearby pixels, so CNNs are used, so as to understand this correlation. Exactly the same logic can also be applied for processing different NLP tasks. As far as the case of texts is concerned, CNNs use a matrix, where each row represents a word and the number of each column depicts the embedding size you need. Given that it is an n*m matrix, then "n" columns denotes "n" words that have been given as input to the model and "m" represents the dimension of the embedding. Moving on, filters are used to extract features. The drawback of this approach is that the relation between tokens/words in a sentence is not similar to pixels, since the meaning between words is not defined only by their position.

Moving on, using Recurrent Neural Networks (RNNs), sequential models were developed. These models are a sub-category of artificial neural networks with the ability to handle the input data ·the words· sequentially one by one, from both directions, either from right to left or left to right. Also, RNN's may consist of both an encoder and a decoder. The encoder accepts two inputs ·the word and the contextual vector. Contextual vector is responsible for capturing the contextual relation of a word, with the number of the generated vectors to depend on the hidden states of the encoder. The input word, after being converted into embedding, outputs a context vector which is back propagated. Finally, after the whole sentence is processed, the final vector is ready to pass to the decoder. The decoder uses this output as input in order to generate the targeted sentence [14].
Nonetheless, there are two obstacles that could not be overcome with the RNN's built in code and led to the development of more complex neural networks:

- Exploding gradients
- Vanishing gradients

Exploding gradients: *when the algorithm assigns, without particular reason, higher importance, that it should do, to the weights. This problem can be solved by truncating the gradients.*

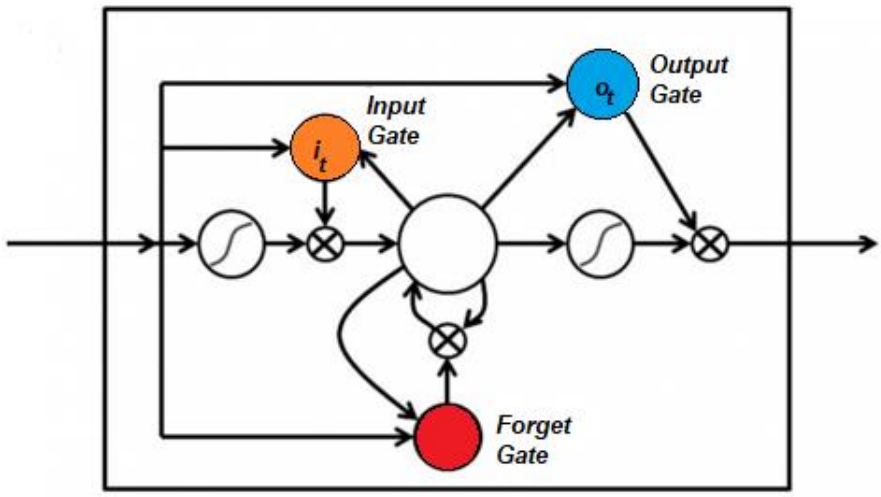
Vanishing gradients: *when the learning process of a model stops, because the gradients are too small. This problem can be solved using the concept of Long Short-Term Memory.*

Basically, Long Short-Term Memory (LSTM) networks solve the problem of vanishing gradients, since they are an extension of RNNs' memory. They used as the building units for the layers of a RNN, organizing the data based on their importance. The LSTM model has the ability to write, read, remember inputs over a long period of time or even delete information from its memory. The importance of each input is defined through the learning process using weights.

This memory can be considered as a gated cell, which consists of three gates:

1. Input
2. Forget
3. Output

These gates follow the form of sigmoid, so their range is [0,1].
The layout of a LSTM model can be seen in *Figure 8*.



Source: https://builtin.com/data-science/recurrent-neural-networks-and-lstm

Figure 8 - LSTM model

The drawback of these models is that they also have shortcomings, mainly after 2 or more layers.

2.5.1 Attention mechanism

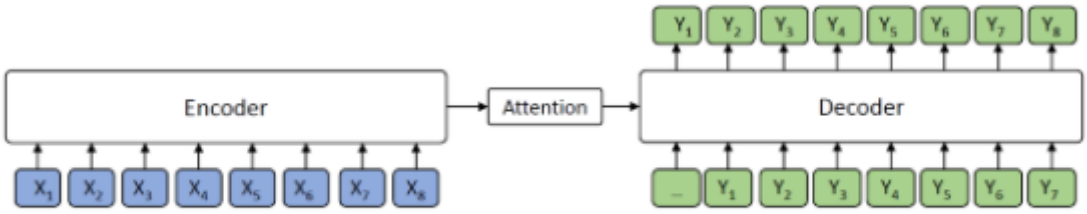
The problem of LSTM models was overcome by using the attention mechanism. An attention mechanism is used between the encoder and decoder block (for machine translation). Using attention mechanism is like emphasizing the important words in the input. Moreover, this mechanism helps in the recognition of the alignment between the input and output. The context vectors, generated in the hidden states of the encoder, are the inputs to this mechanism.

Attention mechanism is divided into two categories ·Global attention and Local attention. As Global attention is defined the attention mechanism where the output of all hidden states is passed to attention, while as Local attention is defined the system where only a subset of the hidden state output is passed to attention [14].

The effectiveness of such a mechanism can be calculated using a simple attention score, as described below. Firstly, the value of the output of the hidden state of the encoder is passed to the decoder in order to produce an initial decoder hidden state value (DHS1). After that the following procedure takes place:

1. The calculation of the dot product between each of the hidden state of the vector of the encoder with the DHS1.
2. The calculation of the average number of each vector in order to shrink the dimensions(if it is needed)
3. The normalization of the above-derived number using softmax
4. The multiplication of each of the encoder hidden state vectors with the above normalized score, creating the alignment score
5. Adding all the alignment vectors in order to generate the context vector

The attention mechanism is depicted on *Figure 9.*



Source: *The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.* (jalammar.github.io)

Figure 9 - Attention mechanism

2.5.2 Transformer architecture

Transformer is a model that uses the attention mechanism so as to boost the training speed of a neural machine application [3]. Also, its ability of parallel processing is considered as the biggest advantage of this method.

For the better explanation of the model, a machine translation application will be used, in order to provide a high-level look and a better understanding of the mechanism.

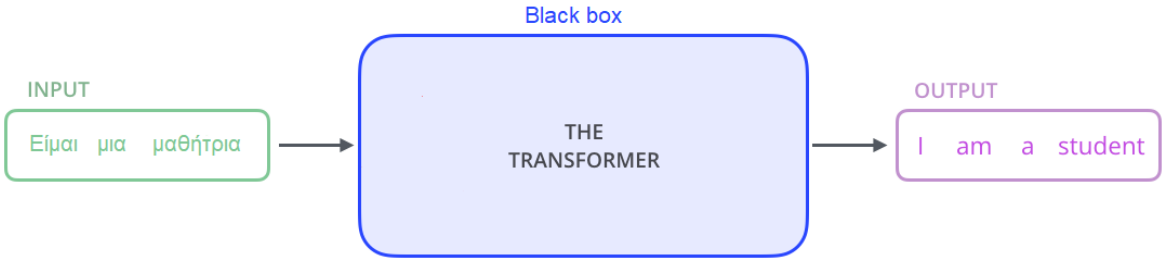


Figure 10 - Transformer architecture

By looking inside the *Figure 10*, someone can clearly see an encoding component, a decoding part and the connection between them. The encoding component includes a pile of encoders (the original paper mentions 6 of them, however the experiment with other arrangements is allowed). The part of decoder has the same number of piles as the encoder (see *Figure 11*).

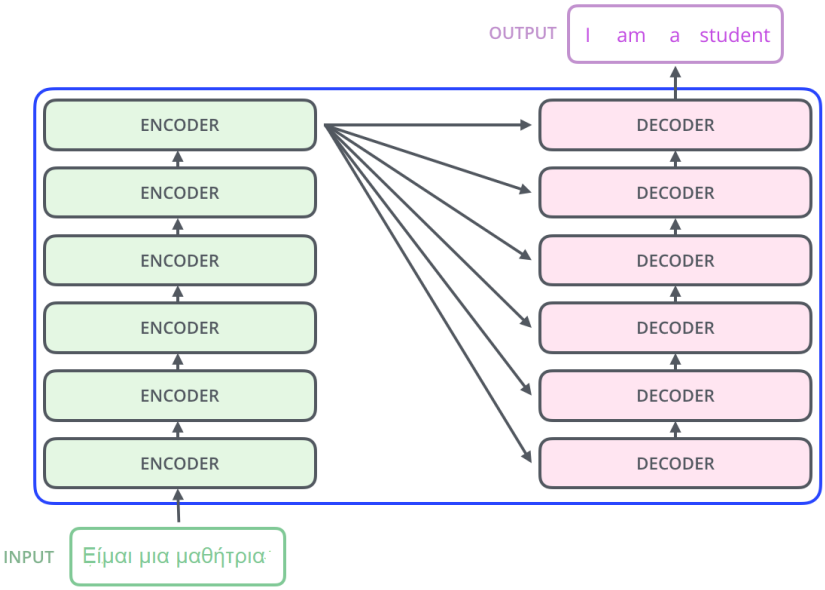
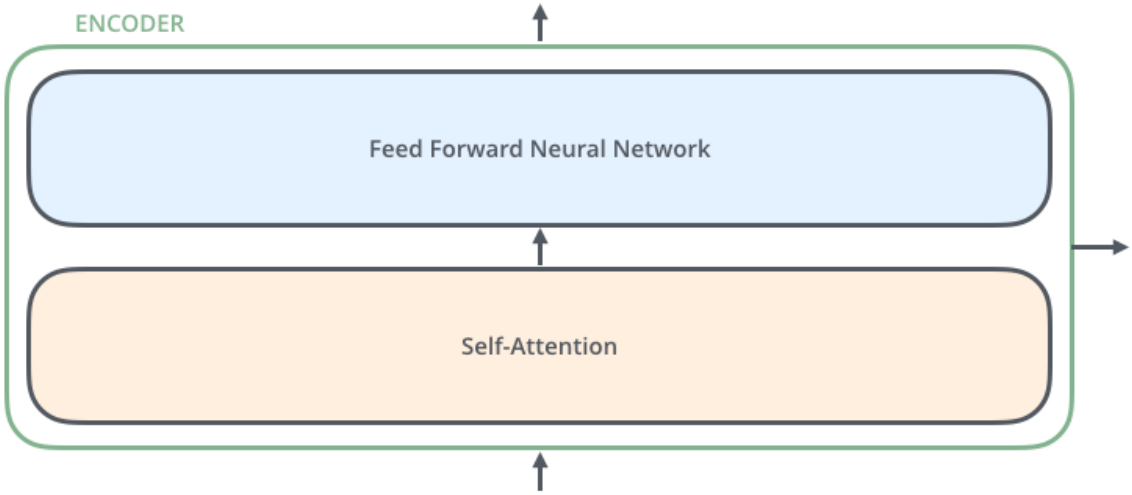


Figure 11 - Encoder and Decoder components

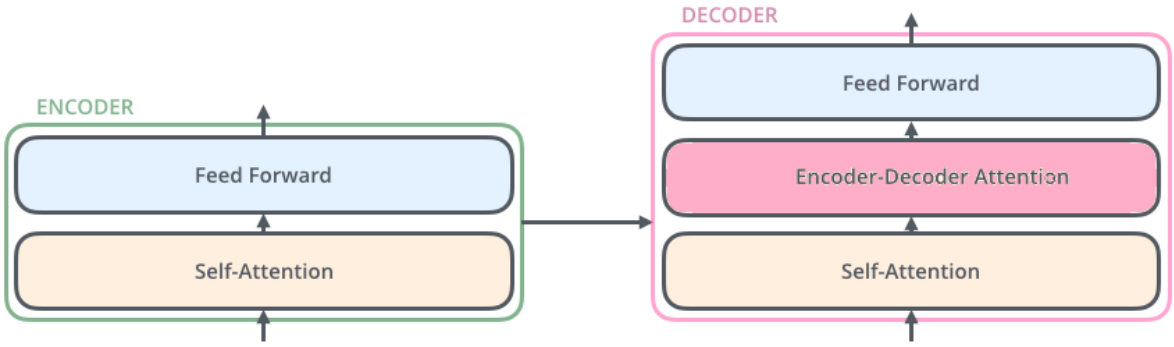
Each encoder is identical and is divided into two sub categories:



Source: *The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.* (jalammar.github.io)

Figure 12 - Encoder design

When the input enters the encoder, first it flows through a self-attention layer. The relationship between the words of each input sentence is complex so the attention model cannot fully decode it. Another mechanism, called "self-attention" (*more information about this method is going to be given below*) is used for this purpose. After the self-attention layer, the output is fed to a feed forward neural network. This procedure repeats for each individual position (see *Figure 12*).
Moving on to the decoder part depicted on *Figure 13*, someone can observe that it consists of both the aforementioned layers, with the only difference to be one extra attention layer, which helps the decoder to focus on the important parts of the input sentence.



Source: *The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.* (jalammar.github.io)

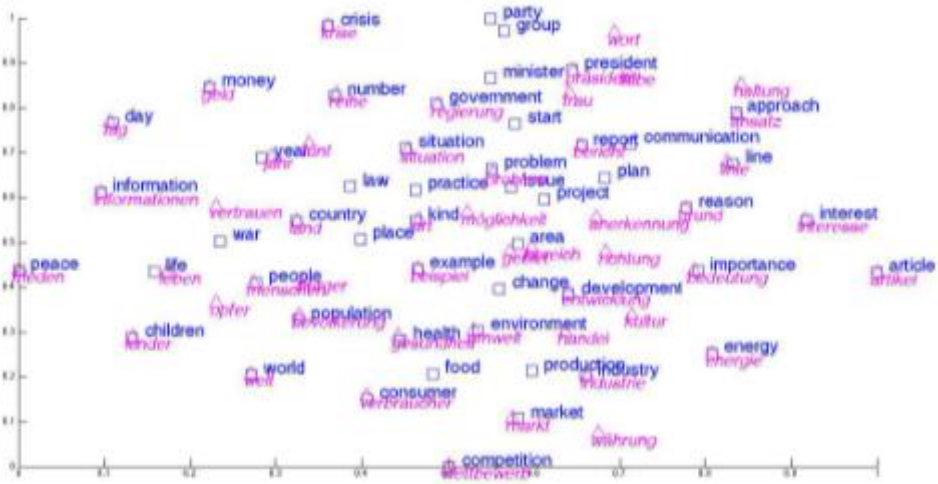
Figure 13 - Decoder design

2.5.2.1 Word embedding theory

Computers can only understand numbers, vectors and matrixes. NLP applications require also their input to be vectors. So, the transformation of the input words into entities that are understandable by the computer is necessary. For this reason a technique called "Word Embedding" was developed in order to solve the aforementioned issue [10]. Word Embedding is a natural language modeling method that is used to map words or phrases to a corresponding vector of real numbers. This vector has two crucial properties:

- Dimensionality Reduction
- Contextual Similarity

Word Vectors are vector representations which use a much lower dimensional space. These vectors are used for semantic parsing. Meanwhile, during their creation, they preserve some similarities, meaning that words that can regularly be found nearby in text (see *Figure* 14), they are also close in vector space. So, Word embedding is a method aims to create a low-dimensional vector representation from collection of texts, while at the same time try to preserve the contextual similarity of the used words.



Source: *Luong et al*

Figure 14 - Word embedding

General, in NLP applications, the first action is the transformation of each word to a representative vector, using the Word embedding technique. The embedding only happens in the encoder pile.

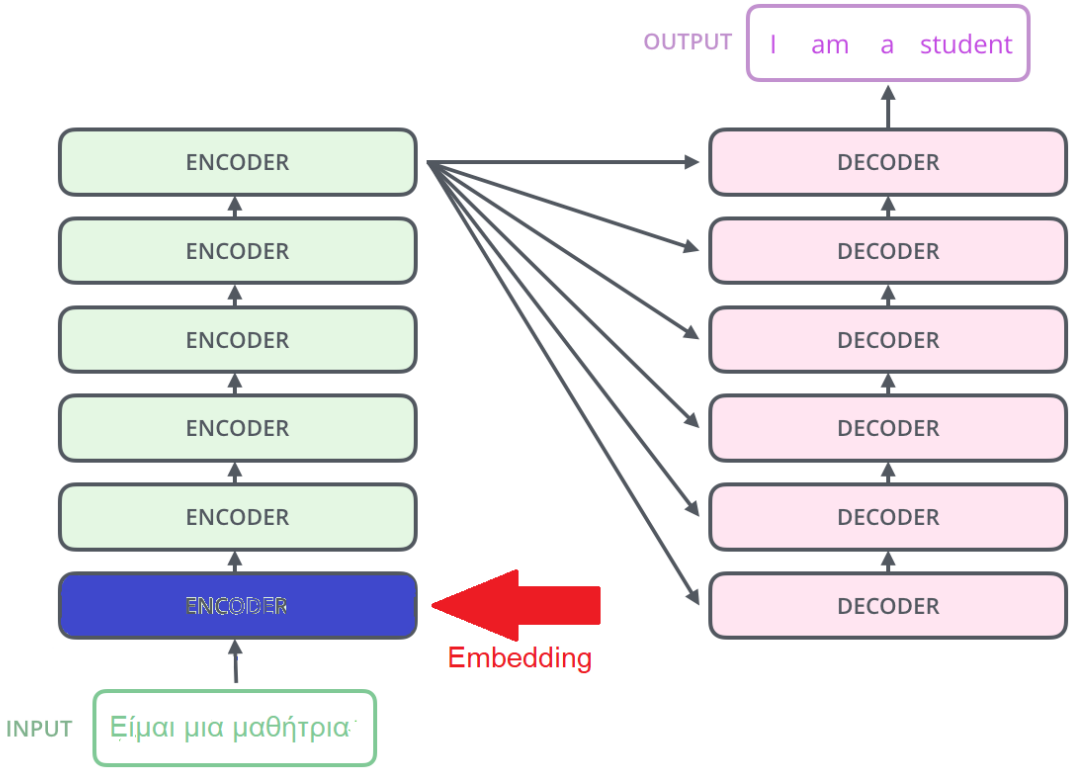


Figure 15 - Embedding layer

The general principle, common to all encoders, is that they receive a list of vectors each of a size 512 ·this size is a hyper parameter that can be set by the developer, and basically it represents the length of the longest sentence of the training set (see *Figure 15)*. After embedding, each word's representation vector flows through the Self-Attention and then Feed-Forward layer of the encoder. As can be seen in the *Figure 16*, the vector in each position flows through its own path, which reveals a key property of the Transformers. There is an important difference between the two layers. In the first layer, there are dependencies between the paths of each vector, while all the vectors' paths are independent, in the second one. This difference allows the parallel execution of the feed-forward networks.

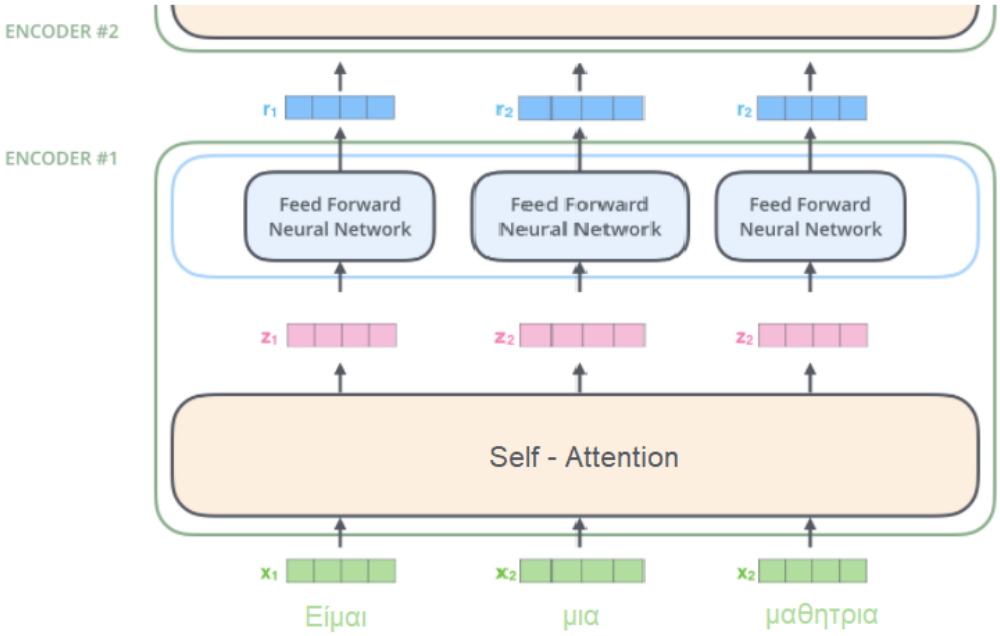


Figure 16 - Parallel execution

The order of the words in the input sentence is defined by adding a vector to each input embedding. These vectors follow a specific pattern and determine either the position of each word either the distance between different words in the sequence (see *Figure 17*).

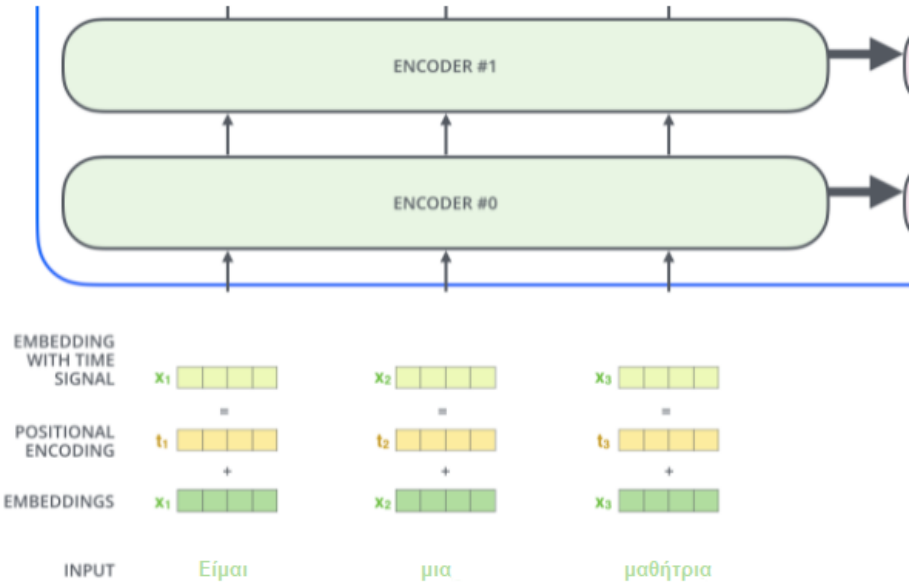
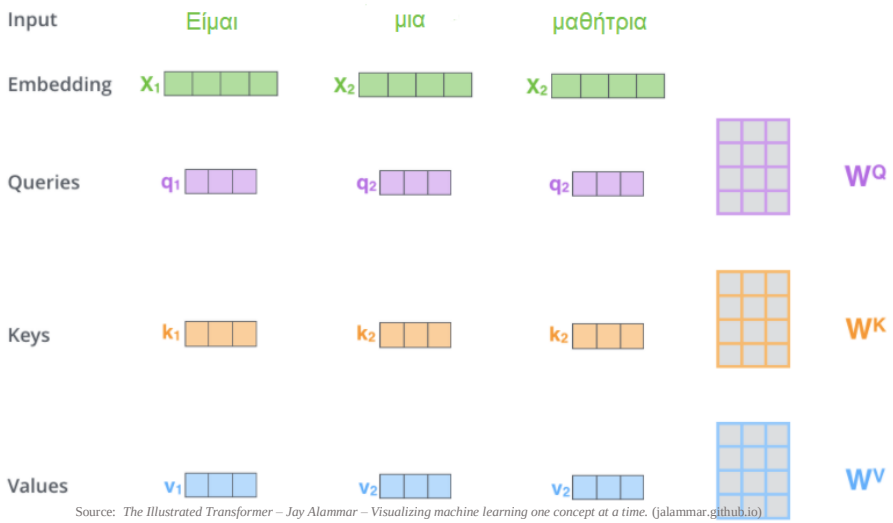


Figure 17 - Positional vector

2.5.2.2 Self-Attention mechanism

The self-attention mechanism is chiefly used in machine reading algorithms, where it tries to investigate and derive the relationship that exists between the words of the input sentence. The implementation of this method is presented below:

1. The creation of three vectors – Query (Q), Key (K), and Value (V) - from each of the encoder's input vectors. These vectors are formed by the multiplication of the embedding and the three matrices that have been created during the training phase of the model (*Figure 18*).



Source: *The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.* (jalammar.github.io)

Figure 18 - Query/Key/Value

2. The calculation of the score that determines the importance of each word in the input sentence. The score is calculated by taking the dot product of the key and query vectors. *Figure 19* explains the procedure in depth.

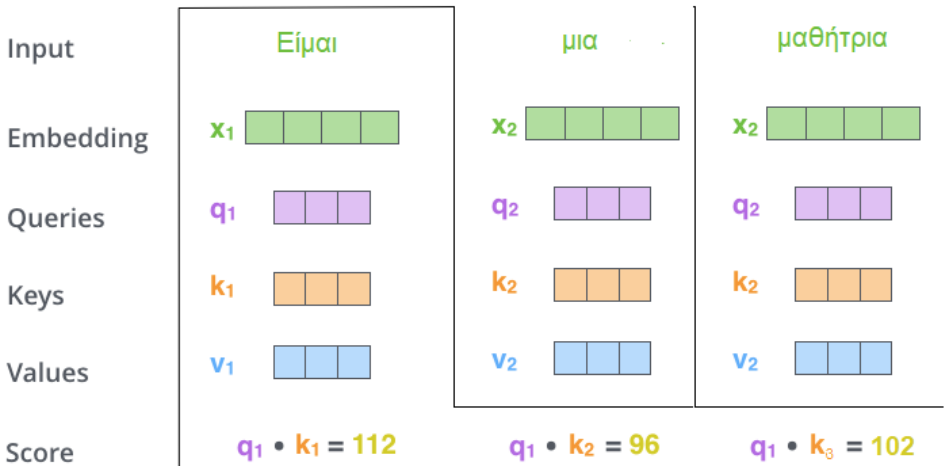


Figure 19 - Importance of each word of the input sentence

For example, by processing the word in the position #1, three scores are calculated. The same procedure is followed until the words in all positions are processed.

3. The division of each score by 8. This number occurs by the square root of the dimension of the key vector, as it is written in the original paper of Transformers. However, it could take different values.
4. The previous result is passed through a softmax operation, where it is normalized. The degree of expression of each word at the specific position is determined by this softmax score.
5. Each value vector is multiplied by the previous score, so as to exclude the irrelevant values.
6. Sum up the previous vectors (Z vectors).

The self-attention calculation is completed after these 6 steps and the output can now pass to the feed-forward neural network.
For simplicity reasons, all the previous analysis was explained using vectors. However, in reality the calculation is done using matrices, in order to accelerate the process, as depicted on *Figure 20* and *Figure 21*.

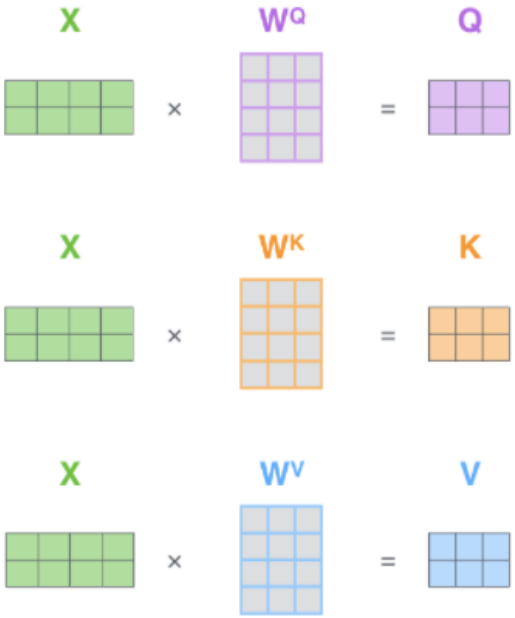
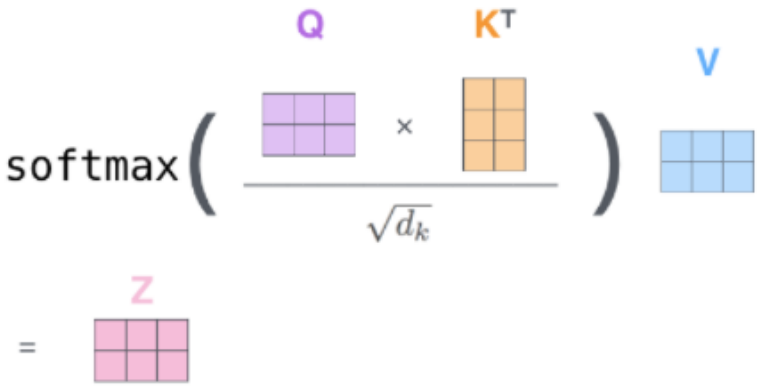


Figure 20 - Query/Key/Value matrices

Source: *The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.* (jalammar.github.io)



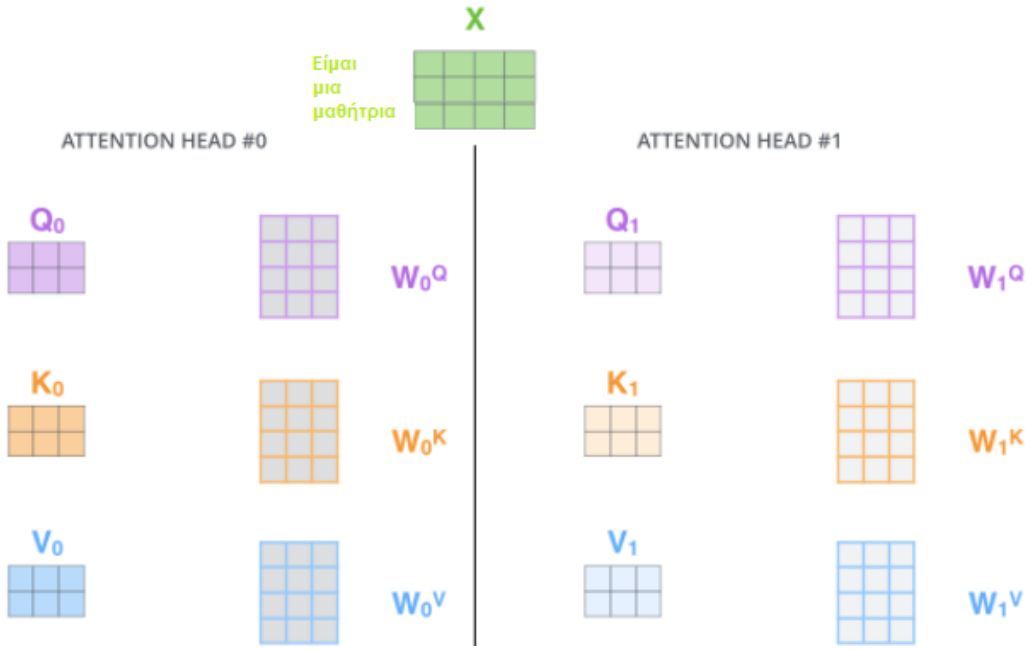
Source: *The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.* (jalammar.github.io)

Figure 21 - Z matrix calculation

2.5.2.3 Multi-headed attention

As multi-headed attention, is called the mechanism that uses 8 different Query, Key and Value matrices for each encoder. The advantages of this method are:

- The model's capability to focus on different positions is expanded.
- Different subspaces are created.

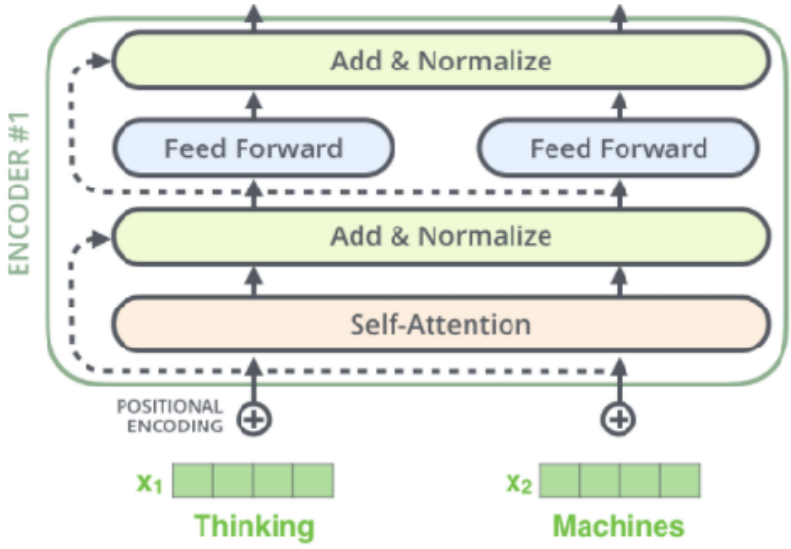


Source: *The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.* (jalammar.github.io)

Figure 22 - Multi-head attention mechanism

Separate Query, Key and Value matrices are maintained for each head, using the multi-head attention (see *Figure 22*). So, by following the aforementioned analysis, different Z matrices are generated. However, the feed-forward network can only accept one matrix. This matrix is created from the concatenation of the Z vectors of each attention head and the multiplication of them by an additional weight matrix.

Before moving on to the decoder part, it is important to note that, each sub-layer of each encoder is followed by a normalization step, as can be seen on *Figure 23*.

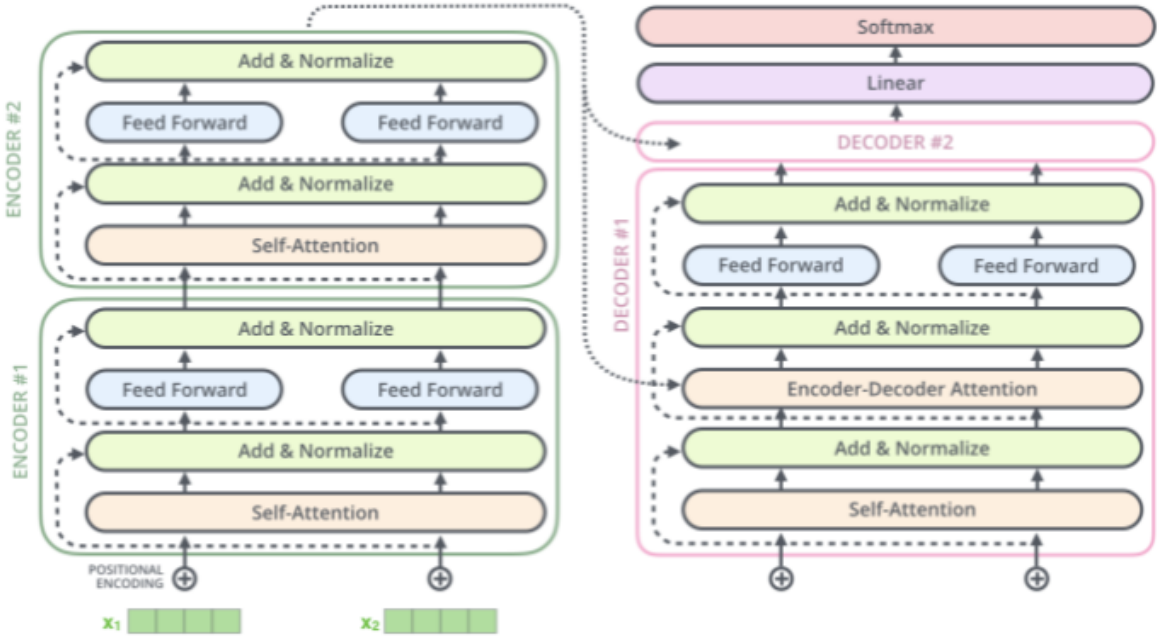


Source: *The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.* (jalammar.github.io)

Figure 23 - Normalization step

After finishing the encoding phase, the decoding phase begins. Decoder part is alike the encoder. The first decoder accepts as input the output of the final encoder, which is transformed into a set of attention vectors (or matrices) (K and V), that help the decoder to focus on the important parts of the input. The output of each step is fed to the next decoder until the appearance of a specific symbol that ends the process. There is also positional encoding to the decoder input, so as to indicate the position of each word.

The self-attention layer operates slightly different in the decoder. It is only allowed to attend to earlier positions in the output array. This is achieved by masking future positions before the softmax step. As far as the Encoder-Decoder Attention layers is concerned, it operates similarly with the multi-headed self-attention, with the only difference that it generates its Query matrix from the layer below it, while the Keys and Values matrices are taken from the output of the encoder stack. The final layers are responsible for the transformation of the output to its final form (see *Figure 24*).



Source: *The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.* (jalammar.github.io)

Figure 24 - Transformer architecture

2.5.3 Bidirectional Encoder Representations from Transformers (BERT)

BERT, which stands for Bidirectional Encoder Representations from Transformers, is a new method for NLP applications (like Classification or Question-Answering etc.) that presents state of the art results. BERT's innovation lies within its code, where it exploits the architecture of Transformers in order to generate a language model. As have already been mentioned, Transformers are divided into two separate mechanisms - an encoder that accepts and deciphers the input and a decoder that generates a prediction for the examined task. However, BERT needs only the encoder mechanism for its tasks [4].

As opposed to directional models that read the input text sequentially (right to left or left to right), BERT can edit all the words of the input sentence at the same time. This ability makes the model directionless and more flexible, by allowing it to learn the framework of a word, based on its surroundings.

BERT only accepts the input into a particular format. So, the input sentence should pass through a specific modification, before processing. Some extra embeddings are used for this purpose:

i. One token called [CLS] is added at the beginning of every input, indicating the start of the sentence
ii. Tokens called [SEP] are added at the end of the input for the separation of the sentences
iii. Positional embeddings
iv. Segment embeddings

BERT has been distributed with four different versions:

- The basic one(BERT base), which contains 12 layers with 12 attention heads
- The extended (BERT Large), which contains the same number of layers as the basic, whereas the number of attention heads increases to 16
- BERT Cased
- BERT Uncased

In this point it is important to be noted that the size of the model it considers as a critical parameter (see *Figure 25*). BERT Large, which consists of 345 million parameters, is the largest model of its kind. Even though it demonstrates superiority even on small-scale tasks, the needed processing power is exponentially increased.

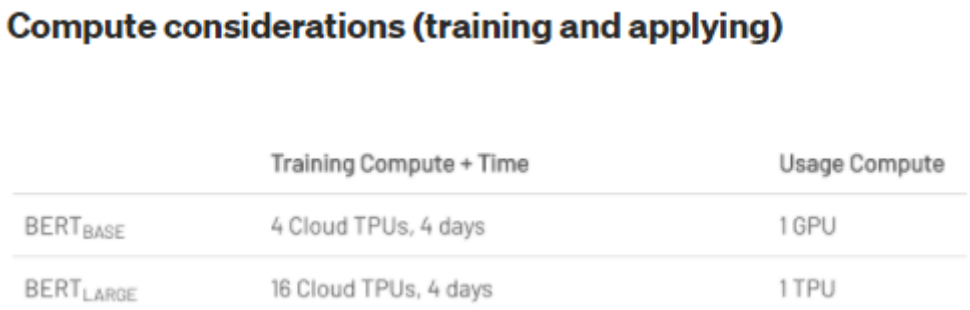


Figure 25 - Computational power needs

2.5.3.1 Training phase

Language models are trained in order to achieve a prediction goal. In the majority of them the challenge lies to the prediction of the next word in the sequence. For example:

" **Το παιδί κάθε πρωί πηγαίνει στο _____** "

However, this approach is applied in directional models, leading to limits in context learning. As opposed to this, BERT uses two training strategies:

- Masked language modelling
- Next sentence prediction

Basically, these strategies are unsupervised predictive tasks.

2.5.3.1.1 Masked Language Modelling (MLM)

Before passing the inputs into BERT, 15% of the words of each sentence are replaced with a token called [MASK].
This percentage (15%) corresponds to pre-processed tokens. The analysis of this percentage is described below.

- 80% (out of 15%) of the words are replaced with a [MASK] token
- 10% of them are replaced with a random word
- the remaining words are not replaced, since the original word is used on them

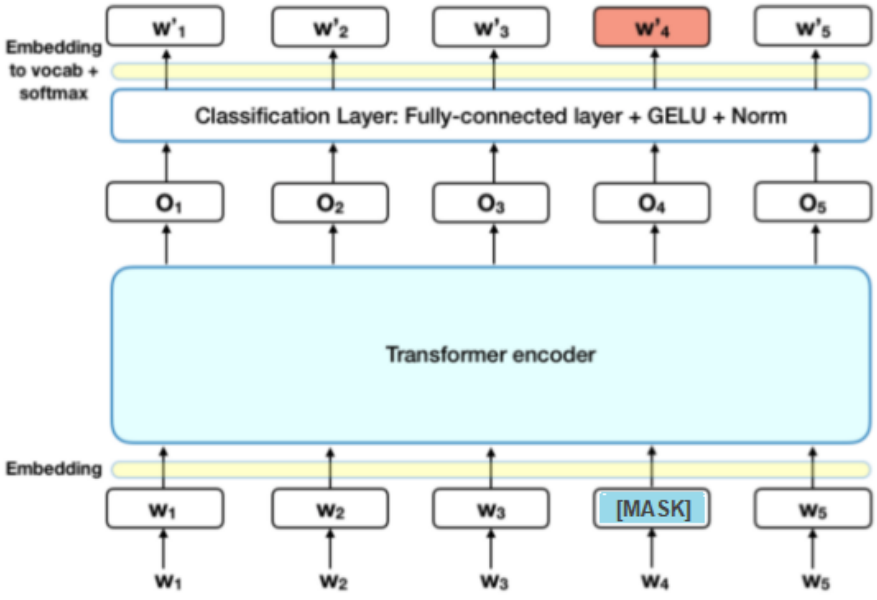
This technique is used for specific reasons.

- If the [MASK] token was used 100% of the time, then the produced tokens wouldn't necessarily be good representations for the non-masked words. This design would lead to false optimization of the model.
- If the [MASK] token was used 90% of the time, and the remaining percentage was random words, then this would lead to a model, where the observed word is never correct.
- If the [MASK] token was used 90% of the time, and the remaining percentage of words are kept the same, then the model could just copy the non-contextual embedding.

After the completion of the replacement, the model tries to predict the original value of the masked words, based only on the information that perceives by the surroundings. This is a quite demanding task that requires the following adjustments as far as the technical part is concerned.

i. A new classification layer is added, on top of the encoder output
ii. The multiplication of the output vectors with the embedding matrix, in order to be transformed into the vocabulary dimension
iii. The calculation of probabilities for every word with softmax

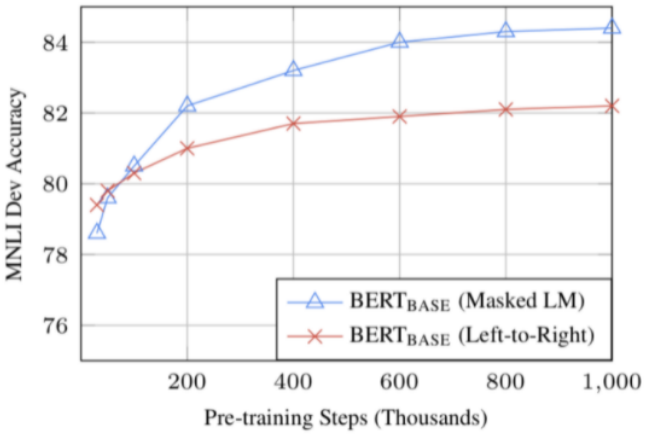
BERT masked language modelling architecture is presented on *Figure 26*.



ttps://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp

Figure 26 - BERT masked language modelling architecture

It is important to be noted that, the loss function calculates only the prediction of the masked values, while at the same time neglects the prediction of the non-masked words, leading to a slower convergence and increased awareness.
At the beginning, BERT's bidirectional approach (in MLM) converges slower than left-to-right approaches, however after some small number of training steps it surpasses it (see *Figure 27*).



Source: BERT [Devlin et al., 2018]

Figure 27 – Superiority of bidirectionality

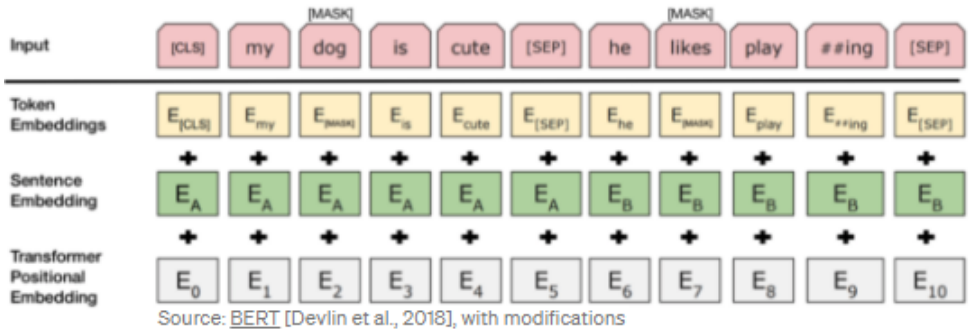
2.5.3.1.2 Next Sentence Prediction

During training phase, BERT model is fed with pairs of sentences as input and by learning tries to predict if the second sentence in the pair follows the first sentence in the original text. In the training process, 50% of the samples are a pair in which the second sequence is the subsequent sequence in the original corpus, whereas in the other 50%, the second sentence is a random sentence from the collection. There is a hypothesis here, that the random sentence will be detached from the first sentence.

In technical terms, some modifications should be done in the input, in order to help the model distinguish the sentences.

i. The [CLS] token and the [SEP] token are added in their pre-defined positions
ii. Each sentence will be indicated using a sentence embedding (like Sentence A and Sentence B), which will be added to each token.
iii. A positional embedding is added to each token to pinpoint its position in the sequence.

Next sentence prediction model is depicted on *Figure 28*.



Source: BERT [Devlin et al., 2018], with modifications

Figure 28 - Next sentence prediction model

To predict if the second sentence follows the first, the following steps are executed:

i. The Transformer model is fed with the entire input sequence.
ii. A 2x1 shaped vector is created from the output of the [CLS] token by a classification layer

iii. Calculating the probability using softmax

During the training process of BERT, these two strategies aforementioned above are executed together. The goal is to minimize the combined loss function.
One other important factor, for a variety of tasks, is the size of the training set is. For example, on Natural Language Interference (MNLI) tasks, if the training process is executed on 1 million steps rather than 500K steps, then the accuracy can be improved by 1.0%, using the same BERT model for both cases. Having said that, more training steps equal to higher accuracy.

2.5.3.2 Fine-Tuning

BERT is undeniably a breakthrough in the use of Machine Learning for NLP. The only thing that needs to be done is some small changes in the code in order to be appropriate modified for every case.

- Classification tasks such as sentiment analysis are executed similarly to Next Sentence classification. The only difference is an extra layer that is added on top of the Transformer output for the [CLS] token.
- When the software receives a specific question in Question-Answering applications, it should mark the appropriate answer in the sequence. Using BERT this task is more optimized, since two extra vectors, which indicate the beginning and the end of sentence that includes the answer are added.
- In Name Entity Recognition (NER) task, the software is called to pinpoint the various types of entities, exist inside the received text sequence. Using BERT, the output vector of each token of a NER model can be feed, after training, into a classification layer that can later predict the label.

In the fine-tuning training most hyper-parameters remain the same.

2.5.4 DistilBERT

The developer of the code is the team of Hugging Face. DistilBERT code is based on the knowledge distillation, a technique in which a model, objectively small, is trained in order to reflect the behavior of a larger model. Using the teacher-student training, they achieved to train the student network (DistilBERT) to mimic the output characteristics of the teacher network (BERT). The training setup involved eight 16GB V100 GPUs and lasted almost three and a half days. During the training phase, DistilBERT used very large batches leveraging gradient accumulation, with dynamic masking, while at the same time the next sentence prediction objective was removed. More specifically, it contains only half of the layers from BERT. Its performance was tested on the development sets of GLUE [17] and is presented on *Table 2*.

Table 2 - Comparison between BERT/DistilBERT

	BERT	**DistilBERT**
Size(millions)	Base: 110 Large: 340	Base:66
Training time	Base: 8 x V100 x 12 days	Base: 8 x V100 x 3.5 days
Performance	Outperforms state-of-the-art in Oct 2018	3% degradation from BERT
Data	16GB BERT data 3.3 billion words	16GB BERT data 3.3 billion words
Method	BERT	BERT Distillation

2.6 Evaluating machine learning model performance

Machine learning has become essential to our daily lives, since it is used in a variety of cases. So, accurate and trustworthy predictions are expected to be provided when using machine learning models.

Before moving on, it is important to obtain a general understanding of some useful terms in machine learning.

2.6.1 Useful terms

Training set: *is referred to a subset of a dataset used to build predictive models. The training set includes a set of input examples that will be used to train a model by adjusting the parameters of the set.*

Validation set: *is a subset of a dataset, whose purpose is to assess the performance of the model, during the training phase. Validation set periodically evaluates the model and allows for fine-tuning of its parameters (not all modeling algorithms need a validation set).*

Test set: *basically is the final evaluation that a model undergoes after the training phase. A test set is a subset of a dataset used to assess the possible future performance of a model.*

2.6.2 Model classification metrics

Classification is about predicting the class labels of a given input dataset [2].

Accuracy

$$Accuracy = \frac{Total\ number\ of\ correct\ predictions}{Total\ number\ of\ all\ predictions} \qquad Eq.2$$

The perfect accuracy equals to 1. Attention is needed when the dataset is imbalanced, where accuracy score can be very misleading.

Precision

$$\text{Pre}cision = \frac{True\ Positive}{True\ Positive + False\ Positive} \qquad Eq.3$$

Precision corresponds to the purity of the machine learning output. It answers to the question: ***How many elements exist, that they shouldn't be there?***

Recall

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \qquad Eq.4$$

Recall, on the other hand, corresponds to lost data. The question here is: ***How much data should be taken into calculation but they are not?*** If the recall score equals 1, it means that the model has not overlooked anything.

All the scores were calculated using confusion matrix. Confusion matrix is a summarized table, where the values of each correct and incorrect prediction have been counted and then summarized and broken down based on their class [18].
Figure 29 shows the picture of the structure of a 2x2 confusion matrix, where the classification model predicted "YES", where the actual value was "YES".

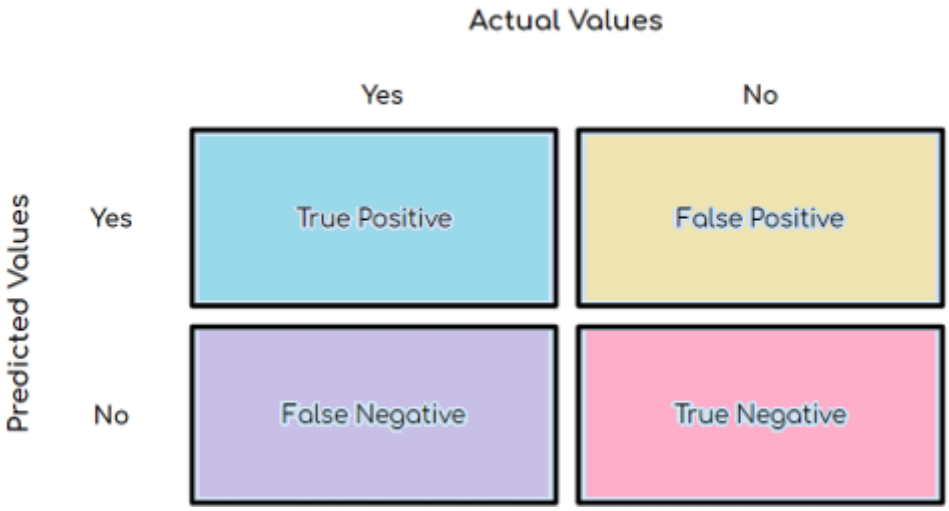


Figure 29 - Example of a confusion matrix table

Positive: *The observation is positive*
Negative: *The observation is not positive*
True Positive: *The model correctly predicts the positive class*
True Negative: *The model correctly predicts the negative class*
False Positive: *The model incorrectly predicts the positive class when it is actually negative (Type 1 error)*
False Negative: *The model incorrectly predicts the negative class when it is actually positive (Type 2 error)*

2.7 Jupyter Notebook

The Jupyter Notebook is an open source web application that can be used in order to create and share documents. Jupyter Notebooks are a spin-off project from the IPython project. The name, Jupyter, comes from the core supported programming languages that it supports: Python, Julia and R [14].
Notebook documents are documents produced by the Jupyter Notebook App. These documents contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, etc.). These documents can be both human-readable documents as well as executable documents.
The Jupyter Notebook App is a server-client application that permits editing and running notebook documents via a web browser. This app can be executed on a local desktop requiring no internet access. Alternative can be installed on a remote server and accessed through the internet.
The Jupyter Notebook App has a <Dashboard> or else it is called a <control panel> that shows the local files and allows to open notebook documents or shutting down their kernels. A notebook kernel basically executes the code contained in a Notebook document. Jupyter supports the IPython kernel, meaning that it allows the execution of programs in Python. By opening a Notebook document, the associated kernel is automatically launched. When the notebook is executed, the kernel carries out the computation and displays the results. Depending on the type of computations, the kernel might preoccupy significant CPU and RAM. The RAM is only released when the kernel is shut-down [19].

2.8 Colaboratory (Colab)

Colaboratory (Colab) [11] is basically an online Jupyter Notebooks environment developed by Google. Its capabilities are the same or even higher from the level of a local notebook, with the main difference to be that it is in the cloud. This feature makes Colab quite competitive, since no software installation is needed and also it is available from any computer that has internet connectivity. Moreover, it is free of use and no setup is required. To use Colab, the only thing needed is a Google account. However, the free subscription has some limitations, concerning the memory and the computer power that offers. All notebooks are executed into a virtual machine which will

automatically terminate if it has been idle for a while. Also, there is a time limit (almost 12 hours) that the virtual machine is running.
Colab runs its notebooks in Python 3 at present, including some of the most popular Python libraries that are already pre-installed. At the same time, if the user wants to add an additional package, he/she should only use the *pip* command in a notebook cell, as presented on *Figure 30*.

```
!pip install transformers
```

Figure 30 - Pip command

This package will remain active until the end of the session. The created notebooks are stored in the Google Drive associated with the user's Google account. Moreover, the users can upload their own local notebooks on Colab.

Colab's layout is pretty similar to a standard Jupyter Notebook, as can be seen on *Figure 31*.

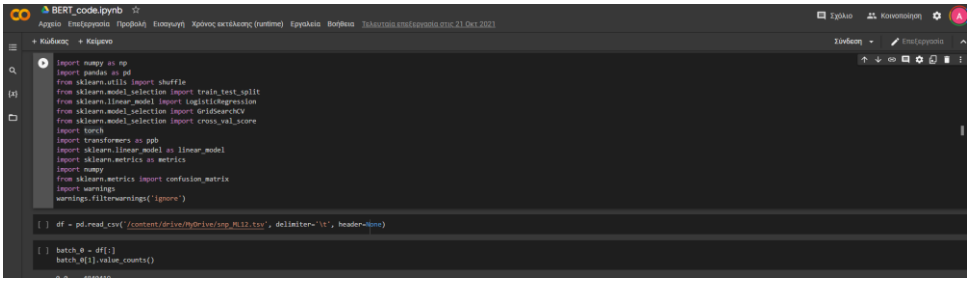


Figure 31 - Colab's layout

There is no kernel menu in Colab as there is only one kernel to choose from. Restarting the kernel is done from the Runtime menu. Also, one very useful feature of Colab notebook is the panel of variables, where the user can see the types and values of each of the current variables.
Runtime environments are the fundamental idea behind the development of Colaboratory. Colab is mostly used, because it can easily handle GPU intensive tasks, like training deep learning models. It is basically a tool that gives access to a GPU or TPU. The user can easily select the environment of his/her preference by going to the *Runtime* tab and select *Change runtime type (see* Figure 32*).*

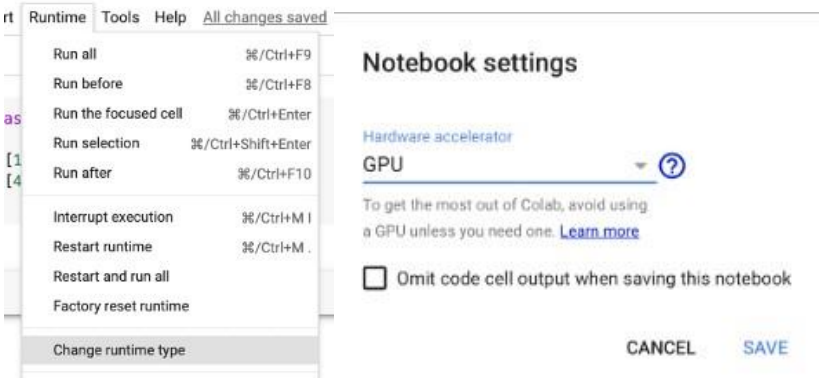


Figure 32 - Selection of runtime environment based on user's preference

Attention. There is no guarantee that the user will get the same GPU every time, as there is the factor of randomness in between.

3. Research methodology

This chapter presents the methodology that was used during this thesis. However, the real question is· **What exactly is this dissertation going to address and resolve?**

3.1 The problem

Nowadays, there is a plethora of methods for classification of sentences and entities using deep neural network frameworks. However, it is still an open question if an entity can be distinguished more easily when comparing to other entities. This statement is going to be examined in this thesis.
The examined classes of this approach are 4, in total.

- Disease
- Gene
- Chemical
- SNPs

Before moving on, it is important to define SNP entity.
SNP stands for Single Nucleotide Polymorphisms. This entity is correlated with the most common type of genetic variation among people. Each SNP represents a difference in a single DNA building block. For example, a SNP may replace the nucleotide adenine (A) with the nucleotide guanine (G) in a certain stretch of DNA.They appear normally throughout a person's DNA, almost once in every 1,000 nucleotides on average. These variations appear in many individuals. In order for an entity to be classified as a SNP, a variant must be found in at least 1 percent of the population. Scientists have found more than 600 million SNPs in populations around the world [21]. Most commonly, they are found in the DNA between genes. SNPs can act as biological markers. This ability helps scientists to locate genes that are associated with diseases. Most SNPs do not contribute on the development or heath of humans. However, some of these genetic differences have proven to be essential in the study of human well-being. For example, SNPs help the scientists to predict an individual's response to numerous drugs or even the susceptibility of human organism to environmental factors, like toxins.

In general, in this thesis we are going to use machine learning for NLP classification task, implemented in Google Colab.
The thesis is divided into two main codes ·the acquisition of the data and the processing phase.

3.2 Acquisition of the dataset

As was mentioned in chapter *2.3 Biomedical natural language processing* and *2.4 BioC format,* a variety of biomedical data, are hidden inside specific formats, because it is easier to be transferred and analyzed this way. So, the first challenge was to parse the BioC files, in order to obtain the desired dataset.
The code that was used for the parsing of the files was originally developed by the supervisor of this thesis, Dr. Alexandros Kanterakis and his associates. However, the original code was modified, in order to give us relevant – with our research – elements. Moreover, some extra parts of code were added in order to bring the outcome to the desired form. These extra parts mainly concern the format of the created final finals, which are presented later on the code analysis.
The code ran in Colab environment, since the resources required for these scripts are too big for a regular personal computer.
7000 BioC files were parsed in total. The parsing of the files stopped, when sufficient sample of sentences had been collected. Approximately 100 papers were included inside each file. SNPs entities were basically the reason for parsing such amount of files, since sentences containing only the SNP entity are not so common in bibliography.

A thorough analysis of the code is presented below.
The first step is to install all the needed libraries in Colab environment. These libraries stay active until the end of each session. The BioC library is essential for the processing of the files.

```
! pip install bioc
import re
import os
import glob
import lxml.etree
import json
import bioc
from tqdm import tqdm
from collections import Counter, defaultdict
import pandas as pd
import csv
```

After that, the program should be loaded with the files that need to be parsed. All files should be located at the same folder and their name should have a specific form (like "2070bioc.xml"), in order to be easily read.

```
bioc_directory = '/content/drive/MyDrive/Bioc_pentakosia'
```

500 BioC files exist inside this directory. Generally, the 7000 BioC files were separated, in order for the execution time and the demand for extra RAM to be limited. It took almost 2hours and 15minutes for Colab to parse these 500 files, using the specification presented on *Figure 33*.

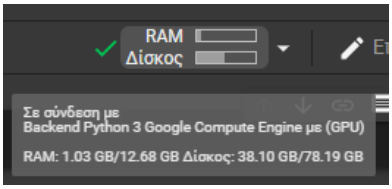


Figure 33 - Specifications

The next step is to create functions that help us to parse the files.
The following function will return all the file names existing inside the folder. Then the second function will use all the names in order for the program to read the whole text.

```
def get_filenames():
    filenames_wild = os.path.join(bioc_directory, '*.xml')
    filenames = glob.glob(filenames_wild)
    print ('Found {} BioC XML files'.format(len(filenames)))
    #print(filenames)
    return filenames
def parse_bioc(filename):
    with open(filename, encoding="utf8") as f:
        text = f.read()
        try:
            b = bioc.loads(text)
        except lxml.etree.XMLSyntaxError as e:
            errors['could_not_parse'].append([filename])
    return b
```

The next sequence of functions contains one of the most important parts of the code, since here the whole text is differentiated into the desired entities. It is possible, some entities to appear more than once inside the text. The use of set method helps us to overcome this obstacle, since each entity was held in the memory only once.

```
def parse_collection(collection):
    delme = defaultdict(int)
    wholetext = set()
    specifictext_D=set()
    specifictext_S=set()
    specifictext_G=set()
    specifictext_C=set()
    c = 0
```

As have already been mentioned a BioC file consists of collections. So, the use of for loop helps the program to go deeper into the tree, in order to find the wanted entities.

```
     #for document in collection['documents']:
    for document in collection.documents:
        genes = set()
        for passage in document.passages:
            text = passage.text
            wholetext.add(text)
```

The if-statement helps us to differentiate all the sentences of the text with the same type of entity and store them into one variable. The variable wholetext includes all the sentences of the document regardless of their type, while the variables specifictext_D, specifictext_S, specifictext_G, specifictext_C include only the words of "Disease" type, "SNP" type, "Gene" type and "Chemical" type, accordingly.

```
            for annotation in passage.annotations:
                delme[annotation.infons['type']] += 1
                c += 1
                if c > 10000:
                    pass
                if annotation.infons['type'] == 'Disease':
                    specifictext_D.add(annotation.text)
                elif annotation.infons['type'] == 'SNP':
                    specifictext_S.add(annotation.text)
                elif annotation.infons['type'] == 'Gene':
                    specifictext_G.add(annotation.text)
                elif annotation.infons['type'] == 'Chemical':
                    specifictext_C.add(annotation.text)
    return wholetext, specifictext_D, specifictext_S, specifictext_G, specifictext_C
```

Each file encloses data for every examined type of entity. The combination and the exportation of them, so as to be available for use later in the code, can be achieved by the following function. Using the same logic as before, the wtext variable includes all the sentences of the documents of every

collection, whereas the remaining variables contain all the words where the type of the entity is the common characteristic.

```
def import_graph():
    filenames = get_filenames()
    wtext=set()
    stext_D=set()
    stext_S=set()
    stext_G=set()
    stext_C=set()
    for filename in filenames:
        collection = parse_bioc(filename)
        wtext.update(parse_collection(collection)[0])
        stext_D.update(parse_collection(collection)[1])
        stext_S.update(parse_collection(collection)[2])
        stext_G.update(parse_collection(collection)[3])
        stext_C.update(parse_collection(collection)[4])
    return wtext, stext_D, stext_S, stext_G, stext_C
```

Moving on, the variable wtext is converted into a list. List type variables have many conveniences. One of them is that can be easily transformed into string type variables.

```
list_wt_v = list(filter(None, list(wtext)))
for i in range(0, len(list_wt_v)):
    if list_wt_v[i][-1] != '.' :
        list_wt_v[i] = ' ' + list_wt_v[i] + '. '

def datatostring(list_wt_v) :
    my_str_wt = ''
    for x in list_wt_v :
        my_str_wt += '' + x

    return my_str_wt
```

Using regular expressions, all the data are divided into separate sentences. Then, the set method is used again, since it is essential to keep each sentence only once.

```
div_wt = re.split(r'(?<=[^A-Z].[.?]) +(?=[A-Z])', datatostring(list_wt_v))
div_wt = set(div_wt)
print("Whole text length:", len(div_wt))
print('')

#creating a set of sentences with only genes and the set of all sentences
disease=set()
snp=set()
gene=set()
chemical=set()
entities=set()
set_wt=set()
```

At this point, the code is trying to differentiate each sentence based on the type of entities that are included. As mentioned before, in this thesis four biomedical entities· Disease, SNP, Gene and Chemical· are going to be examined. Each sentence contains only one of the aforementioned entities. For example the following sentence contains only the gene entity and no other entity: *The role of **Pax2** in mouse inner ear development*.
So basically, the sentences are divided into four categories:

- Sentences with only the **disease** entity
- Sentences with only the **SNP** entity
- Sentences with only the **gene** entity
- Sentences with only the **chemical** entity

For simplicity reasons, let's assume that a variable E has the following characteristics:

E = {Disease, Snp, Gene, Chemical}

Also, as S(E) are defined the sentences that have a specific entity, for example:

*The role of **Pax2** in mouse inner ear development* ⟶ *S(Gene)*

```
#S(E) = Has ONLY E  (not any other entity)
for k in div_wt:
    a = set(k.split())
    for y in a:
        if y in stext_D or y in stext_S or y in stext_G or y in stext_C:
            entities.add(k)
            break
    else:
        ######## no entity at all S3(E) #########
        set_wt.add(k)


for k in entities:
    a = set(k.split())
    for y in a:
        if y in stext_D:
            disease.add(k)
        elif y in stext_S:
            snp.add(k)
        elif y in stext_G:
            gene.add(k)
        elif y in stext_C:
            chemical.add(k)

disease_only=disease-snp-gene-chemical
snp_only=snp-disease-gene-chemical
gene_only=gene-disease-snp-chemical
chemical_only=chemical-disease-snp-gene
```

In this part of the code, the generation of final files, which will be later used on the DistilBERT model, takes place.

Each time, one pair of entities is going to be examined, meaning that the following cases are created:

1. S(Disease) vs S(SNP)
2. S(Disease) vs S(Gene)
3. S(Disease) vs S(Chemical)
4. S(SNP) vs S(Gene)
5. S(SNP) vs S(Chemical)
6. S(Gene) vs S(Chemical)

Comparing the pairs of sentences, we are going to examine if an entity can be distinguished easier than the other, based on their performance scores.
Moving on, let's randomly take as an example the 1st case and try to analyze the generation of the final files. The sentences should be characterized with a specific label. The label of 0 was given to all the sentences that had into them the **disease** entity, whereas the sentences that include the **SNP** entity took the label 0.

```
######### S (Disease) vs S(SNP) #########
final_d_only=list(disease_only) #label-->0
final_s_only=list(snp_only) #label-->1
```

The variable of each label should have the same length as the variable with which is going to match.

```
#creating labels
lab_do=[]
for lab1 in range(0, len(final_d_only)):
    lab_do.append(0)

lab_so=[]
for lab2 in range(0, len(final_s_only)):
    lab_so.append(1)
```

As we have already mentioned, this project run on Google Colab. Due to memory limitations, the length of each sentence had to be reduced. For this reason, it was necessary the length of each element to be collectively given in a new column.

```
#append the length of each element to list
new_do=[]
for t1 in range(0,len(final_d_only)) :
    d1=len(final_d_only[t1])
    new_do.append(d1)

new_so=[]
for t2 in range(0,len(final_s_only)) :
    s2=len(final_s_only[t2])
    new_so.append(s2)
```

In this point the final file is created. The outcome contains sentences with their respective labels. By counting characters, the minimum length of each sentence was set to be 21, while the maximum 159. File's extension is TSV (stands for Tab Separated Values), since it's the only acceptable type for DistilBERT code, who follows.

```
#creating the final file
df_do_1 = pd.DataFrame({
    'Sentences': final_d_only,
    'Labels': lab_do,
    'Length': new_do } )

df_do_1 = df_do_1[df_do_1['Length'] > 15]
df_do_1 = df_do_1[df_do_1['Length'] < 300]
df_do_1 = df_do_1.reset_index(drop=True)
#df_do_1 = df_do_1.iloc[:5]

df_so_2 = pd.DataFrame({
    'Sentences': final_s_only,
    'Labels': lab_so,
    'Length': new_so } )

df_so_2 = df_so_2[df_so_2['Length'] > 15]
df_so_2 = df_so_2[df_so_2['Length'] < 300]
#df_so_2 = df_so_2.iloc[:5]

df_dso_final_a=df_do_1.append(df_so_2)
df_dso_final_a=df_dso_final_a[['Sentences','Labels']]
df_dso_final_a.to_csv('test_label_do_so.csv',index=False, header=False)

csv.writer(open("S4_Ed_vs_Es.tsv", 'w+'), delimiter='\t').writerows(csv.reader(open("test_labe
l_do_so.csv")))
```

Using the above example, we end up to the generation of final files that can be used for evaluation of the 1st case. The final files for all the presented cases can be generated using the same logic patterns.

So, the dataset that will be used for processing phase has the following set-up.

```
B, H-chains recovered from lipogranulomas obtained from TMPD-treated BALB/c mice.   0
However, little is known about how prenatal perturbation translates into adult brain dysfunction.   0
Factors influencing flexor tendon adhesions.    0
Neither intracranial hemorrhage nor proteinuria occurred.   0
Surgical regeneration therapy using myoblast sheets for severe heart failure.   0
It took many years of additional work to establish that ACh was also a neurotransmitter of the central nervous system.  1
Another approach incorporates pH-sensitive monomers to modulate the lower critical solution temperature (LCST) of thermosensitive polymers. 1
The enzyme activity was significantly stimulated by Fe2+.   1
The role of Pax2 in mouse inner ear development.    1
```

Figure 34 - Dataset layout

As can be seen on the *Figure 34*, the first half of the dataset consists of sentences of the one label, while the other one of sentences of the other label. Of course, the total number of the sentences of each category differs, however this incompatibility will be dealt later on the code.

3.3 Processing phase

For the sake of completeness, the examined cases are presented again.

1. S(Disease) vs S(SNP)
2. S(Disease) vs S(Gene)
3. S(Disease) vs S(Chemical)

4. S(SNP) vs S(Gene)
5. S(SNP) vs S(Chemical)
6. S(Gene) vs S(Chemical)

For each case the maximum number of sentences was variable. The number of sentences for each repetition is listed below.

- 100
- 500
- 1000
- 1500
- 2000
- 2500
- 3000

10 repetitions were completed in total, for each one of the above classes.

Colab was also used for the processing phase. The upper limit of the number of sentences was defined based on Colab's limitations. Colab's free of charge version collapsed after a while above 3000 sentences.
During the phase of experiments Colab randomly gave us the specifications seen on *Figure 35*.

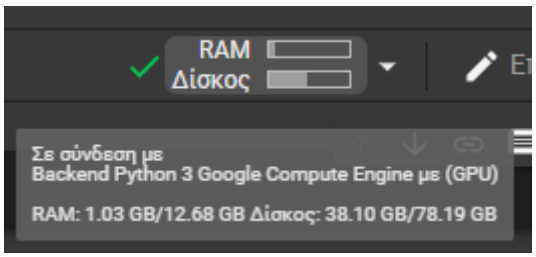


Figure 35 - Specifications

The execution time will be presented later on.

This phase actually consists of two main models: DistilBERT and Logic Regression model. The final result of the processing of DistilBERT, will be taken in by the second code. Using logic regression the sentences will be classified as either positive or negative (1 or 0, respectively). A key point of the code is that DistillBERT is already pre-trained and only the logistic regression model will need training.
The vector size of the data passed through the models is equal to 768.

3.3.1 Explaining of the codes

The first section of the code includes all the needed tools and libraries.

```
!pip install transformers
!pip install sklearn-utils
import numpy as np
import pandas as pd
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import torch
import transformers as ppb
import sklearn.linear_model as linear_model
import sklearn.metrics as metrics
import numpy
from sklearn.metrics import confusion_matrix
import warnings
warnings.filterwarnings('ignore')
```

As have already been mentioned, the proper form for the input file is TSV extension. The file is read by the code, using pandas.

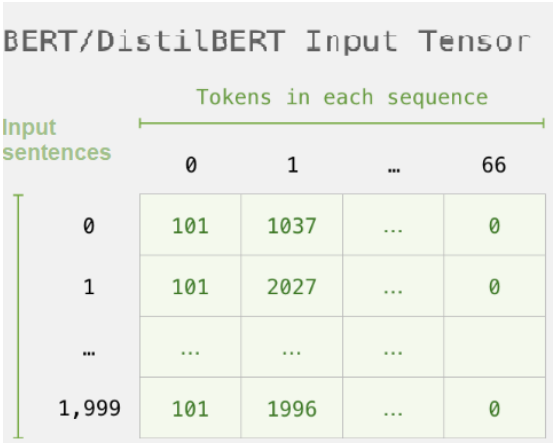
```
df = pd.read_csv('/content/drive/MyDrive/S4_Eg_vs_Es.tsv', delimiter='\t', header=None)
```

The input file contains sentences from both categories (0 or 1). First, the sentences of the first category are listed, followed by the sentences of the second category (see *Figure 34*). Using shuffle function, the sentences are mixed (see *Figure 36*), so as to have approximately the same number of zeroes and ones, during the running of each repetition.

```
Health care professionals viewed accreditation...  0.0
However, no significant difference in the effe...  1.0
The rs4680 genotyping was performed using ABI ...  1.0
The highest score attainable for EDs according...  0.0
Individual body weights were measured and reco...  0.0
                                                   ...  ...
As a result, including wrongdoing in scenarios...  0.0
G/T 0.08 0.10 0.77 (0.39-1.50) 0.443     rs769...  1.0
```

Figure 36 – Shuffled sample

The next step is to tokenize the dataset, with its basic idea to be presented on *Figure 37*. All the sentences will be tokenized and processed together as a batch. This can only be achieved if all the vectors have the same length. The maximum length is defined from the sentence with the maximum length. The vectors, corresponding to sentences with smaller length, are filled in with the id=0.



Source: *The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.* (jalammar.github.io)

Figure 37 - Tokenization of the sentences

```
# For DistilBERT:
model_class, tokenizer_class, pretrained_weights = (ppb.DistilBertModel, ppb.DistilBertTokeniz
er, 'distilbert-base-uncased')

# Load pretrained model/tokenizer
tokenizer = tokenizer_class.from_pretrained(pretrained_weights)
model = model_class.from_pretrained(pretrained_weights)
tokenized = batch_4[0].apply((lambda x: tokenizer.encode(x, add_special_tokens=True)))
max_len = 0
for i in tokenized.values:
    if len(i) > max_len:
        max_len = len(i)

padded = np.array([i + [0]*(max_len-len(i)) for i in tokenized.values])
attention_mask = np.where(padded != 0, 1, 0)
```

The next step is the processing of the dataset using DistilBERT. For this reason, an input tensor is created from the token matrix.

```
input_ids = torch.tensor(padded)
attention_mask = torch.tensor(attention_mask)

with torch.no_grad():
    last_hidden_states = model(input_ids, attention_mask=attention_mask)
```

The output of DistilBERT code is hidden in the line *last_hidden_states*. Basically, it is a tuple. Its shape is defined by three variables ´ the maximum number of examples (in our case, it is variable between 100 and 3000), the maximum number of tokens in the sequence (it changes in every run) and the number of hidden units in the DistilBERT model (768).

This 3D output tensor should be sliced in order to end up to a 2d tensor that can be used by the model. By keeping only the [CLS] token and discarding everything else, the model has the whole information that needs to move on to the next stage.

```
features = last_hidden_states[0][:,0,:].numpy()
```

So, the 3D tensor is sliced in a 2D numpy array. This array contains the embeddings of all the sentences of the dataset.

The training of the Logistic Regression model follows.

The first step is to split the data into train and test datasets.

```
train_features, test_features, train_labels, test_labels = train_test_split(features, labels,
test_size=0.25)
```

The 768 columns of the array correspond to "features" variable, while the labels come from the initial dataset. The last argument defines the size of the test sample, which can vary. In our case, the test set was defined to be 25% of the whole dataset.

```
lr_clf = LogisticRegression()
lr_clf.fit(train_features, train_labels)
```

Now, it's the time to test the performance of the code.

```
print("classifier score:",lr_clf.score(test_features,test_labels))
y_pred = lr_clf.predict(test_features)

cf = confusion_matrix(test_labels, y_pred)
print(cf)

accuracy = (cf[0,0] + cf[1,1]) / (cf[0,0] + cf[1,0] + cf[0,1] + cf[1,1])
print('accuracy = ', accuracy)

precision = (cf[0,0]) / (cf[0,0] + cf[0,1])
print('precision = ', precision)

recall = cf[0,0] / (cf[0,0] + cf[1,0])
print('recall = ', recall)
```

All the results were evaluated using confusion matrix.

4. Presenting and analyzing the findings

In this chapter, the results of the research are going to be presented and discussed. The final percentage for each case occurred using the average function between the 10 repetitions.

4.1 SNP vs Disease, Gene and Chemical

Using only the common logic, it is assumed that sentences with SNPS can be distinguished easily, when comparing with sentences with other entities, since they have very specific format.

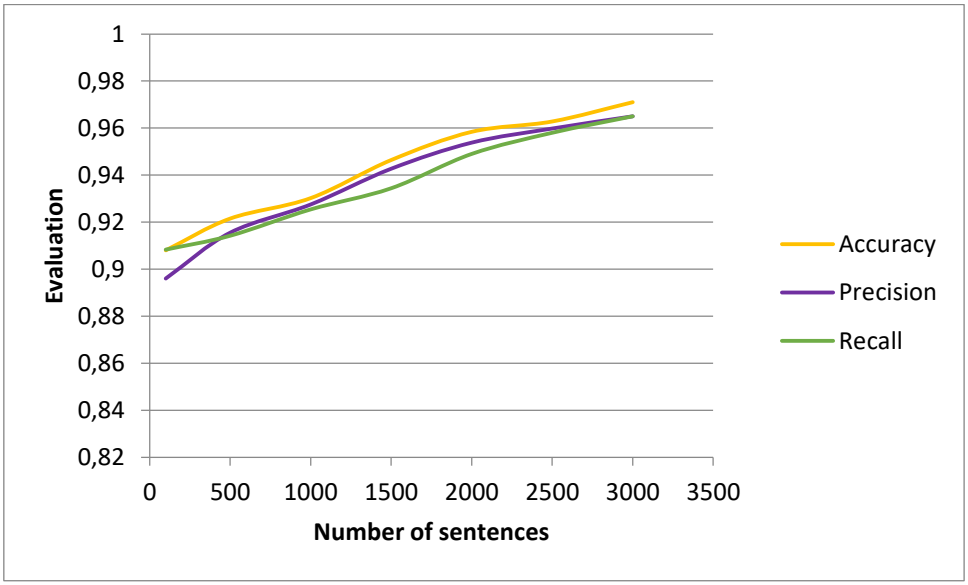


Figure 38 - SNP vs Disease Evaluation diagram

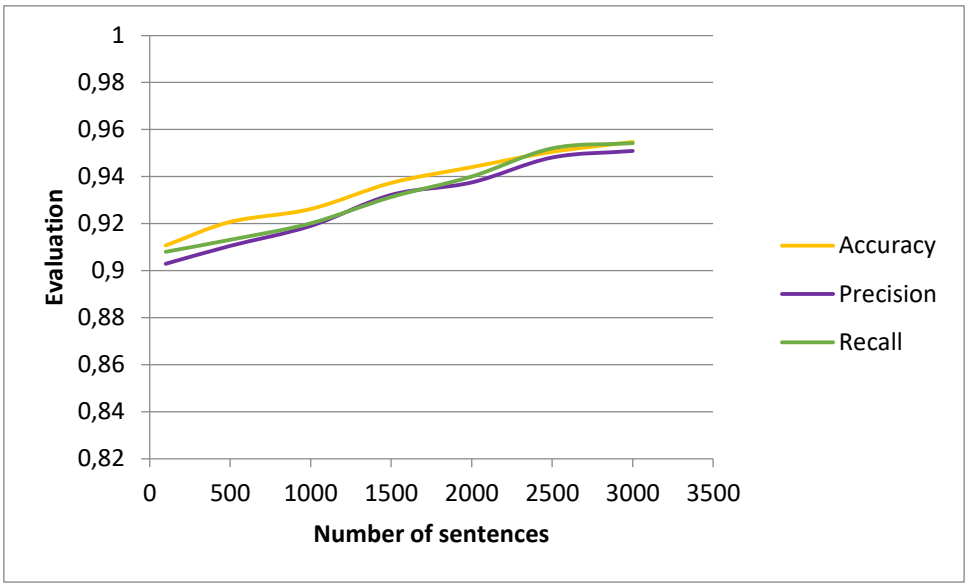


Figure 39 - SNP vs Gene Evaluation diagram

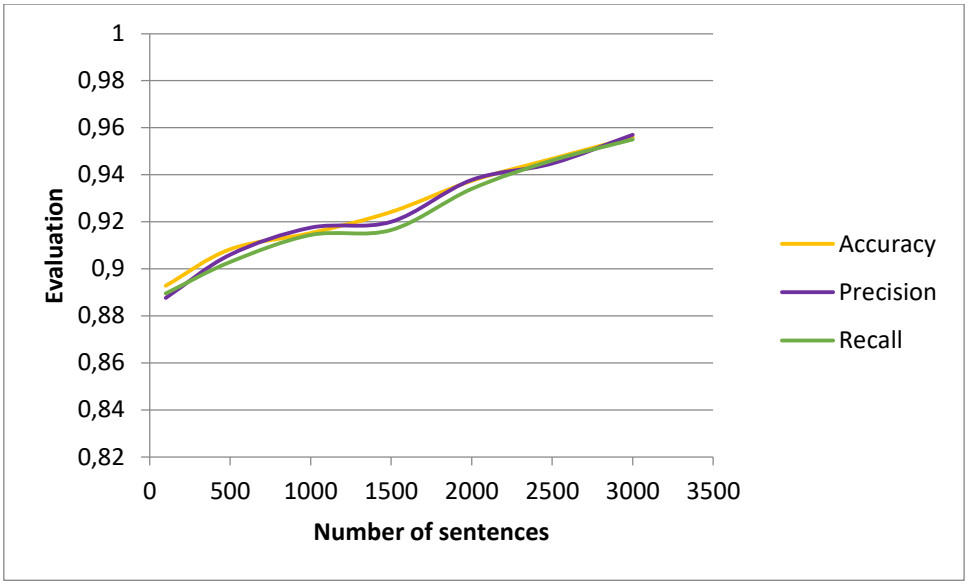


Figure 40 - SNP vs Chemical Evaluation diagram

Figure 38, Figure 39 and *Figure 40* depict the total evaluation when comparing sentences with SNP with sentences with Disease, Gene and Chemical respectively. As someone can observe from the above diagrams, this entity is recognized without difficulty inside the sentences, regardless of the entity that is compared with. Therefore our deep learning network is more likely to distinguish the sentences with quite an exceptional accuracy, precision and recall percentage. More specifically, when comparing SNP with Disease entities, we understand that the model can easily recognize the difference between the two entities. That is also a logic conclusion, since these two entities are completely different in their forms. The same thing does not happen to such a great extent when comparing SNPs with Genes or Chemicals. But still, the model performs outstanding, as can be seen on *Table 3*.

	Accuracy	Precision	Recall
Snp Vs Disease	0,971	0,965	0,965
Snp Vs Gene	0,9547	0,9509	0,9542
Snp Vs Chemical	0,956	0,957	0,955

Table 3 - Accuracy/Precision/Recall at the maximum number of sentences for the three cases

The following diagrams presents the discrepancy created between repetitions when the model runs a sample of 100 sentences with one with 3000 sentences.

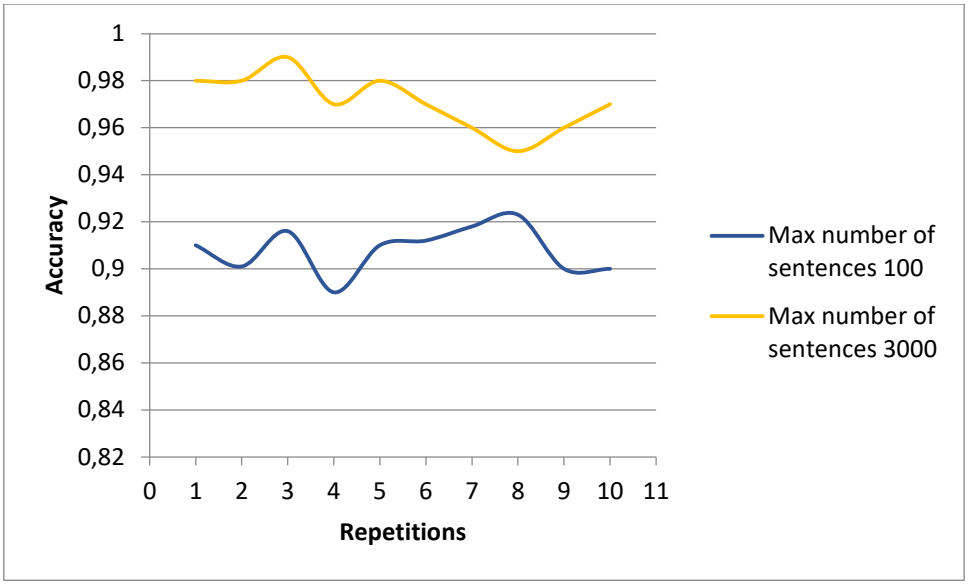


Figure 41 - SNP vs Disease Accuracy diagram

Concerning accuracy score, the observed deviation is approximately 3.3% for the first case whereas is 4% for the second case (see *Figure 41)*.

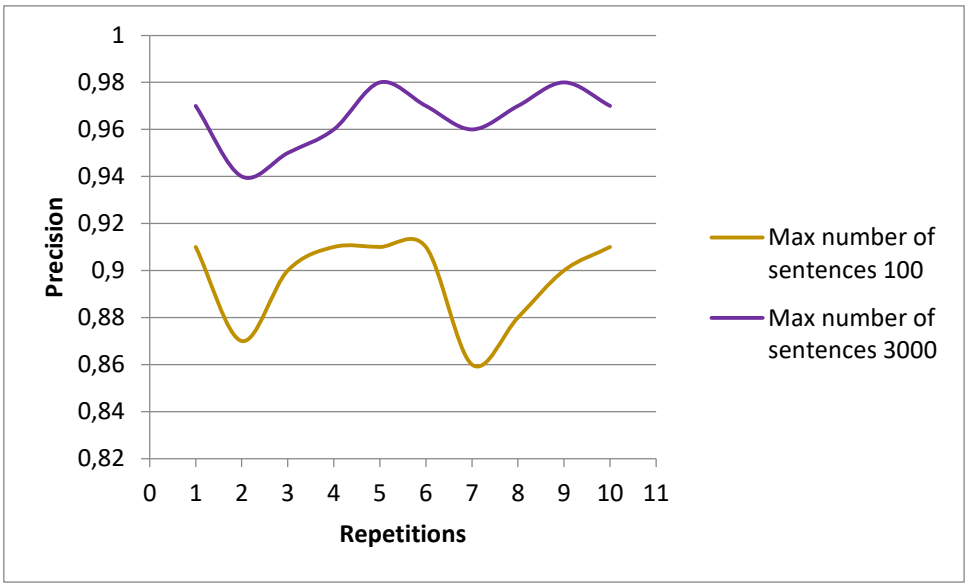


Figure 42 - SNP vs Disease Precision diagram

As far as the precision score is concerned, the depicted aberration is 5% and 4% respectively (see *Figure 42*).

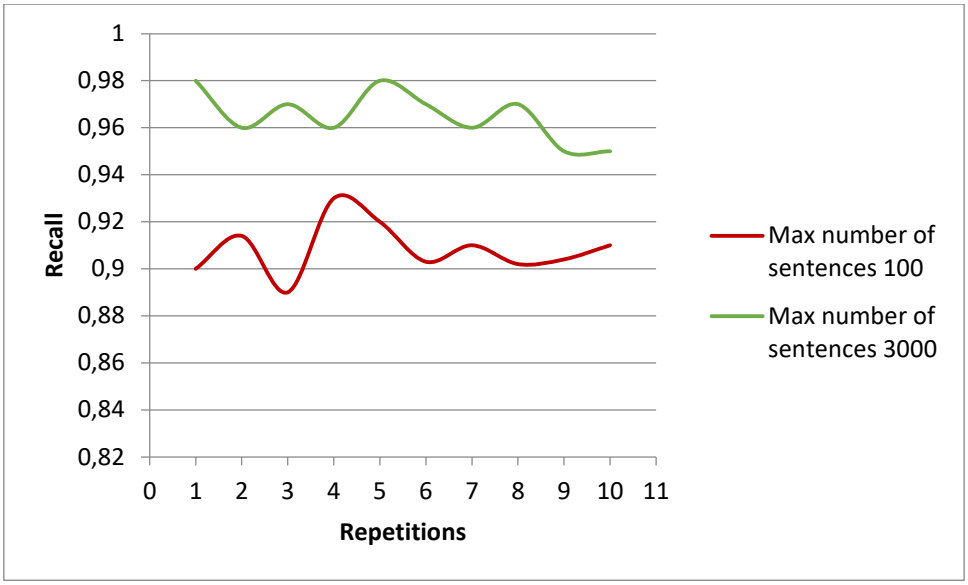


Figure 43 - SNP vs Disease Recall diagram

The same behavior displays the recall score, where the measurable deviation is 4% and 3% for the two examined cases, as can be seen on *Figure 43*.

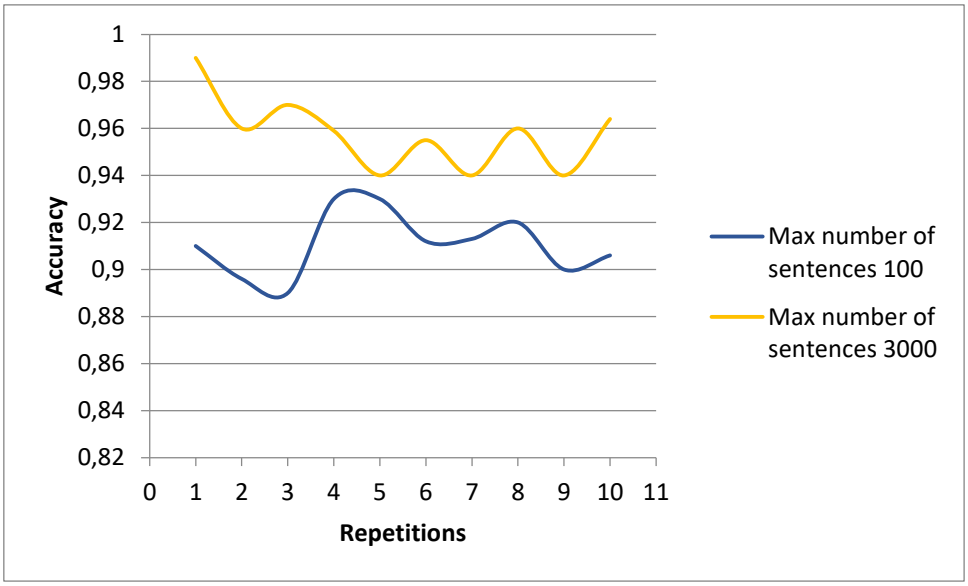


Figure 44 - SNP vs Gene Accuracy diagram

The accuracy diagram for SNP vs Gene is presented on *Figure 44*. The maximum fluctuation for the accuracy score is 4% and 5% for each case.

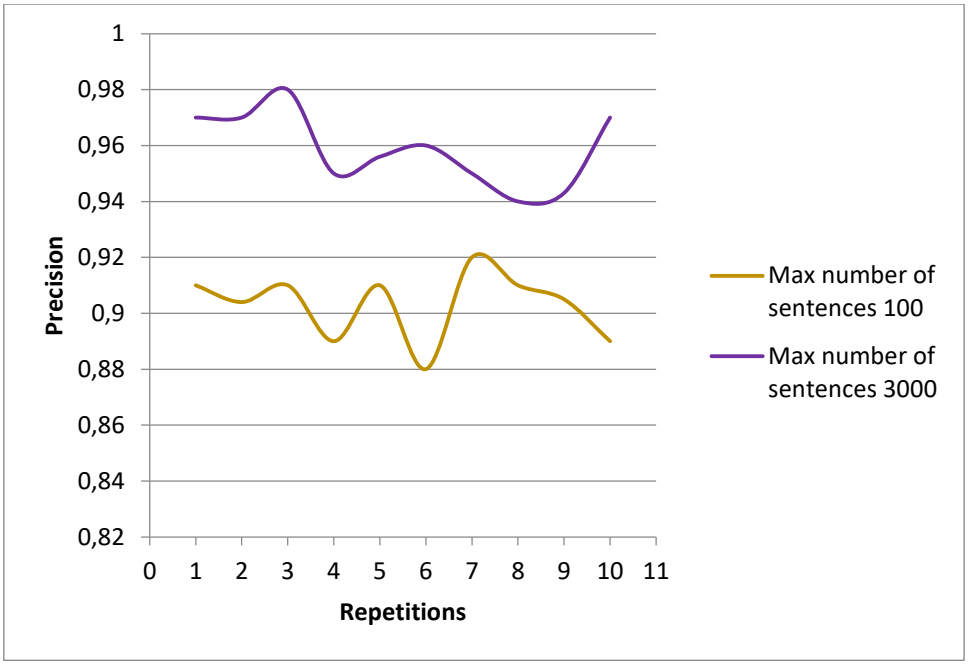


Figure 45 - SNP vs Gene Precision diagram

Concerning precision score, the observed deviation is approximately 4% for both cases (see *Figure 45*).

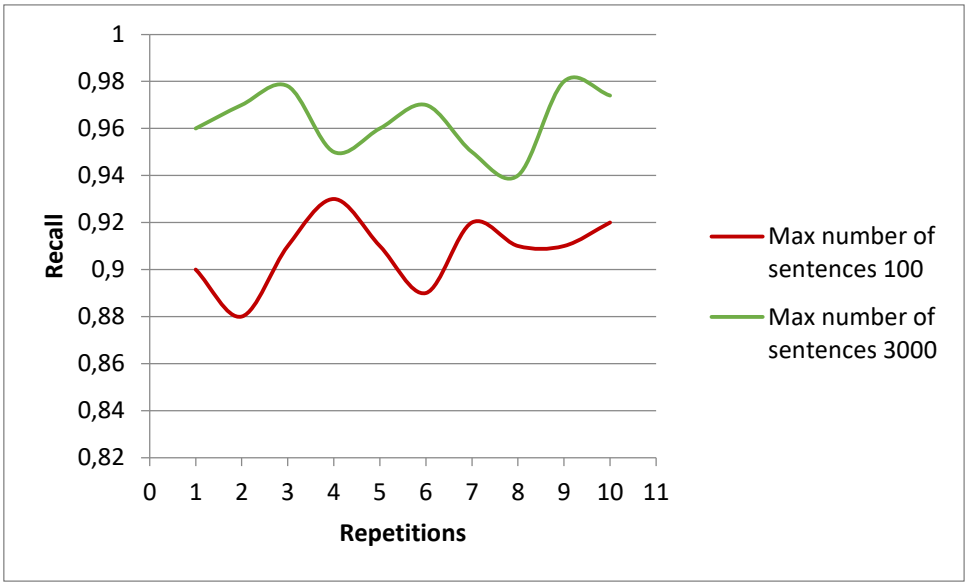


Figure 46 - SNP vs Gene Recall diagram

As far as the recall score is concerned, the depicted aberration is 5% and 4% respectively (see *Figure 46*).

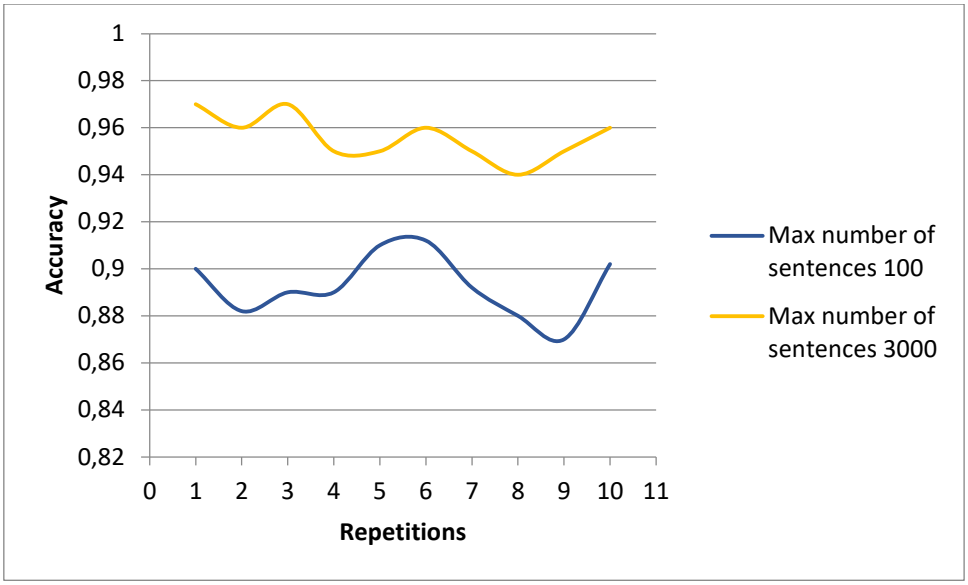


Figure 47 - SNP vs Chemical Accuracy diagram

Concerning accuracy score, depicted on *Figure 47*, the observed variation is approximately 4.2% for the first case whereas is 3% for the second case.

Figure 48 - SNP vs Chemical Precision diagram

Similar behavior displays the precision score, where the measurable fluctuation is 5% and 3% for the two presented cases (see *Figure 48*).

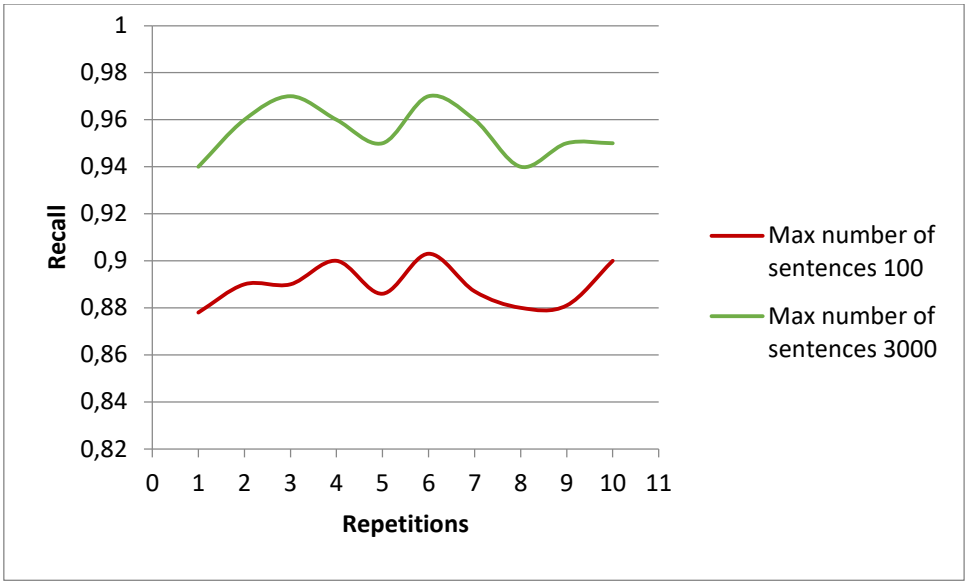


Figure 49 - SNP vs Chemical Recall diagram

As far as the presented recall score is concerned (see *Figure 49*), the depicted aberration is 2.5% and 3% respectively.

Overall, the maximum deviation did not exceed 5% in any evaluation score. This percentage is considered relatively small, if one considers that we refer to statistical models. So, someone could assume that even if the model was trained with smaller dataset, it would be capable to distinguish these entities successfully.

4.2 Disease vs Gene

Moving on, we are going to examine how our model reacts when it has to distinguish sentences with only Disease or only Gene entities.

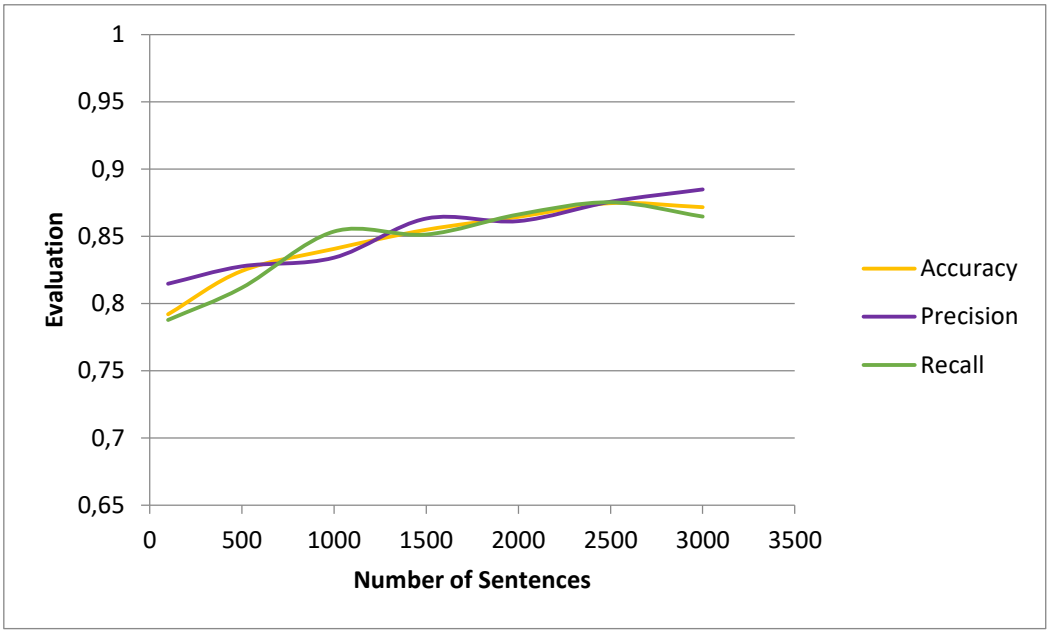


Figure 50 - Evaluation diagram

As we can see at *Figure 50* even at 3000 sentences (maximum number of sentences for this experiment) none of the performance scores was above 0.9. This means that our model faced some extra difficulty to recognize correctly these entities. Moreover, a recall value of 0.86 out of 1 means that the model overlooks some data in order to fit. However, the fact that the 3 scores are increased as the number of sentences increase shows that the model presents better behavior while trained with larger test/training sets.

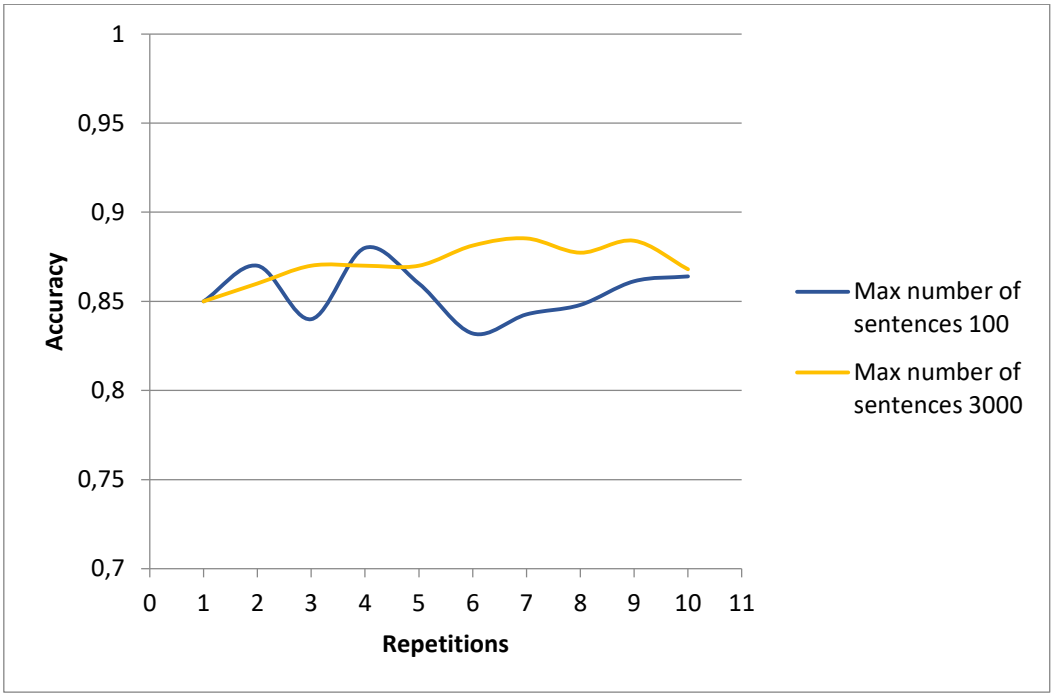


Figure 51 - Accuracy diagram

Concerning the accuracy score of *Figure 51*, the observed variation is 4.8% for the first case whereas is approximately 3.5% for the second case.

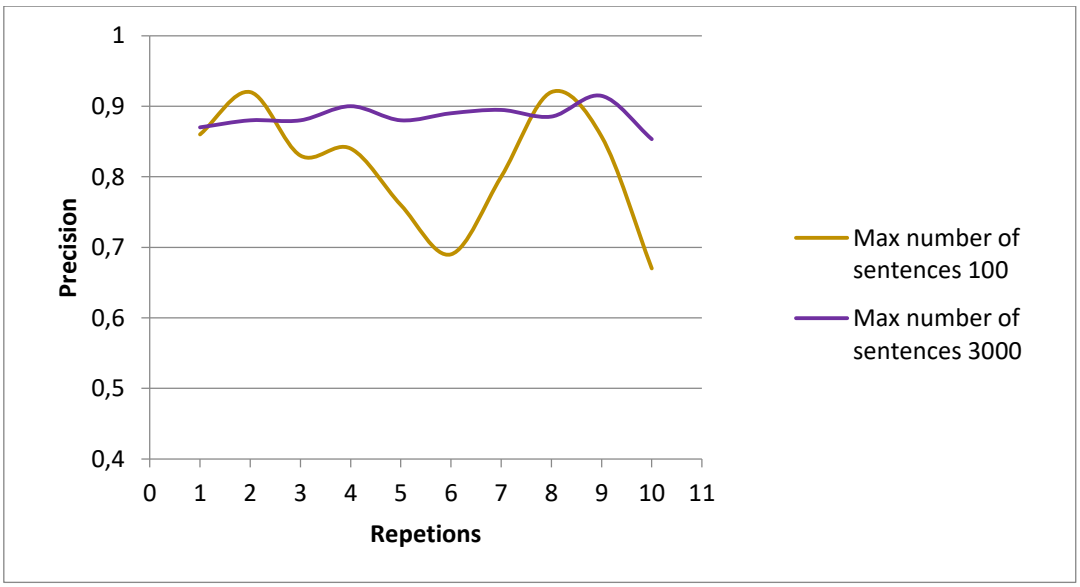


Figure 52 - Precision diagram

As far as the precision score is concerned, the depicted aberration is 25% and 6% respectively (see *Figure 52*). That is the biggest difference has been observed in this research until now. Such large-

scale variations between repetitions of the same case depict that there is great uncertainty inside the model.

Even worse behavior is displayed in the Recall diagram.

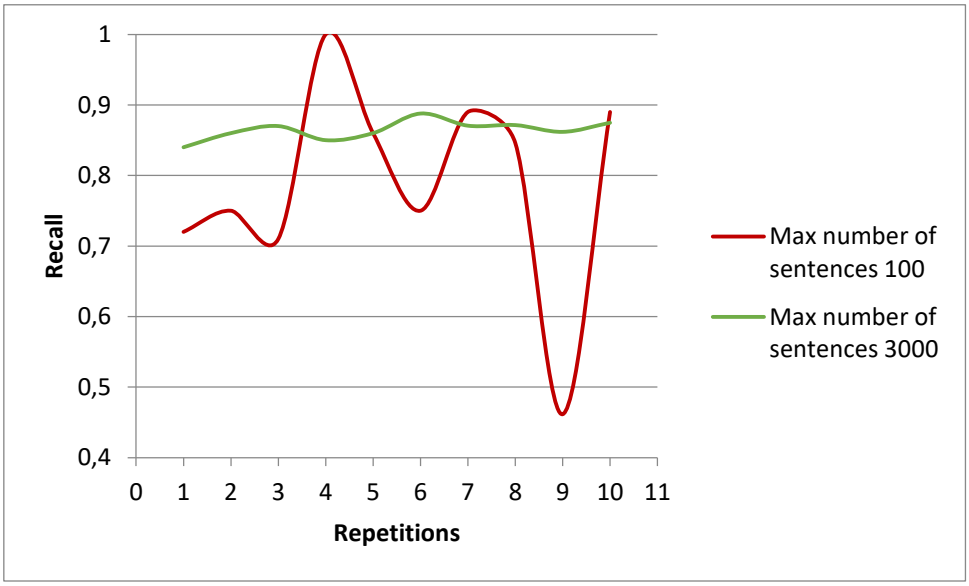


Figure 53 - Recall diagram

In *Figure 53*, the measurable fluctuation is almost 55% in the case where the maximum number of sentences is 100, whereas this behavior improves as the number of sentences increases, reaching approximately 3%.
Overall, a model trained with a small dataset is quite possible to present incorrect results.

4.3 Disease vs Chemical

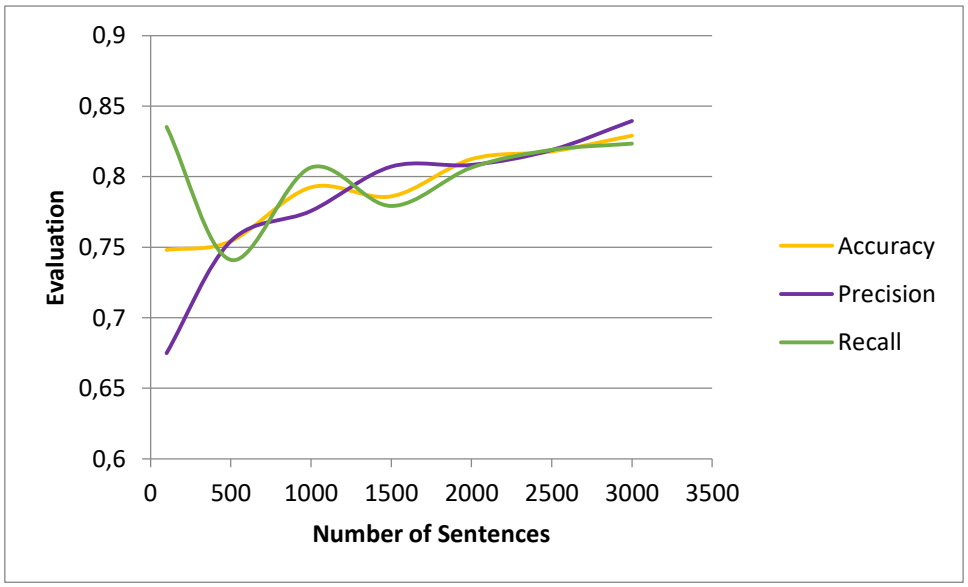


Figure 54 - Evaluation diagram

Comparing the Figure 50 with *Figure 54* someone can understand that, the Disease entity is easier to be recognized when it is correlated with gene entity rather than chemical entity. *Table 4* presents a spherical comparison between the performance scores of the two aforementioned cases.

Table 4 - Comparison between the 3 performance scores

	Accuracy	Precision	Recall
Disease vs Gene	0,87159	0,8848	0,86463
Disease vs Chemical	0,82902	0,83944	0,82337

As we can see from the *Table 4*, it is easier for the model to recognize correctly the two entities of the first case than the second one.

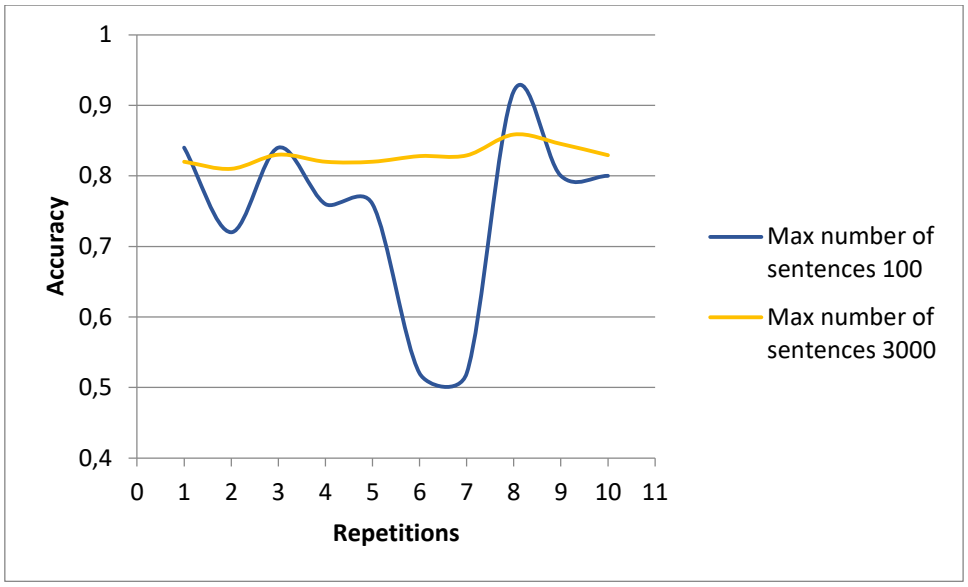


Figure 55 - Accuracy diagram

As far as the accuracy score is concerned, the presented aberration is 40% and 4.9% respectively (see *Figure 55*).

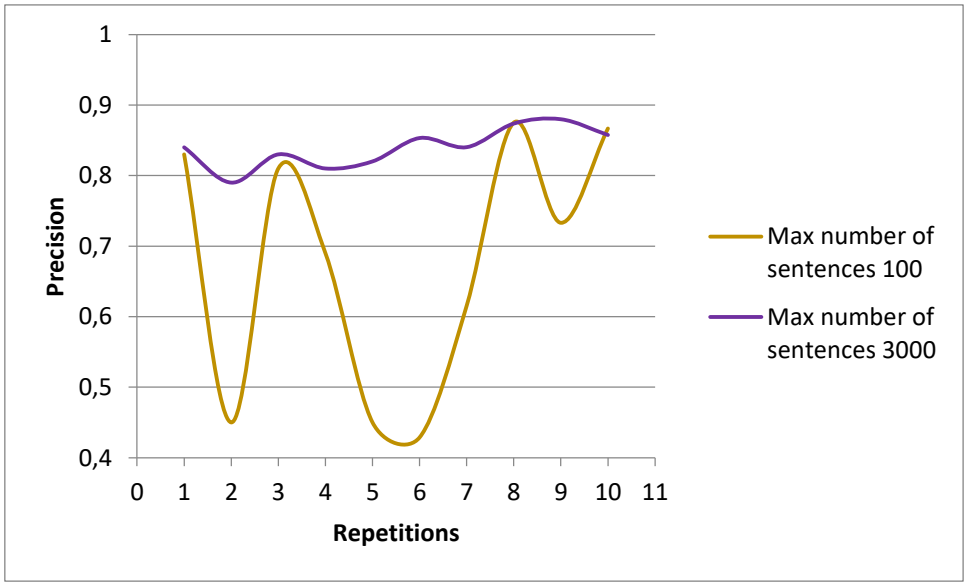


Figure 56 - Precision diagram

Similar behavior displays the precision score (see *Figure 56*), where the measurable fluctuation is 44.5% and 9% for the two presented cases. That's the first time inside this research, that there is so great variation between the repetitions even at 3000 sentences.

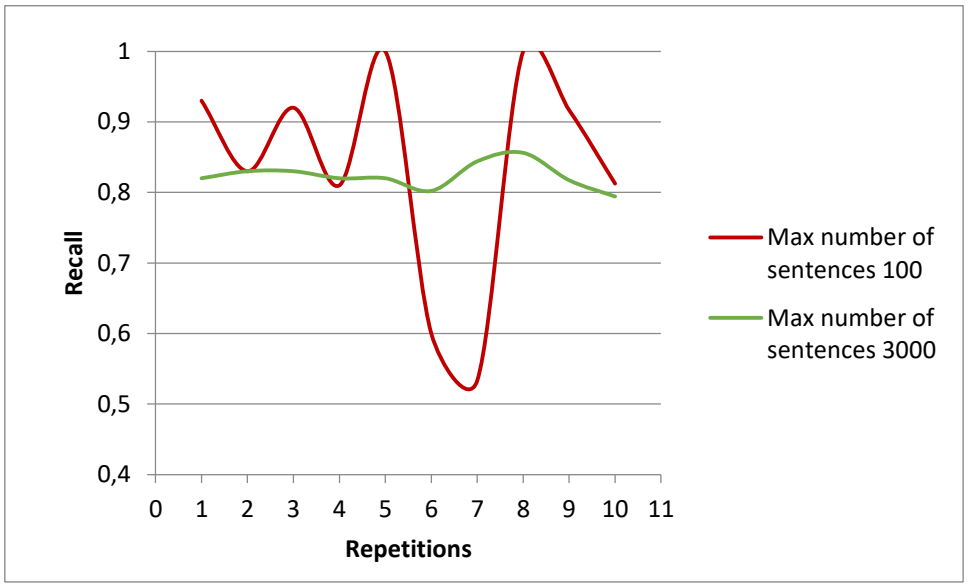


Figure 57 - Recall diagram

Similar behavior shows also the *Figure 57*, where the measurable aberration is 46% and 6.2%

In summary, the model cannot distinguish the two entities with great success. The uncertainty is obvious in every diagram. So, it is possible that if someone trains the model with a small dataset, it won't be able to provide accurate results.

4.4 Gene vs Chemical

The problem with genes and chemicals is that there is variety in naming, meaning that some genes or chemicals have names like a common English word or they contain stopwords, creating false positives in the recognition procedure. Taking this parameter under consideration, someone would expect that this behavior would also affect model's performance.

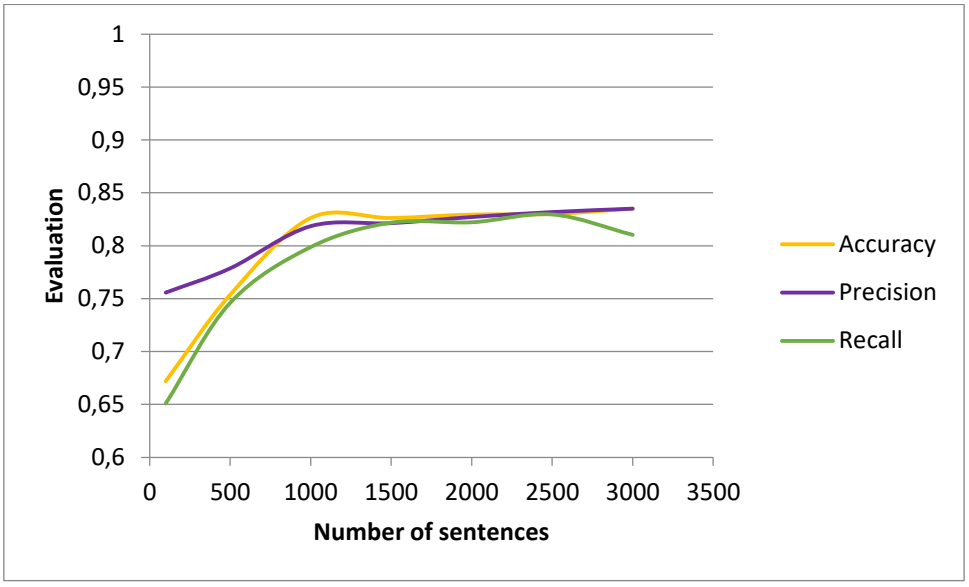


Figure 58 - Evaluation diagram

By observing the evaluation diagram, depicted on *Figure 58*, someone can conclude that these two entities have quite similar format and it is quite challenging for the model to distinguish them.

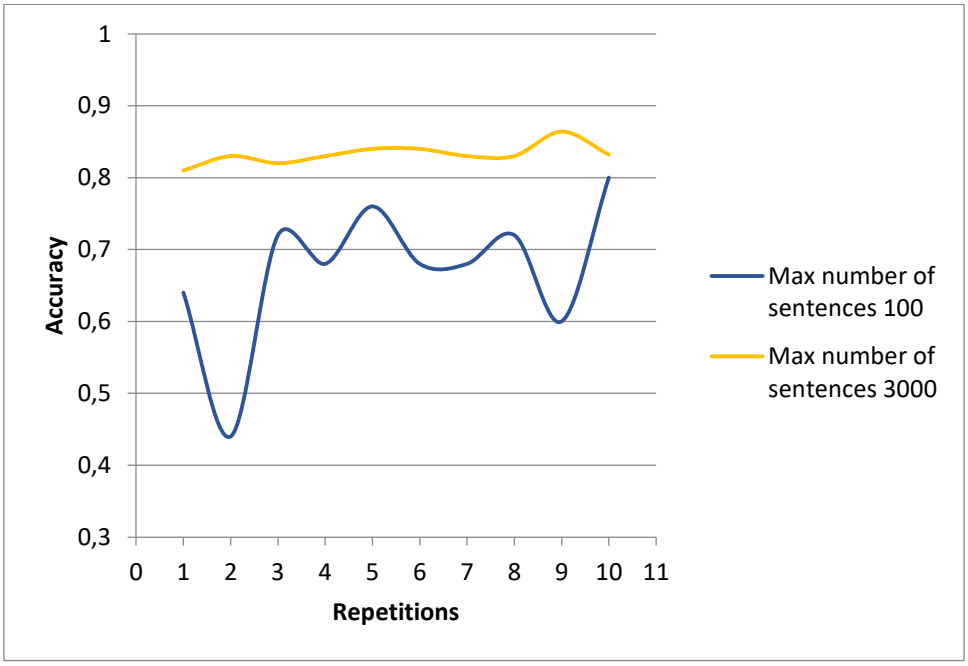


Figure 59 - Accuracy diagram

Concerning accuracy score, the observed variation is approximately 36% for the first case whereas is 5.4% for the second case, as can be seen on *Figure 59*.

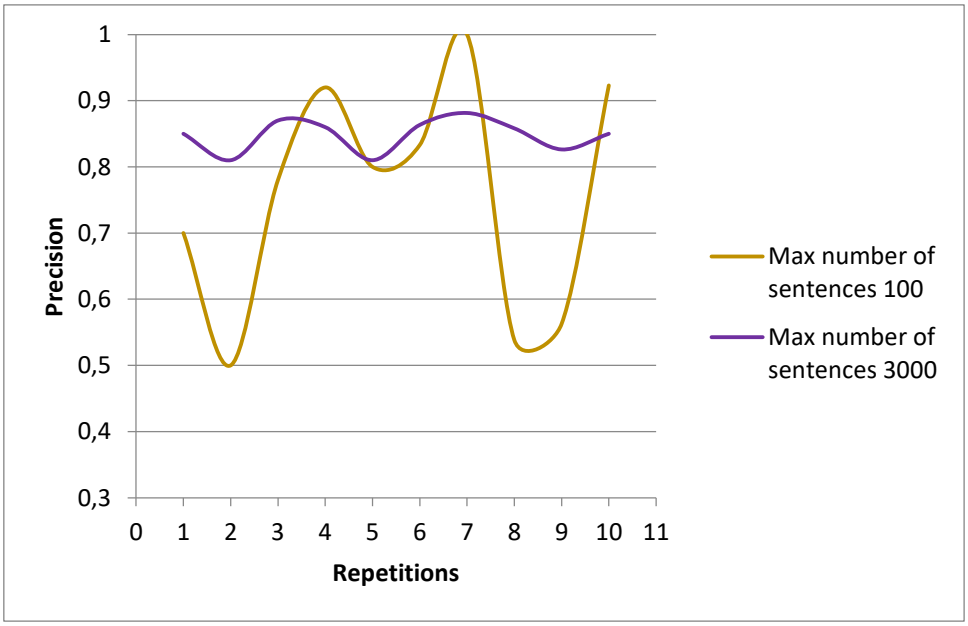


Figure 60 - Precision diagram

The precision diagram is presented on *Figure 60*, where the depicted aberration is 50% and 7% respectively.

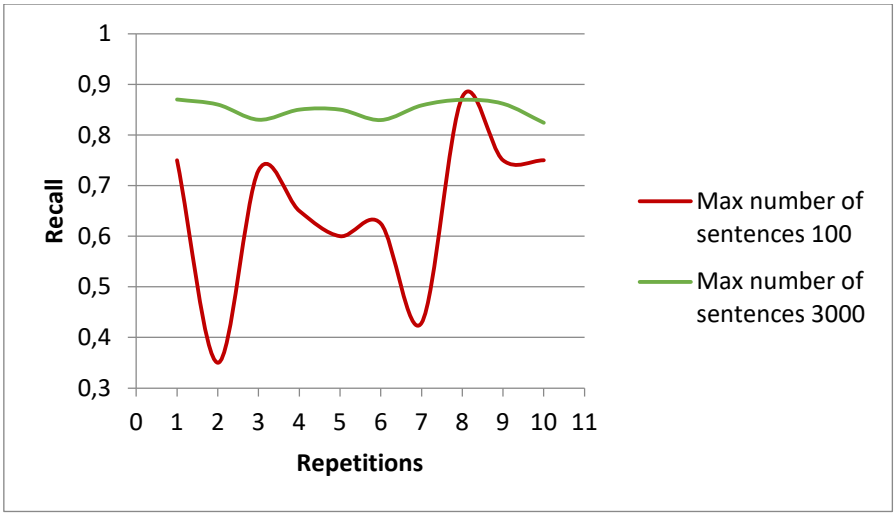


Figure 61 - Recall diagram

The same behavior is also directed on the recall diagram (see *Figure 61*), where the measurable aberration is 52.5% and 4.6%.

Given the above diagrams, a high fluctuation on the values of max number of sentences 100 is observed. So it is safe to assume that if someone wants more legitimate results, then he/she should definitely train the model with higher number of sentences.

4.5 Execution time

In chapter 2.8 Colaboratory (Colab) was pointed out that Colab's free version could not manage datasets with over 3000 sentences since after a small period of time it collapsed. Also, sentences with vectors approximately over 159 (it symbolizes the characters of each sentence) caused problems during execution period.

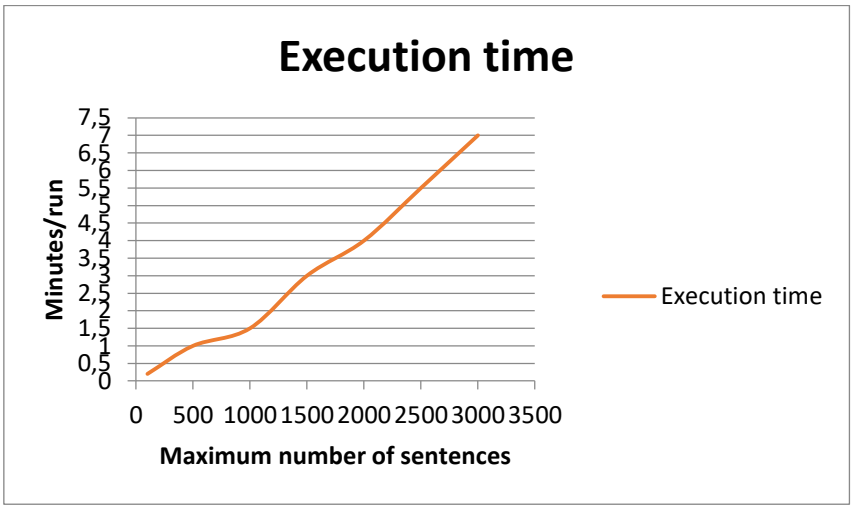


Figure 62 - Mean execution time

As can be spotted on *Figure 62*, the execution time increases as the maximum number of sentences increases. When executing the repetitions with the lowest number of sentences the average running time was 12secs, whereas with the maximum number of cases the execution time reached almost 8minites. Therefore, as the size of dataset increases, the needed computer power also increases.

5. Conclusions and recommendations

In this chapter, a brief summarization of the findings is presented.

Given the performance of Colab's free version during the research, the first conclusion that occurs, is that this version of Colab, when running DistilBERT code, is not able to manage datasets over 3000 sentences, with maximum number of characters of each sentence to be approximately 150. Moreover, the execution time increases as the number of sentences increase.
Keeping up, in the comparison between SNP entity with all the other entities turned out that the model is capable each time to distinguish the entities of the sentences with excellent performance, with accuracy, precision and recall to be over 95% at 3000 sentences. Also, the comparison between Disease and Gene entities showed that the model could recognize the sentences of each entity with quite satisfactory performance, with accuracy to be equal to 87%, precision 88% and recall 86%. Similar behavior was also observed in the comparison, between Disease and Chemical entities, where the model could recognize the sentences of each entity with good performance, with accuracy to be equal to 83%, precision 84% and recall 82%. It seems that comparing Genes and Chemicals is not an easy task for the model, since the model finds it difficult to distinguish the two entities due to their formats. The three performance scores had the lowest value ever found during this research, with accuracy to be equal to 83%, precision 82% and recall 83%.
The total performance score for all the examined cases at the maximum number of sentences is presented on *Figure 63.*

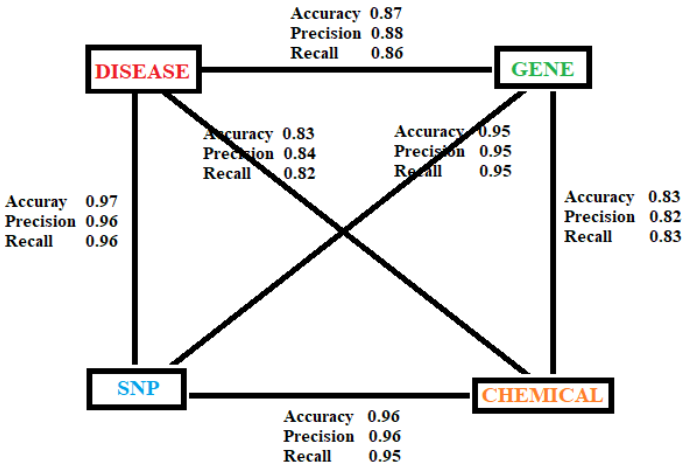


Figure 63 - Summary of total evaluation

By evaluating the results of the research, it is safe to assume that only if one of the two examined entities is SNP, then the model is capable to achieve a high performance score even if it is trained with only 100 sentences. Trying this between other entities carries the risk of uncertainty.

In general, no model built on statistics is free of errors. A percentage of 100% to all three scores does not only need great computer power, but also unlimited data. So, it's up to the researcher to define the "perfect" score of every examined case.

This research has provided us with worth-mentioning results. However there is always room from some extra research to be done. So, better fine tuning of the model is one parameter that could affect the total performance of the model. Moreover, small changes on the dataset could also influence the outcome. For example, datasets with smaller vectors (number of characters inside them) but over 3000 sentences or datasets below 3000 sentences but with bigger vectors in each sentence, could be used in order to examine if the increase in the three performance scores worth the extra time. Another recommendation would be an increase on the number of the repetitions for every case. Moreover, the

comparison between other entities, like Genomics or Proteomics could change the outcome. Last but not least, the use of other models in the place of DistilBert (like BioBert) could affect in great extent the behavior of each examined case.

References

[1]	Aikaterini-Lida Kalouli, Annebeth Buis, Livy Real, Martha Palmer, and Valeria de Paiva. 2019. Explaining Simple Natural Language Inference. In *Proceedings of the 13th Linguistic Annotation Workshop*, pages 132–143, Florence, Italy. Association for Computational Linguistics.
[2]	*Accuracy, Precision, Recall and the Glass of Water*. Medium. (2019). Retrieved 10 May 2022, from https://towardsdatascience.com/accuracy-precision-recall-5c8b8f0abde1.
[3]	Alammar, J. (2018). *The Illustrated Transformer*. Jalammar.github.io. Retrieved 10 May 2022, from http://jalammar.github.io/illustrated-transformer/.
[4]	*BERT Explained: State of the art language model for NLP*. Medium. (2018). Retrieved 10 May 2022, from https://towardsdatascience.com/BERT-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.
[5]	Cohen, K., & Demner-Fushman, D. (2014). *Biomedical natural language processing*. John Benjamins Publishing Co / John Benjamins North America.
[6]	Comeau, D., Batista-Navarro, R., Dai, H., Islamaj Do an, R., Jimeno Yepes, A., & Khare, R. et al. (2014). BioC interoperability track overview. *Database*, *2014*(0), bau053-bau053. https://doi.org/10.1093/database/bau053
[7]	Comeau, D., Islamaj Dogan, R., Ciccarese, P., Cohen, K., Krallinger, M., & Leitner, F. et al. (2013). BioC: a minimalist approach to interoperability for biomedical text processing. *Database*, *2013*(0), bat064-bat064. https://doi.org/10.1093/database/bat064
[8]	Donald C. Comeau, Riza Theresa Batista-Navarro, Hong-Jie Dai, Rezarta Islamaj Doğan, Antonio Jimeno Yepes, Ritu Khare, Zhiyong Lu, Hernani Marques, Carolyn J. Mattingly, Mariana Neves, Yifan Peng, Rafal Rak, Fabio Rinaldi, Richard Tzong-Han Tsai, Karin Verspoor, Thomas C. Wiegers, Cathy H. Wu, W. John Wilbur, BioC interoperability track overview, *Database*, Volume 2014, 2014, bau053, https://doi.org/10.1093/database/bau053
[9]	Friedman, C., Johnson, S.B. (2006). Natural Language and Text Processing in Biomedicine. In: Shortliffe, E.H., Cimino, J.J. (eds) Biomedical Informatics. Health Informatics. Springer, New York, NY. https://doi.org/10.1007/0-387-36278-9_8
[10]	*Glossary of Deep Learning: Word Embedding*. Medium. (2017). Retrieved 10 May 2022, from https://medium.com/deeper-learning/glossary-of-deep-learning-word-embedding-f90c3cec34ca.
[11]	*Google Colab: An Online Jupyter Notebook That You Should Definitely Try*. Medium. (2022). Retrieved 10 May 2022, from https://towardsdatascience.com/google-colab-an-online-jupyter-notebook-that-you-should-definitely-try-2572a3d4afb6.
[12]	*IBM Cloud Learn Hub*. Ibm.com. Retrieved 10 May 2022, from https://www.ibm.com/cloud/learn.
[13]	Islamaj Dogan R, Kim S, Chatr-Aryamontri A, Chang CS, Oughtred R, Rust J, Wilbur WJ, Comeau DC, Dolinski K, Tyers M. The BioC-BioGRID corpus: full text articles annotated for curation of protein-protein and genetic interactions. Database (Oxford). 2017 Jan 10;2017:baw147. doi: 10.1093/database/baw147. PMID: 28077563; PMCID: PMC5225395.
[14]	*Jupyter Notebook: An Introduction – Real Python*. Realpython.com. (2019). Retrieved 10 May 2022, from https://realpython.com/jupyter-notebook-introduction/.
[15]	Medium. 2022. *The story of BERT-ian era*. [online] Available at: <https://medium.com/analytics-vidhya/the-story-of-BERT-ian-era-4fc455f0cfc> [Accessed 10 May 2022].
[16]	Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: an introduction. *Journal of the American Medical Informatics Association : JAMIA*, *18*(5), 544–551. https://doi.org/10.1136/amiajnl-2011-000464
[17]	*Smaller, faster, cheaper, lighter: Introducing DilBERT, a distilled version of BERT*. Medium. (2019). Retrieved 10 May 2022, from https://medium.com/huggingface/distilBERT-8cf3380435b5.
[18]	*Understanding the Confusion Matrix and How to Implement it in Python*. Medium. (2020). Retrieved 10 May 2022, from https://towardsdatascience.com/understanding-the-confusion-

	matrix-and-how-to-implement-it-in-python-319202e0fe4d.
[19]	*What is the Jupyter Notebook? — Jupyter/IPython Notebook Quick Start Guide 0.1 documentation*. Jupyter-notebook-beginner-guide.readthedocs.io. (2015). Retrieved 10 May 2022, from https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html.
[20]	*What Is Machine Learning (ML)?*. UCB-UMT. Retrieved 10 May 2022, from https://ischoolonline.berkeley.edu/blog/what-is-machine-learning/.
[21]	*What are single nucleotide polymorphisms (SNPs)?: MedlinePlus Genetics*. Medlineplus.gov. Retrieved 21 May 2022, from https://medlineplus.gov/genetics/understanding/genomicresearch/snp/?fbclid=IwAR1quSQZ8wgYGCUE7nPNm6YhqM7geJcWvL142V0amFAewMaLD5mpmM-Ajps.