A Text Segmentation technique based on Language Models

Phil Lodovico Riccardo Ranzato
Student Id: 899960

Advisor: Prof. Marco Brambilla

Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

This thesis is submitted for the degree of
Master of Science in Computer Science and Engineering

December 2019

I would like to dedicate this thesis to my family and friends,
who always make me feel loved.

Ringraziamenti

Ringrazio il professor Marco Brambilla per avermi guidato e supportato nella stesura di questo lavoro.

Ringrazio anche il Politecnico di Milano e tutti i professori che sono stati in grado di farmi appassionare alle materie da loro trattate e che mi hanno guidato in questo percorso.

Ringrazio inoltre la mia famiglia per avermi fornito la possibiltà di finire gli studi.

Ringrazio i miei amici dell'università con cui ho affrontato questa avventura e a cui auguro ogni bene futuro. Grazie di avermi aiutato nel momento del bisogno e di aver rallegrato questo periodo della mia vita.

Ringrazio in modo speciale i miei amici al di fuori dell'università, grazie per avermi supportato (e sopportato) in qualsiasi momento.

Da ultima, ma prima per importanza, ringrazio la persona a me più cara, Sara Signorile, la persona che più mi è stata vicino e che mi ha dato la forza nei momenti difficili, grazie alla tua comprensione, al tuo supporto, grazie di tutto.

Senza di voi niente di ciò sarebbe stato possibile,

Grazie

Abstract

In an era in which huge amounts of data are generated every day, it becomes necessary to be able to manipulate them in order to obtain valid and usable information. Processing natural language has always been a central topic of artificial intelligence, both for the fundamental role that covers language in everyday life and for the enormous potential it offers. This thesis aims to analyze the potential of a recent model of language representation called BERT, in particular in a subset of the so-called Natural Language Processing, called Text Segmentation. Text Segmentation covers a fundamental role in Natural Language Processing since it can be integrated with a vast multitude of other functions belonging to Natural Language Processing, such as Text Summarization or image analysis. BERT has proved to be a very powerful tool that allows obtaining discrete results even with a small amount of data for medium segments, while, for rather large and generic segments, the results are scarce. For this reason, the approach of this thesis is based on the use of BERT to be able to segment the text according to two different scenarios: data coming from news articles, characterized by segments of average length, and data coming from books, whose chapters appear to be segments of considerable size.

Abstract

In un'era in cui ogni giorno vengono generate ingenti quantità di dati, diventa necessario riuscire a manipolarli in modo da ottenere delle informazioni valide e utilizzabili. Processare il linguaggio naturale è sempre stato un argomento di centrale interesse dell'intelligenza artificiale, sia per il ruolo fondamentale che copre il linguaggio nella vita di tutti i giorni, sia per le enormi potenzialità che offre.

Questa tesi si pone come obiettivo quello di analizzare le potenzialità di un recente modello di rappresentazione del linguaggio chiamato BERT, in particolare in un sottoinsieme del cosiddetto Natural Language Processing (elaborazione del linguaggio naturale), chiamato Segmentazione del Testo. La Segmentazione del Testo copre un ruolo fondamentale nel Natural Language Processing, poiché può essere integrata a una vasta moltitudine di altre funzioni appartenenti al Natural Language Processing, come al Text Summarization (riassunto del testo) o all'analisi di immagini.

BERT si è dimostrato essere uno strumento molto potente che permette di ottenere discreti risultati anche con un'esigua quantità di dati per segmenti medi, mentre, per segmenti piuttosto grandi e generici, i risultati sono scarsi. Per questo motivo, l'approccio di questa tesi si basa sull'utilizzo di BERT per poter segmentare il testo secondo due diversi scenari: dati provenienti da articoli di news, caratterizzato da segmenti di lunghezza media, e dati provenienti da libri, i cui capitoli risultano essere segmenti di dimensione notevole.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and problem statement

Unstructured data, especially text, images and videos contain a wealth of information. However, due to the inherent complexity in processing and analyzing this data, people often refrain from spending extra time and effort to get out of structured datasets to analyze these unstructured sources of data, which can be a potential gold mine [1].

Nowadays the number of internet users is in constant growth and it is estimated equal to 4.42 billion users. Everyday tasks like using Google Translate, Word Processors like Microsoft Word or personal assistants such as OK Google, Siri, Cortana, and Alexa rely on the understanding of the human language. The difficulty behind these tasks comes from the complexity and the nature of the human language itself.

The rules that dictate the passing of information using natural languages are not easy for computers to understand. Some of these rules can be high-leveled and abstract: such as rhetoric figures or the use of sarcasm; or some of these rules can be low-leveled: the use of the character "s" to signify the plurality of items. Comprehensively understanding the human language requires understanding both the words and how the concepts are connected to deliver the intended message.

While humans can easily master a language, the ambiguity and imprecise characteristics of the natural languages are what make the understanding of the language difficult for machines to implement. Improving the computer's ability to understand language leads to better integration of everyday tools. As consumers become more familiar with NLP and time-saving benefits, they will probably be able to adopt natural language processing at home and in the office for other tasks.

[1]https://towardsdatascience.com/a-practitioners-guide-to-natural-language-processing-part-i-processing-understanding-text-9f4abfd13e72

In this context, several methods have been proposed to address the problem of distinguishing what a user said and processing what a user meant. To converse with humans, a program must understand syntax, semantics, morphology, and pragmatics. The branch of artificial intelligence that deals with the interaction between humans and computers using the natural language is Natural Language Processing. Used to get computers closer to a human-level understanding of language, the ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable [2]. Most NLP techniques rely on machine learning to derive meaning from human languages. Computers are great at working with standardized and structured data, but they lack the same intuitive understanding of natural language that humans do. Since humans don't communicate with "structured data", they can't understand the real meaning behind the language, but the application of Deep Learning has enabled programmers to write programs capable to perform things like language translation, semantic understanding, and text summarization. Until recently, much of the Natural Language Processing was done using Recurrent Neural Networks (RNNs) and Long-Short Term Memory (LSTMs).

1.2 Proposed solution

The goal of this thesis is to propose a deep evaluation of the newest method to identify and extract natural language rules. This method of pre-training language representations is called BERT, Bidirectional Encoder Representations from Transformer, which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. The proposed work focuses on evaluating its capabilities against non-trivial tasks, comparing the results obtained with previous architectures. In this context, BERT outperforms previous methods because it is the first unsupervised, deeply bidirectional system for pre-training NLP. Unsupervised means that BERT was trained using only a plain text corpus, which is important because an enormous amount of plain text data is publicly available on the web in many languages. Pre-trained representations can also either be context-free or contextual, and contextual representations can further be unidirectional or bidirectional. The analyses proposed focuses on plain texts from two different sources: books and articles. In the first case, the approach collects data on books to create a chapter-based structure for each book. Then, this pre-processed data follows the classic text mining steps to create a dataset of a couple of sentences to be fed to BERT. For the same purpose, the approach of the second scenario retrieves the articles to create an article-based structure from which a couple of sentences will be chosen to set up the second dataset.

[2]https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b32

Since BERT is a pre-trained language model, the aim of these datasets is only to fine-tune the BERT model. BERT is fine-tuned to be able to distinguish whether a sentence is the start of a new chapter or a new article. The same approach can be used to identify any other logical divisions in large text collections.

1.3 Structure of the thesis

The structure of the thesis is as follows:

- Chapter 2 defines and explains the background knowledge and concepts that are related to the work that has been performed for this thesis.
- Chapter 3 presents the past works that are related to this thesis in the problem they try to answer and the solution they propose.
- Chapter 4 contains a high-level description of the employed methods that are used in this thesis.
- Chapter 5 describes the source codes and implementations of the used methods.
- Chapter 6 presents the results of the experiments and discusses these outcomes.
- Chapter 7 concludes this report by summarizing the work, doing a critical discussion and advises the possible future work.

Chapter 2

Background

This chapter introduces the theoretical background necessary to properly understand the rest of the thesis.

2.1 Text mining

The amount of data that can be found over the internet is increasing at an exponential rate day by day. One of the most used types of data, that is flowing over the internet, is the textual one in the form of digital libraries, repositories and other textual information such as blogs, the social media network, e-mails, books, and articles.
Determine appropriate patterns and trends to extract valuable knowledge from textual data is a challenging task that requires time and effort. Text mining is a process to extract interesting and significant patterns to explore knowledge from textual data sources.
Text mining is a multi-disciplinary field based on information retrieval, data mining, machine learning, statistics, and computational linguistics [5]. Several text mining techniques like summarization, classification, and clustering, can be applied to extract knowledge. Text mining deals with natural language text which is stored in a semi-structured and unstructured format [22]. For a text mining process, or in general, a data mining process, several main phases are defined. These steps are addressed by the CRISP-DM model that aims to make large data mining projects, less costly, more reliable, more repeatable, more manageable, and faster.
The CRISP-DM model identifies six phases:

1. Business Understanding: this initial phase focuses on understanding the project objectives and requirements from a business perspective, and then converting this knowledge into a data mining problem definition.

2. Data Understanding: after an initial data collection, this aims to get familiar with the data, to identify data quality problems, to discover first insights into the data, or to detect interesting subsets to form hypotheses for hidden information.

3. Data Preparation: the data preparation phase covers all activities to construct the final dataset (data that will be fed into the modeling tool(s)) from the initial raw data. Data preparation tasks are likely to be performed multiple times, and not in any prescribed order.

4. Modeling: various modeling techniques are selected and applied, and their parameters are calibrated to optimal values. Typically, there are several techniques for the same data mining problem type.

5. Evaluation: before proceeding to the final deployment of the model, it is important to more thoroughly evaluate the model, and review the steps executed to construct the model, to be certain it properly achieves the business objectives.

6. Deployment: the creation of the model is generally not the end of the project. Usually, the knowledge gained will need to be organized and presented in a way that the customer can use it.

2.1.1 Differences between supervised and unsupervised tasks

Within the field of machine learning, there are two main types of tasks: supervised and unsupervised. The main difference between the two types is that supervised learning is done using a ground truth: a prior knowledge of what the output values for the samples should be. Therefore, the goal of supervised learning is to learn a function that, given a sample of data and desired outputs, best approximates the relationship between input and output observable in the data. On the other hand, unsupervised learning does not have labeled outputs, so its goal is to infer the natural structure present within a set of data points. Hence, supervised learning is done in the context of classification, while unsupervised learning most common tasks are clustering, exploratory analysis, and dimensionality reduction.

2.1.2 Bag of Words

In Natural Language Processing tasks, a common way to represent text data is Bag-of-Words. It is a way of extracting features from the text for use in machine learning algorithms. In this approach, for each observation are used tokenized words to find out the frequency of each token. Each sentence is treated as a separate document, and a list of all words from all the documents is made, excluding the punctuation. Then, the next step is creating the vectors. Vectors convert text that can be used by the machine learning algorithm. The process of converting NLP text into numbers is called vectorization in machine learning. Different ways to convert text into vectors are used:

- Counting the number of times each word appears in a document
- Calculating the frequency that each word appears in a document out of all the words in the document like TF-IDF [18].

These vectors are measures of presence of words or weights.

2.2 Train and test sets

In Machine Learning, the aim is to try to create a model to predict the test data. So the training data are used to adjust the parameters of the model to fit the model. The validation set is applied for hyper-parameter tuning (regularization, early stopping, drop-off, learning rate) to reduce variance, improving the generalization capacity of your model. The validation data is not seen by the model during training. Test data is used to provide an unbiased evaluation of a final model. Like the Validation set, the Test set, also called the holdout set, is not seen by the model. All the three sets' data should come from the same distribution

2.2.1 Cross-Validation

Cross validation (CV) is one of the techniques used to test the effectiveness of machine learning models, it is also a resampling procedure used to evaluate a model if data are limited. To perform a CV, a portion of data is kept aside to be used, when the training is completed, for testing and validating purposes. Several Cross-Validation techniques are used:

- Test-Train Split approach: in this approach, the complete data are randomly split into training and test sets, ideally split the data into 70:30 or 80:20.
- K-Fold Cross-Validation: this approach is generally preferred since it results in a less biased model compared to other methods. In k-fold cross-validation, the training data is randomly split into k mutually exclusive subsets (the folds) of approximately equal size. Then use as training data $k-1$ of the subsets and then tested on the subset left out. This procedure is repeated k times and in this fashion, each subset is used for testing once. Averaging the test error over the k trials gives an estimate of the expected generalization error [6].

- Leave-One-Out Cross-Validation: LOOCV can be viewed as an extreme form of k-fold cross-validation in which k is equal to the number of examples. In LOOCV, one example is left out for testing each time, and so the training and testing are repeated k times. It is known [10] that the LOOCV procedure gives an almost unbiased estimate of the expected generalization error.

2.3 Neural Network

Artificial Neural networks, or ANNs, are processing devices (algorithms or actual hardware) that are loosely modeled after the neuronal structure of the mammalian cerebral cortex but on much smaller scales.

Definition 1 (Neural Network) *a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs [3].*

Neural networks are typically organized in layers, which are made up of a number of interconnected nodes containing an activation function. Patterns are presented to the network via the input layer, which communicates to one or more hidden layers where the actual processing is done via a system of weighted connections. Most ANNs contain some form of learning rule which modifies the weights of the connections according to the input patterns that it is presented with.
A category of ANNs uses back-propagation as a kind of learning rule. Back-propagation stands for backward propagation of error. With the back-propagation, Learning is a supervised process that occurs with each cycle or epoch through a forward activation flow of outputs, and the backward error propagation of weight adjustments. Backpropagation performs gradient descent in order to achieve a global minimum. The global minimum is that theoretical solution with the lowest possible error. Indeed, in most problems, the solution space is quite irregular which may cause the network to settle down in a local minimum which is not the best overall solution. Since the nature of the error space cannot be known a priori, neural network analysis often requires a large number of individual runs to determine the best solution. Most learning rules have built-in mathematical terms to assist in this process which controls the speed and the momentum of the learning. The speed of learning is actually the rate of convergence between the current solution and the global minimum. Momentum helps the network to overcome local minima in the error surface and settle down at or near the global minimum.

2.4 Fine-Tuning

Fine-tuning is the process in which the parameters of a trained model must be adjusted very precisely while trying to validate that model taking into account a small data set that does not belong to the train set. That small validation data set comes from the same distribution as the dataset used for the training of the model. Hence, fine-tuning means taking weights of a trained neural network and use it as initialization for a new model being trained on data from the same domain.
Basically, fine-tuning is used for situations where the dimension of data or computation time or maybe computation capacity are not enough to train a whole network from scratch. In this way, usually the last layers of a pre-trained network are replaced with new ones. The first layers have already found the needed good features and last layers try to classify: for this reason, the last layers might be replaced in order to fit the current labels.

2.4.1 Fine-Tuning Techniques

1. A common practice is to truncate the last layer (softmax layer) of the pre-trained network and replace it with a new softmax layer that is relevant to the current problem.

2. Another effective technique is using a smaller learning rate to train the network. Since the pre-trained weights are expected to be quite good already as compared to randomly initialized weights, there is no need to distort them too quickly and too much. A common practice is to make the initial learning rate 10 times smaller than the one used for scratch training.

3. It is also a common practice to freeze the weights of the first few layers of the pre-trained network. This is because the first few layers capture universal features like curves and edges that are also relevant to our new problem. We want to keep those weights intact. Instead, we will get the network to focus on learning dataset-specific features in the subsequent layers.

2.5 Transfer Learning

Transfer Learning or Domain Adaptation is related to the difference in the distribution of the train and test set. *Insufficient training data* is an inescapable problem in some special domains. The collection of data is complex and expensive making it extremely difficult to build a large-scale, high-quality annotated dataset. Transfer learning relaxes the hypothesis that the training data must be independent and identically distributed (i.i.d.) with the test data, which motivates the use of transfer learning against the problem of insufficient training data. In transfer learning, the training data and test data are not required to be i.i.d., and the model in the target domain does not need to be trained from scratch, which can significantly reduce the demand for training data and training time in the target domain.

Definition 2 (Transfer Learning) *Given a learning task T_t based on D_t, and we can get the help from D_s for the learning task T_s. Transfer learning aims to improve the performance of predictive function $f_t(\cdot)$ for learning task T_t by discover and transfer latent knowledge from D_s and T_s, where $D_s \neq D_t$ and/or $T_s \neq T_t$. In addition, in the most case, the size of D_s is much larger than the size of D_t, $N_s \gg N_t$ [19].*

It is something broader than Fine-tuning, which means that it is known apriori that the train and test come from different distribution and the objective is to tackle this problem with several techniques depending on the kind of difference, instead of just trying to adjust some parameters
In the real world, there are many examples of transfer learning: learning to ride a motorbike might help to ride a car or learning to recognize an apple might help to recognize a pear. The study of Transfer learning is motivated by the fact that people can intelligently apply knowledge learned previously to solve new problems faster or with better solutions.

2.5.1 Transfer Learning Categories

Based on the definition of transfer learning, it is possible to distinguish the relationships between traditional machine learning and various transfer learning settings. Transfer Learning can be categorized under three main sub-settings: inductive transfer learning, transductive transfer learning, and unsupervised transfer learning, based on different situations between the source and target domains and tasks.

Inductive Transfer Learning In this setting, the target task is different from the source task and the source and target domains can be the same or not [7]. Some labeled data in the target domain are required to induce an objective predictive model $f_t(\cdot)$ to be used in the target domain. Moreover, it is possible to further categorize according to different situations of labeled and unlabeled data in the source domain:

1. *A lot of labeled data in the source domain are available.* This aims at achieving high performance in the target task by transferring knowledge from the source task while multitask learning tries to learn the target and source task simultaneously.

2. *No labeled data in the source domain are available.* In this case, the inductive transfer learning setting is similar to the *self-taught learning* setting. In the self-taught learning setting, the label spaces between the source and target domains may be different, which implies the side information of the source domain cannot be used directly [17].

Transductive Transfer Learning In this setting, the source and target tasks are the same, while the source and target domains are different. No labeled data in the target domain are available while a lot of labeled data in the source domain are available. Moreover, according to different situations between the source and target domains, it is possible to further categorize into two different cases:

1. The feature spaces between the source and target domains are different, $X_s \neq X_t$.
2. The feature spaces between domains are the same, $X_s = X_t$, but the marginal probability distributions of the input data are different, $P(X_s) \neq P(X_t)$.

The latter case of the transductive transfer learning setting is related to domain adaptation for knowledge transfer in text classification [8].

Unsupervised Transfer Learning In the unsupervised transfer learning the target task is different from but related to the source task. The unsupervised transfer learning focus on solving unsupervised learning tasks in the target domain, such as clustering, dimensionality reduction, and density estimation [4], [21]. In this case, there are no labeled data available in both source and target domains in training.

2.6 Recurrent Neural Network

Recurrent Neural Networks (RNNs) add an interesting twist to basic neural networks. A vanilla neural network (one that uses Back-propagation) takes in a fixed size vector as input which limits its usage in situations that involve an input with no predetermined size. RNNs are designed to take a series of inputs with no predetermined limit on size.

RNNs can take one or more input vectors and produce one or more output vectors and the outputs are influenced not just by weights applied on inputs like a regular neural network, but also by a hidden state vector representing the context based on prior inputs and outputs. This effective way of leveraging the relationships between one input and its surrounding neighbours is called *Parameter Sharing*. So, the same input could produce a different output depending on previous inputs in the series. In summary, in a vanilla neural

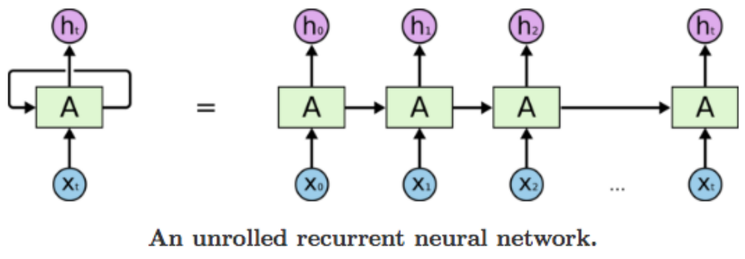


Figure 2.1 Unrolled recurrent neural network [9]

network, a fixed size input vector is transformed into a fixed size output vector. Such a network becomes "recurrent" when you repeatedly apply the transformations to a series of given input and produce a series of output vectors. They are networks with loops in them, allowing information to persist.

2.6.1 Bidirectional Recurrent Neural Network

Bidirectional recurrent neural networks (BiRNNs) connect two hidden layers running in opposite directions to a single output, allowing them to receive information from both past and future states. BiRNNs are trained with similar algorithms as RNNs since the two-directional neurons do not interact with one another. Unlike standard recurrent neural networks, BiRNNs are trained to predict both the positive and negative directions of time simultaneously. BiRNNs split the neurons of a regular RNN into two directions, one for the forward states (positive time direction), and another for the backward states (negative time direction). By employing two-time directions simultaneously, input data from the past and future of the current time frame can be used to calculate the same output. Which is the opposite of standard recurrent networks that requires an extra layer for including future information.

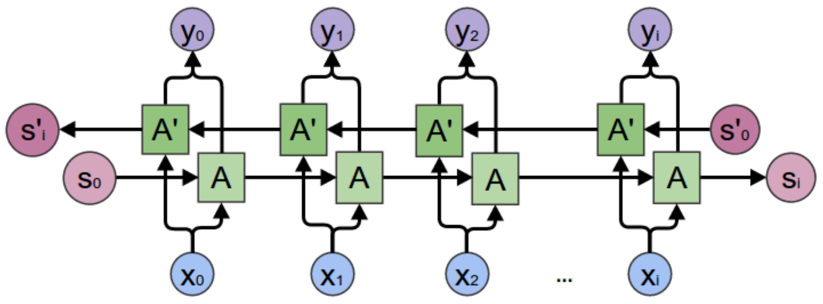


Figure 2.2 Bidirectional recurrent neural network [15]

2.7 Pre-trained language representation

There are two existing strategies for applying pre-trained language representations to downstream tasks: feature-based, like using tasks-specific architectures that include the pre-trained representations as additional features; or fine-tuning, like introducing minimal task-specific parameters, and then training on the downstream tasks by simply fine-tuning the pre-trained parameters. Both these approaches share the same objective function during pre-training, where they use unidirectional language models to learn general language representations.

2.8 BERT

2.8.1 Introduction

Google's BERT language representation model, which stands for Bidirectional Encoder Representations and Transformers, is designed to pre-train deep bidirectional representations from an unlabeled text by jointly conditioning on both left and right context in all layers. The pre-trained BERT representations can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications. Some BERT key ideas are the use of the transformer architecture and the unsupervised pre-training. Using a transformer architecture means that BERT is a sequence model that forgoes the sequential structure of RNNs for a fully attention-based approach; while being an unsupervised pre-trained model means that its weights are learned in advance through two unsupervised tasks, that are the masked language modeling and the next sentence prediction. Thus, BERT does not need to be trained from scratch, but starting from the pre-trained model, its weights are fine-tuned. Bert is not like traditional attention models that use a flat attention structure over the hidden states of an RNN, but uses multiple layers of attention, and also incorporates multiple "attention heads" in every layer. BERT major contribution is to have generalized a deep bidirectional architecture. Like OpenAi GPT [16], as activation function BERT uses *gelu*, rather than *relu*.

BERT is the first fine-tuning based representation model that achieves state-of-the-art performance on a large suite of sentence-level and token-level tasks, outperforming many systems with task-specific architectures (neural or not neural, based on a sentence or paragraph embeddings).

2.8.2 BERT Architecture: The Transformer

BERT Architecture consists of a multi-layer bidirectional **Transformer** encoder. The BERT Transformer uses bidirectional self-attention that learns contextual relations between words (or sub-words) in a text. The Transformer includes an Encoder, that reads the input, and a Decoder, that produces a prediction for the task.

Difference between Transformer Encoder and Decoder: bidirectional Transformer is often referred to as a "Transformer encoder" while the left-context-only version is referred to as a "Transformer decoder" since it can be used for text generation.

The Transformer is a model that uses attention to boost the speed with which these models can be trained [20]. It is like a black box that, given a sentence A, returns a sentence B. Taking a look at a simple architecture this black box can be seen as a group of encoders that feed a group of decoders.
These encoders are all identical in structure, but with different weights. Each of these encoders is broken down into two sub-layers: the first layer is a self-attention layer, a layer that helps the encoder look at other words in the input sentence as it encodes a specific word, its outputs are fed to a feed-forward neural network (the exact same feed-forward network is independently applied to each position); the second layer is a Feed-Forward Neural Network layer. The decoder has the same structure plus another middle-layer: the Encoder-Decoder Attention layer, which helps the decoder focus on relevant parts of the input sentence.
The Transformer Architecture requires to add a classification layer on top of the encoder output, to multiply the output vectors by the embedding matrix transforming them into the vocabulary dimension and to calculate the probability of each word in the vocabulary with softmax.

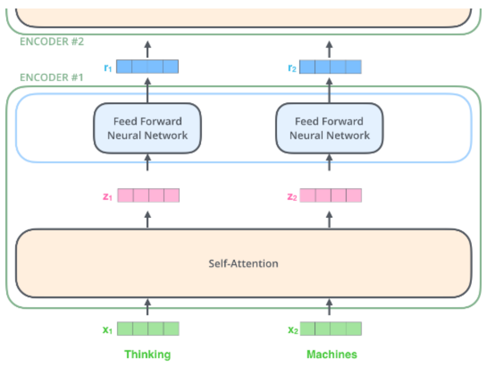
Input Representations

Input representation can unambiguously represent both a single text sentence or a pair of text sentences (Question, Answer) in one token sequence. For a given token, its input representation is constructed by summing the corresponding token, segment and position embeddings. There are two types of special tokens: *CLS and SEP*.
The *CLS* token, which stands for Classification, is the first token of every sequence. The final hidden state (output of Transformer) corresponding to this token is used as the aggregate sequence representation for classification tasks. Ignored for non-classification tasks.
The *SEP* token, which stands for Sentence Pairs Separator, is BERT's way to indicate the separation between two sentences packed together into a single sequence.

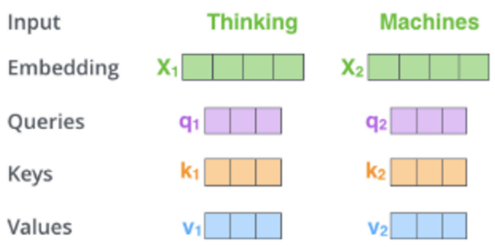
Each input word is then turned into a vector using an Embedding Algorithm, flowing through its own path in the decoder.

The Encoder

An encoder receives a list of vectors as input, processing the list of vectors by passing these vectors into a self-attention layer. After flowing through the encoder, then these vectors go into the feed-forward neural network, and then the Feed-Forward Neural Network sends out the output upwards the next encoder (Figure 2.3a [1]).



(a) Encoder



(b) Self-Attention Vectors

Self-Attention Layer Self-attention associates each word with its real meaning, connecting pronouns with the word to which they refer, for example. This layer allows looking at other positions in the input sequence to gain a better encoding for each word.
Calculating Self-Attention:

1. This layer creates three vectors from each encoder's input vectors, the embeddings of each word. These three vectors are the Query Vector q_i, the Key Vector k_i and the Value Vector v_i (Figure 2.3b). These vectors are created by multiplying the embedding by three matrices trained in the training process and they are abstractions useful for calculating attention.

2. For each word, it calculates the score against each word of the input sequence. This score determines how much focus to place on the other parts of the input sentence for the word. The calculation is done by taking the dot product of the query vector with the key vector of the word to be scored.

3. Divide the scores by the square root of the key vector-dimension d_k.

[1]http://jalammar.github.io/illustrated-transformer/

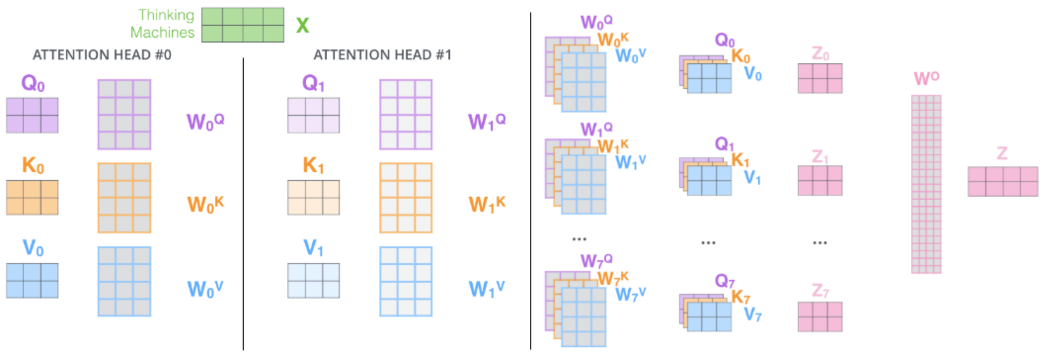


Figure 2.4 Multi-Head Self-Attention

4. Pass the result through a softmax operation that normalizes the scores to have only positive scores that sum up to 1. Clearly, the word i at the i-th position will have the highest softmax score.

5. Multiply each value vector by the softmax score. The intuition is to focus on the word wanted by keeping intact its values while multiplying irrelevant words by tiny numbers to cut their importance.

6. Sum up the weighted value vectors. This produces the output of the self-attention layer (at i-th position).

However, the actual implementation makes calculation in matrix form. Calculate Query Q, Key K and Value V matrices by multiplying a matrix X (of embeddings) with the trained matrices W^Q, W^K, W^V. Using matrices, the step 2 through 6 can be condensed into one formula

$$softmax\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

A further improvement is the *Multi-Headed Attention* (Figure 2.4 [2]) that improves performance of the attention layer in two different ways: expanding model ability to focus on different positions and giving the attention layer multiple representation sub-spaces, by creating multiple sets of Q-K-V matrices. BERT Transformer uses eight attention heads, meaning that for each encoder it has eight sets.

Each set is initialized randomly and after training each set is used to project input embeddings into different subspace. The feed-forward layer is not expecting 8 matrices, but one matrix made of a vector for each word, thus these matrices are condensed into one matrix and then multiplied by an additional weights' matrix W^0. The Transformer handles the order

[2]http://jalammar.github.io/illustrated-transformer/

of words in the sentence by using a *positional encoding* added to each input embedding. Each of these follows a pattern, that the model learns, determining the position of the word. Adding these values provides meaningful distances between embedding vectors.

The Residuals

Each sub-layer, like self-attention layer and FFNN layer, in each encoder has a residual connection around it, followed by a layer-normalization step .

The Decoder

The encoder starts by processing the input sequence, then the output of the last encoder is transformed into a set of attention vectors K and V. These vectors are used by each decoder in its encoder-decoder attention layer. Until a special symbol (indicating that the transformer decoder has completed its output) is reached, the output of each step is fed to the bottom decoder and to the next (like the encoder). Self-Attention in decoder Works slightly different: in the decoder, the self-attention layer is only allowed to attend earlier positions in the output sequence. This is done by masking future positions before the softmax step in the self-attention calculation. The encoder-decoder attention layer works just like the Multi-Headed Self-Attention with one main difference: it creates Queries matrix Q from its below-layer and takes the Keys and Value Matrix from the output of the encoder stack.

The Final Linear and Softmax Layer

The Decoder output is in the form of a vector of floats, that are turned into word by the Linear Layer, which is followed by the softmax layer. The Linear Layer is a simple fully connected neural network that projects the vector produced by the stack of decoders, into a much larger vector called *logits vector*, with the size of the vocabulary. Softmax Layer turns the scores produces by the linear layer into probabilities (all positive and all add up to 1). The cell with the highest probability is chosen and the word associated is produced as the output.

2.8.3 BERT Unsupervised Prediction Tasks

Masked Language Model

BERT improves the fine-tuning based approach by proposing a new pre-training objective: the Masked Language Model. Masked Language Model, also known as MLM, randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked word based only on its context. Unlike left-to-right language model pre-training, the MLM objective allows the representation to fuse the left and the right context, which allows to pre-train a deep bidirectional Transformer. In this MLM is also introduced the next sentence prediction task that jointly pre-trains text-pair representations. These pre-trained representations eliminate the needs of many heavily engineered task-specific architectures. BERT is the first fine-tuning based representation model that achieves state-of-the-art performance on a large suite of sentence-level and token-level tasks, outperforming many systems with task-specific architectures.

Bidirectional conditioning would allow each word to indirectly see itself in a multi-layered context (thus, traditional models are trained only left-to-right or right-to-left). To have a bidirectional training, BERT masks some percentage of the input tokens at random, and then predicts only those masked tokens (referred to as a Cloze [3] in literature). The final hidden vectors corresponding to the mask tokens are fed into an output softmax over the vocabulary, as in a standard Language Model. BERT only predicts the masked words rather than reconstructing the entire input. This allows to obtain a bidirectional approach. Before feeding word sequence into BERT, 15% of the words in each sequence are replaced with a MASK token. This 15% is divided into an 80% that uses MASK token, a 10% with random words and a 10% that uses the original words, this choice comes from different reasons:

- If the MASK tokens are used 100% of the time the model would not necessarily produce good token representations for non-masked words. The non-masked tokens were still used for context, but the model was optimized for predicting masked words.
- If the MASK tokens are used 90% of the time and random words 10% of the time, this would teach the model that the observed word is never correct.
- If the MASK tokens are used 90% of the time and kept the same word 10% of the time, then the model could just trivially copy the non-contextual embedding.

But this approach comes with some disadvantages: using MLM means having a mismatch between pre-training and fine-tuning since the MASK token is never seen during fine-tuning, to mitigate this disadvantage, masked words are not always replaced with the actual MASK token, but the training data generator chooses 15% of tokens at random; only 15% of tokens are predicted in each batch, thus more pre-training steps may be required for the model to converge.

[3]*Cloze test*: a test of reading comprehension that involves having the person being tested supply words which have been systematically deleted from a text

Next Sentence Prediction

BERT pre-trains a binarized next sentence prediction task that can be trivially generated from any monolingual corpus. On the pre-training, each couple of sentences is labeled with a variable *IsNextSequence*. In the BERT training process, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sequence in the original document. This is done with [CLS] and [SEP] tags. The prediction follows these steps: the entire input sequence goes through the Transformer model, then the output of the [CLS] token is transformed into a 2×1 shaped vector, using a simple classification layer (learned matrices of weights and biases) and the last step is the calculation of the probability of *IsNextSequence* with softmax. When training the BERT model, Masked LM and Next Sentence Prediction are trained together, with the goal of minimizing the combined loss function of the two strategies.

2.8.4 BERT Attention Patterns

In all the patterns, BERT essentially learned a sequential update pattern (explicitly encoded in architectures such as Recurrent Neural Networks).

Definition 3 (Attention) *positive combinations across neurons, greater when the number of neurons increases.*

Attention to Next Word This pattern appears over multiple layers of the model, emulating the recurrent updates of an RNN. All the attention is focused on the next word in the input sequence, except at the delimiter tokens. Figure 2.5[4] provides a visual example of the next word attention pattern. The SEP token disrupts the next-token attention pattern since its attention is directed to the CLS: this pattern operates primarily within each sentence. This attention pattern enables BERT to capture sequential relationships.

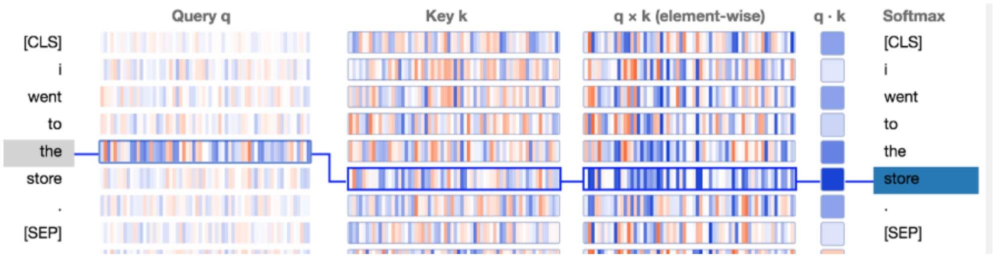


Figure 2.5 Next Word Attention Pattern

[4]https://towardsdatascience.com/deconstructing-bert-part-2-visualizing-the-inner-workings-of-attention-60a16d86b5c1

Attention to Previous Word Most of the attention is directed to the previous token in the sequence. Some attention is also dispersed to other tokens, especially the SEP tokens. This pattern is related to a sequential RNN, in particular, a forward RNN.

Attention to Identical/Related Word Attention is paid to identical or related words, including the source word itself, dispersing attention over many different words.

Attention to other Words Predictive of Word Attention is paid to identical or related words in the other sentence. Particularly helpful for next sentence prediction task, since it helps to identify relationships between sentences.

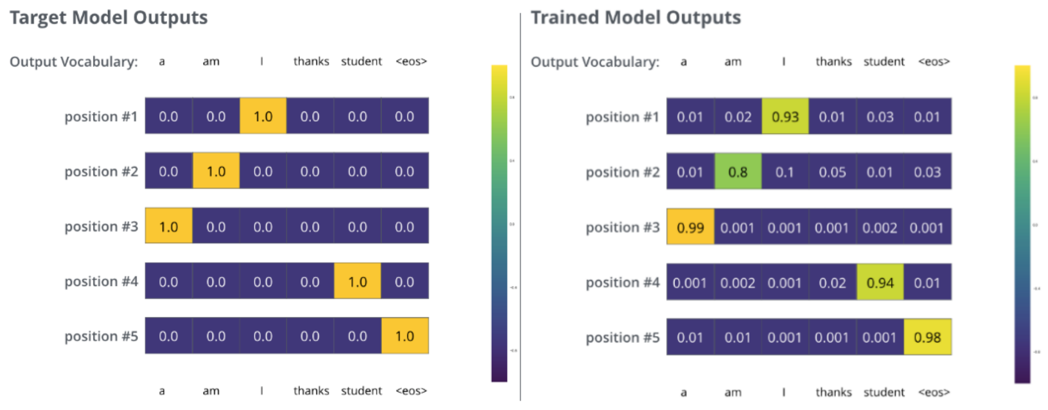
Attention to Delimiter Tokens Attention seems to be directed to other words that are predictive of the source word, excluding the source word itself. The attention of the word that will predict is on the word to be predicted and vice-versa. The query vectors match the SEP key vector along with the same set on neurons. The result is the SEP-focused attention pattern.

Bag of Words Attention pattern Most of the attention is directed to the delimiter tokens, either the CLS token or the SEP tokens. A way for the model to propagate sentence-level state to the individual tokens. Thus, in this pattern, attention is divided evenly across all words in the same sentence.

BERT is essentially computing a Bag-of-Words embedding by taking an almost unweighted average of the word embeddings (value vectors). A clear pattern is that, for each query, a small number of neurons dominate the calculation of the attention scores:

- When query and key vector are in the same sentence, the product shows high values at these neurons.
- When query and key vector are in different sentences, the product is strongly negative at these same positions.
- When query and key are both from the same sentence, they tend to have values with the same sign along the active neurons, resulting in a positive product.
- When the query is from one sentence and the key is from another, the same neurons tend to have values with opposite signs, resulting in a negative product.

So thanks to its sentence-level embeddings added to the input layer, BERT has achieved its own Concept of Sentence. Queries and Keys acquire sentence-specific values thanks to CLS and SEP tokens (sentence-level embeddings).



Target Model Outputs

Output Vocabulary:	a	am	I	thanks	student	<eos>
position #1	0.0	0.0	1.0	0.0	0.0	0.0
position #2	0.0	1.0	0.0	0.0	0.0	0.0
position #3	1.0	0.0	0.0	0.0	0.0	0.0
position #4	0.0	0.0	0.0	0.0	1.0	0.0
position #5	0.0	0.0	0.0	0.0	0.0	1.0

Trained Model Outputs

Output Vocabulary:	a	am	I	thanks	student	<eos>
position #1	0.01	0.02	0.93	0.01	0.03	0.01
position #2	0.01	0.8	0.1	0.05	0.01	0.03
position #3	0.99	0.001	0.001	0.001	0.002	0.001
position #4	0.001	0.002	0.001	0.02	0.94	0.01
position #5	0.01	0.01	0.001	0.001	0.001	0.98

Figure 2.6 Probability Vector Before and After Training

2.8.5 BERT Training

Since the training is done on a labeled training dataset, it is possible to compare its output with the actual correct output. The size of the vector to be used is equal to the size of the output vocabulary, having a position for each word of the vocabulary (*one-hot encoding*).

Loss Function The goal is to have a probability distribution indicating the correct output. To compare two probability distribution one can simply subtract one from the other using the well known cross-entropy ($H(p,q) = E_p - log(q)$) or the Kullback–Leibler divergence ($H(p,q) = H(p) - D_{KL}(p||q)$, where $H(p)$ is the entropy of p). Each probability distribution has to be represented by a vector of the size of the vocabulary and for each word in the output vocabulary, there has to be the highest probability at the cell associated. Each set of probabilities ends with a last output distribution called <*eos*> (Figure 2.6 [5]).

After the training, the expected probabilities will be different. There are different ways to proceed:

- *Greedy Decoding*: since the model produces the outputs one at a time, it selects the highest probability word, throwing away everything else.
- *Beam Search*: hold the two top words, run two more times the model, each time assuming the first output position is each of them. The word producing fewer error in both configurations is kept.

[5]http://jalammar.github.io/illustrated-transformer/

Chapter 3

Related Work

This chapter explains the existing works in the literature that have been the basis or have influenced the making of this thesis. A lot of studying has been done in Natural Language Processing and, in particular, the focus of the thesis is concerned in a branch of Natural Language Processing called Text Segmentation. In the following sections, the foundations of this field are presented in addition to the meaningful studies that have been the basis of this work.

3.1 Text Segmentation

Text segmentation is a method of splitting a document into smaller parts, which are usually called segments. It is widely used in text processing. Each segment has a relevant meaning. Those segments are categorized as word, sentence, topic, phrase or any information unit depending on the task of the text analysis [13]. As introduced above, Text Segmentation is the task defined as the process of segmenting a chunk of text into meaningful sections based on their topical continuity. Text segmentation is one of the fundamental Natural Language Processing (NLP) problems which finds its use in many tasks like summarization, passage extraction, discourse analysis, Question-Answering, context understanding, and document noise removal.

Previous methods for this task require manual feature engineering, huge memory requirements and large execution times. The complexity of text segmentation varies with the type of text and writing styles. In some cases, context is a very important signal for the task, while in other cases, dependence on context may be minimal.

Multiple supervised and unsupervised methods have been proposed to tackle this complexity. Many unsupervised methods are heuristic, while ad-hoc methods need huge memory, have large execution times, and do not generalize well across multiple text types. Supervised methods require labeled data and often the performance of such systems comes at the cost of hand-crafted highly tuned feature engineering.

To identify topic boundaries in order to segment a document, there are many approaches which may be used in conjunction. The boundaries might be apparent from paragraphs or by words commonly used to introduce a new subject [14].

It is also possible to identify topic changes by analysing the lexical properties in a text. A difference in the lexical cohesion inside a text might indicate topic changes. Another cue used to identify topics is the usage of words: a significant vocabulary change may occur when there is a change of topics [11].

Text segmentation can make use of word embedding generating a low dimensional vector representation of words in a document. This approach allows for a more expressive and efficient representation by building low dimensional vectors while maintaining the contextual similarity of words [12].

3.2 Text Segmentation Levels

In 2018, I. Pak and P.L. Teh [13], proposed an evaluation on the metrics used in several studies on text segmentation, aggregating the results in Figure 3.1.

The proposed work falls within the sentence slice.

3.3 Attention-Based Neural Text Segmentation

In 2018, Badjatiya et al. [2] proposed a method that models the text segmentation problem as a binary classification problem. The formal definition of the problem is the following: given a document, the problem is defined with respect to the i-th sentence in the document, as follows.

Given: a sentence s_i with its K sized left-context $\{s_{i-K},...,s_{i-1}\}$ (i.e., K sentences before s_i) and K sized right-context $\{s_{i+1},...,s_{i+K}\}$ (i.e., K sentences after s_i). Here K is the context size.

Predict: whether the sentence s_i denotes the beginning of a new text segment.
This study proposes a neural framework to use the context for learning distinctive features for sentences that mark the beginning of the segment.

The neural model proposed by this study consists: of a data pre-processing layer that provides instances of word embeddings in the form S_i; of a CNN architecture to obtain rich feature representations for each sentence in the left-context, mid-sentence as well as the right-context, generating sentence embeddings; of a BiLSTM network on a smaller sequence

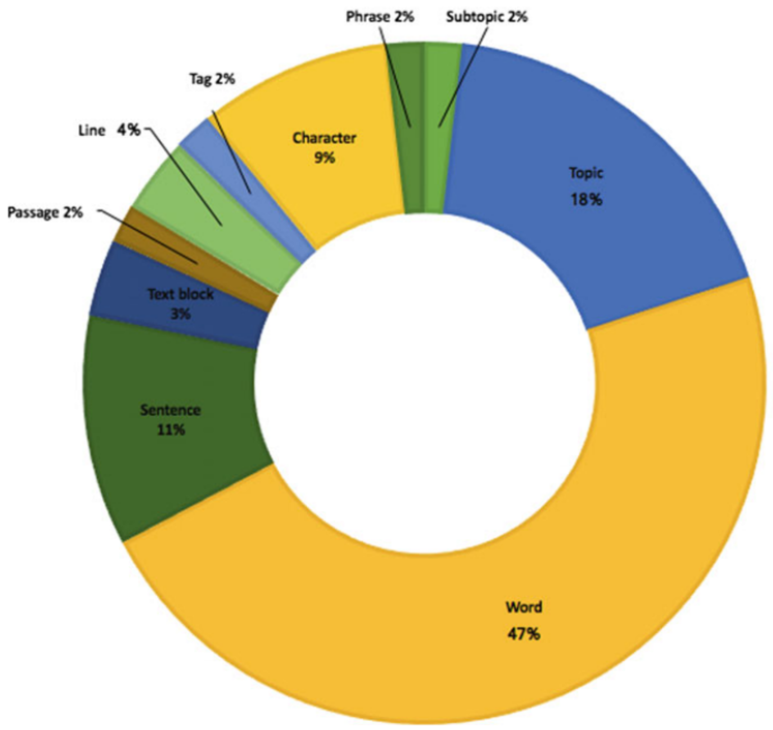


Figure 3.1 Types of Segments in Text Segmentation [13]

that consists of only the main sentence and its neighbours (Stacked BiLSTMs with Attention). To obtain a unified rich feature representation, they use Attention Bidirectional Long-Short Term Memory Network (Attention-BiLSTM) on top of the sequence TS_i, obtained by the CNN architecture.

The resultant context embeddings obtained from a shared encoder for the left-context and right-context and the embedding form id-sentence obtained from separate but similar encoder are passed to a dense fully connected layer followed by a softmax layer.

Chapter 4

Methodology

This chapter presents the main idea of the thesis, together with a high-level view of the logical steps taken along the work done. In order to present the work performed as completely as possible, the main idea of the thesis, the objectives, and the research questions are presented below. Furthermore, it is presented a glossary, containing the keywords used, in order to make the reading clearer. Finally, all the logical steps are explained at a high level, with corresponding inputs and outputs.

4.1 Main Idea

The main idea of the proposed work relies on the assumption that it is possible to use the above explained BERT to perform text segmentation tasks. BERT has been proved to outperform previous methods on a wide array of Natural Language Processing (NLP) tasks with almost no task-specific network architecture modifications or data augmentation. The outperforming results obtained by BERT come from trivial tasks, such as *sentence and sentence-pair classification*, *grammatically correctness* of sentences, *contradiction or entailment* of sentence pairs and *questions and answers* relations between couples of sentences.

The proposed work aims to discover how well the pre-training language representations method BERT performs on a non-trivial task, in particular against Text Segmentation, evaluating its performance against different scenario with different level of complexity.

4.2 Objectives and research goals

The research starts with the purpose of answering the following questions:

- How to use BERT to perform Text Segmentation tasks?
- Is BERT able to achieve effective results?
- Do BERT's performance indexes reflect reality?
- How is it possible to evaluate its performance?

The main goal of this thesis is to verify if BERT's state of art results are reflected in a much more real and non- trivial scenario, as Text Segmentation is.

4.3 Glossary

In order to ease the reading, it is helpful to define some of the terms that will be used throughout the rest of this work.

Transformer: an attention mechanism that learns contextual relations between words (or sub-words) in a text.

Epoch: one cycle through the full training dataset.

MRPC: Microsoft Research Paraphrase Corpus, *Sentence (and sentence-pair) classification tasks*: given a pair of sentences, classify them as paraphrases or not paraphrases.
Example: true if *sentence B* is a paraphrase for *sentence A*.

CoLa: Corpus of Linguistic Acceptability, evaluating grammatically correctness.

GLUE: General Language Understanding Evaluation, is a collection of resources for training, evaluating and analyzing natural language understanding systems.

HPFN: Hyper-Parameters Fine-Tuning, a technique to evaluate which is the configuration of hyper-parameters that leads to better results.

Configuration: set of key-value pairs indicating the BERT hyper-parameters.

Data Quantity: amount of data used during BERT's model training.

Validator: plain text to make a real-world validation of BERT's model.

Experiment: an experiment is a sub-group of a scenario, in which data have been manipulated differently with respect to other experiments.

Scenario: use-case datasets on which the experiments were conducted.

4.4 Solution proposed

This section illustrates a high-level workflow of the solution proposed. All the logical steps are shown in Figure 4.1, moreover, these steps are illustrated aside, in order to give an overall description of each phase, together with its input and outputs. This section aims to give an exhaustive illustration of the steps done, in order to fully understand the following explanations.

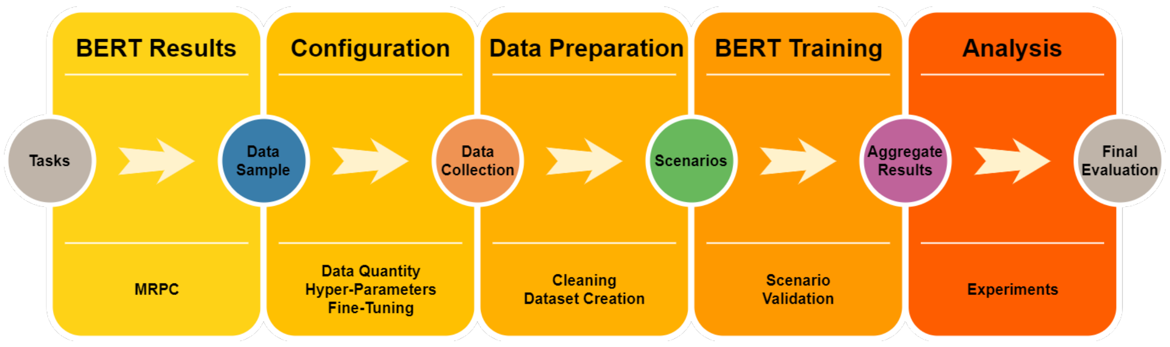


Figure 4.1 Proposed solution workflow

4.4.1 Demonstrating BERT Results

This section proposes to prove that the technology used is really BERT by Google, obtaining similar results on tasks proposed by them. Among all the tasks on which BERT obtained state-of-art results, it was decided to reproduce MRPC, a task based on the GLUE dataset.

4.4.2 Configuration

This phase aims to obtain the best configuration between the quantity of data and hyper-parameters to be used during BERT training, as can be seen in Figure 4.2. Thanks to this phase it was possible to identify which was the best starting setup for all subsequent experiments. The two main variables of interest were the accuracy of BERT, the training time and the F1-Score estimated by BERT. In the following, both phases that make up this phase are explained individually.

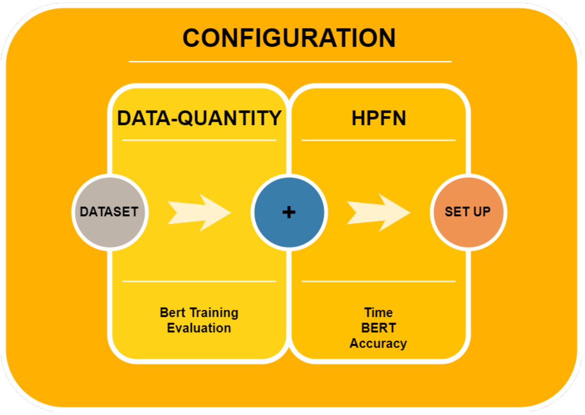


Figure 4.2 Configuration Phase Workflow

Data Quantity Evaluation This sub-phase aims to evaluate the best trade-off between time spent and other performance indexes obtained from BERT training, evaluating the results obtained with different amounts of data. Among all the parameters evaluated, the most important one considered is BERT accuracy. The dataset used as input was created using data from the Harvard Dataverse Database and have been manipulated in order to obtain a format approved by BERT, but no additional pre-processing was done on these data. This phase consists of two main steps, summarized in Figure 4.3.

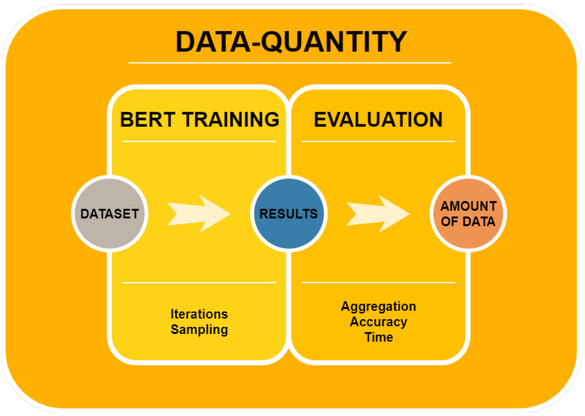


Figure 4.3 Data Quantity Workflow

- **BERT Training**: BERT was trained multiple times for each data quantity, using the default configuration suggested by the official documentation. Each Training iterations was done using the same dataset, but with a different sample for each iteration.

- **Results Evaluation**: after the training, it was possible to study a clear evolution of the variables involved. Thanks to this study it was possible to choose the best amount of data to use based on different parameters, in particular time and accuracy.

Hyper-Parameter Fine-Tuning This sub-phase can be seen at the same level as the one above, as neither has priority over the other. The purpose of this phase is to evaluate the configurations of the hyper modifiable parameters of BERT that are more efficient, leading to a better result. Thanks to this step it was possible to choose the most accurate hyper-parameter configuration, that is to say, that would allow reaching the highest accuracy in the prediction stage. This phase consists of two main steps, summarized in Figure 4.4.

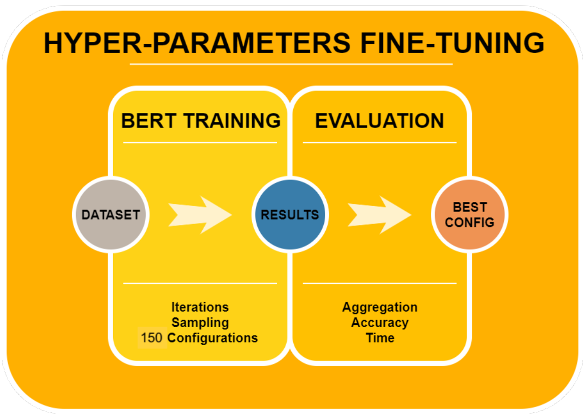


Figure 4.4 Hyper-Parameters Fine-Tuning Workflow

After the decision about which parameters to tune and which values to use, these two main steps were followed:

- **BERT Training**: BERT was trained multiple times for configuration, using the data quantity obtained by the previous step. The decision to evaluate first the data quantity analysis comes from the fact that as the amount of data used for training increases, the time spent also increases. Therefore it was decided to first find the amount of data that produced the best results in the shortest possible time. Each Training iterations was done using the same dataset, but with a different sample for each iteration.

- **Results Evaluation**: after the training, it was possible to study a clear evolution of the configurations involved. Thanks to this study it was possible to choose the best configuration to use based on different parameters, in particular time and accuracy.

4.4.3 Data Preparation

The data preparation phase, summarized by the Figure 4.5, takes as input raw data, of two different types, and transforms them into a format that can be understood and used by BERT in order to fine-tune its last layer and predict as best as possible in the respective scenario. The output format will be described exhaustively in the next chapter.

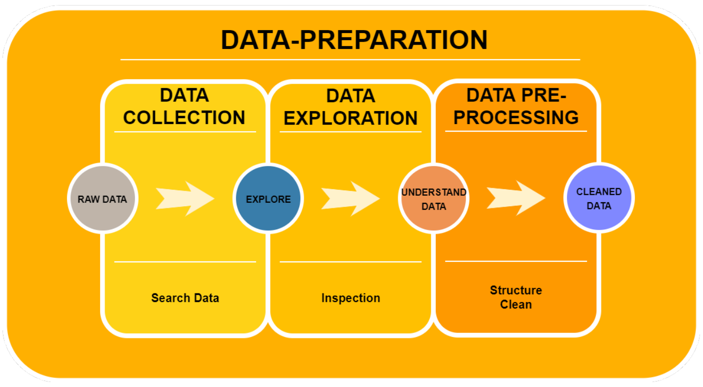


Figure 4.5 Data Preparation Phase Workflow

- **Data collection**: is the very first step, in which data are gathered from two sources. Two different scenarios have been studied: data coming from plain text, representing books and data coming from Harvard Dataverse representatives of news articles. There is no manipulation in this phase, data are collected as they are.
- **Data exploration**: data are studied to find characteristics and potential problems; once again no manipulation of data has been performed.
- **Data Pre-Processing**: consists of all the actions that have to be done to clean and prepare the data.

After this last phase, data have been cleaned and they are ready to be fed to BERT.

4.4.4 BERT Training

In this section, the goal is to provide a high-level view of BERT's fine-tuning, highlighting the logical steps that allowed to obtain structured results. This phase can be considered the most important of all the others, as it allows to obtain a fine-tuned model, adapted to the needs of prediction linked to the experiment in question. As we will see later, this model is able to predict the start of a new segment of text quite efficiently. it is important to remember that the task of segmenting the text is not trivial and consequently the results are to be considered acceptable even with a consistent margin of error. Having said this, the explanation of the steps leading to the completion of BERT's fine-tuning follows.

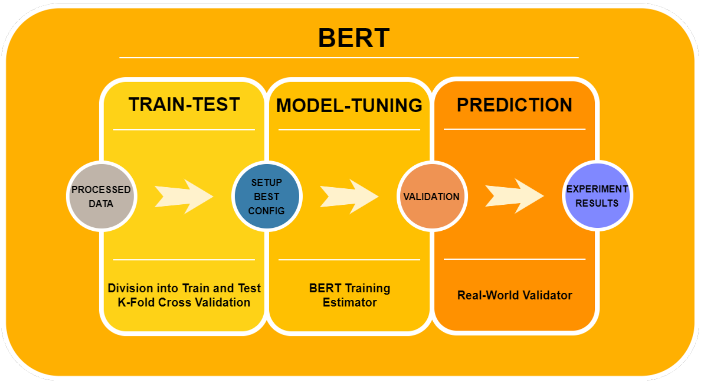


Figure 4.6 BERT Training Phase Workflow

As can be seen from Figure 4.6, in order to obtain a valid predictive model, the following logical steps have been followed:

- **Train and Test Division**: this sub-phase receives the processed data from the previous phase and subdivides them into train and test set according to the K-Fold Cross Validation paradigm. This results in K sub-iterations for each iteration, each time with different trains and tests. In this way, more realistic results are obtained.
- **Model-Tuning**: this is the crucial step in all the proposed work. The BERT model is fine-tuned, according to the best configuration found after the Hyper-Parameters Fine-Tuning. The output of this phase is determined by the BERT estimator, which computes important variables such as Accuracy, F1-Score, Precision and others.
- **Prediction**: once the BERT model is fine-tuned, the other crucial step of the work carried out, i.e. a real-world validation, begins. This phase consists in predicting the beginning of a new segment of text using as input an entire text, which has obviously followed all the steps of Data Preparation previously described. As output is obtained a list of values reflecting the belonging or not to the previous text segment.

4.4.5 Results Analysis

This phase shows how the results predicted by the fine-tuned BERT model are processed. As Figure 4.7 shows, this phase gets the results of the predictions in input and aggregates them, mediating the results by experiment and by sub-experiment. This way they get aggregated results that can be viewed to get a clear view of how the experiments performed behaved. Only thanks to this phase it is possible to draw conclusions on how the evolution of the experiments has improved or worsened the fine-tuning of the BERT model.

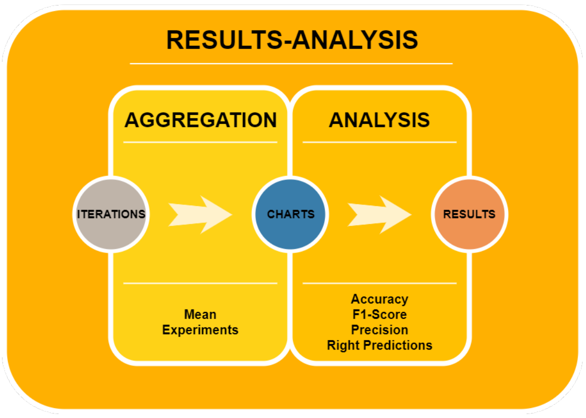


Figure 4.7 Results Analysis Phase Workflow

Chapter 5

Implementation

In this chapter, it is illustrated in detail the entire development and implementation, going through all the stages:

1. BERT Results Demonstration
2. Data Quantity Evaluation
3. Hyper-Parameters Fine-Tuning
4. Data Collection and Exploration
5. Pre-Processing
6. BERT

In the end, the libraries used in the implementation are shortly presented.

5.1 BERT Results Demonstration

The starting point of the work carried out was to repropose the official results obtained by the creators of BERT. In order to do so, it was decided to reproduce the tasks known as MRPC (Microsoft Research Paraphrase Corpus) among those proposed. MRPC refers to a *Sentence (and sentence-pair) classification tasks*. This task was replicated using the BERT pre-trained model version named uncased_L−12_H−768_A−12. This model offers characteristics as follows:

- Layers: 12, 768 Hidden
- Heads: 12

- Parameters: 110M

There are several versions of BERT pre-trained model, ranging from cased and uncased versions, base and large versions and multi-lingual and language-specific versions. Currently, it is not possible to reproduce BERT−Large with a GPU of 12GB-16GB of RAM, because the maximum batch−size that can fit in memory is too small. The results proposed in the paper of BERT [20] are trained on a single Cloud TPU which has 64GB of RAM. This limitation affected also the choice of the model used in this thesis. The technology used to reproduce all the proposed work is Google Colaboratory, that offers (at time of writing) free GPU with the following characteristics:

- GPU: 1xTesla K80, with 2496 CUDA cores
- RAM: 12GB
- Disk: 320GB

Given these hardware limitations all the BERT−Large pre-trained models can't be used, hence, the model used in all the experiments was the one proposed above, unless different specifications.

BERT pre-trained model version used in this demonstration corresponds to the BERT−Base model version and in this example, it was fine-tuned with GLUE Data, as the official implementation does. Since the data contained in GLUE MRPC are only 3600, it is possible to fine-tune BERT pre-trained model in minutes. This task was chosen to be reproduced for this very reason: given the limited amount of data used to fine-tune the pre-trained BERT model, it is possible to reproduce accurate results even with one of the smaller capacity hardware. In Listing 5.1 and 5.2, it is possible to see that the results obtained by this thesis reflect the official results proposed by the creators of BERT.

```
***** Eval results *****
  eval_accuracy = 0.845588
  eval_loss = 0.505248
  global_step = 343
  loss = 0.505248
```

Listing 5.1 Official Results

```
***** Eval results *****
  eval_accuracy = 0.8455882
  eval_loss = 0.4693081
  global_step = 343
  loss = 0.4693081
```

Listing 5.2 Reproduced Results

5.2 Data Quantity Evaluation

The purpose of this section is to identify the optimal amount of data to perform the subsequent steps of the proposed work. It is important to stress the importance of this step, as it allows to observe how the accuracy of the pre-trained model evolves based on the time spent in fine-tuning, thus enabling to choose the best amount of data to use for subsequent experiments to have an average accuracy value optimal. Hyper-parameters used for these evaluations are the default ones, proposed in the official implementation and defined as follows:

- Batch-size: 32
- Learning Rate: 2e-05
- Train Epochs: 3
- Warm-up Proportion: 0.1
- Max Sequence Length: 128

These variables will be commented and explained in detail below in 5.3. Each amount of data has been executed on BERT's model a total of *5* times, to obtain a more precise estimate. At each iteration, the model is fine-tuned from scratch, in order to avoid the use of a more precise model. It was decided to store BERT's model estimation in JavaScript Object Notation (JSON). This notation has been used to store variables in key-value pairs, in order to describe each iteration, with named attributes and associated value. An example of JSON file used to store BERT's evaluations is presented in Listing 5.3, representing a nested structure in which the main item is named with the data quantity used, followed by variables inside it.

```json
{
    "10000": {
        "iteration": 2.0,
        "bert_accuracy": 0.8287000060081482,
        "bert_precision": 0.8294247388839722,
        "bert_f1_score": 0.8285113573074341,
        "bert_recall": 0.8276000022888184,
        "bert_auc": 0.828700065612793,
        "bert_loss": 0.7328279614448547,
        "bert_tp": 4138.0,
        "bert_tn": 4149.0,
        "bert_fp": 851.0,
        "bert_fn": 862.0,
        "time": 1803.0
    }
}
```

Listing 5.3 Data Quantity JSON Example

How BERT Training was performed will be explained in Section 5.6.

The discriminating variable of choosing the best amount of data was the Accuracy of BERT.

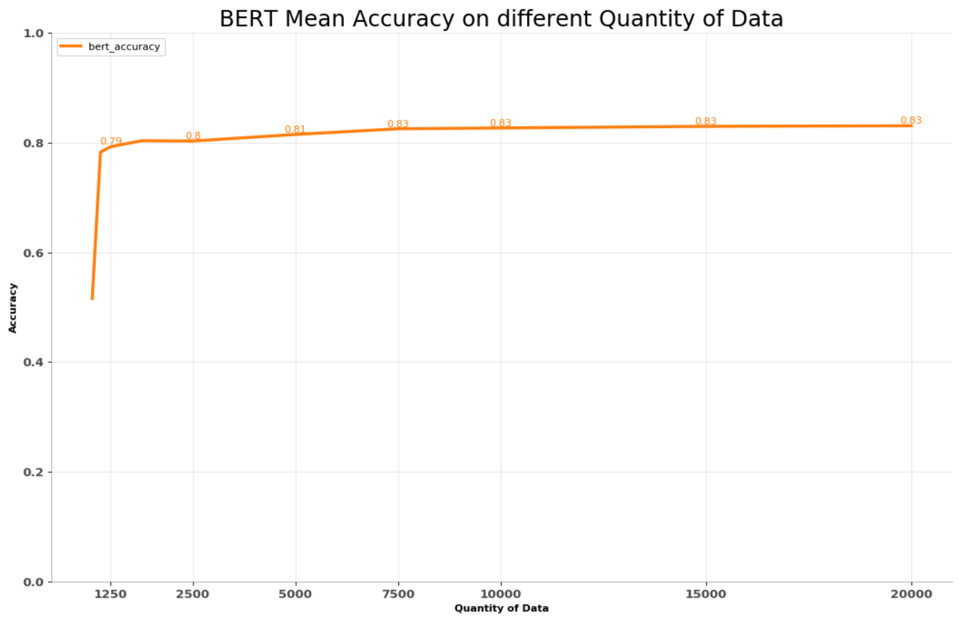


Figure 5.1 BERT Accuracy with Different Amount of Data

Different amount of data have been tested, ranging from very little quantity to higher ones, for a total of 10 different data quantities: *50, 250, 500, 1250, 2500, 5000, 7500, 10000, 15000, 20000*. The whole procedure took approximately 15 hours of training. As can be seen from Figure 5.1, the accuracy computed by BERT drastically decreases using a smaller amount of data than *5000*. For amounts of data greater than *7500*, BERT Accuracy value remains stable at around 83%, while fine-tuning *Time* increases as it can be seen below on Figure 5.2.

In addition, other variables were considered in the choice of the amount of data to be used along with the subsequent steps of the work done, such as:

- *Time*: this variable has been evaluated to find the best trade-off between *Time* and the resulting *BERT Accuracy*. Since each experiment is performed multiple times, a very high execution time would have involved unmanageable times. As it can be seen in Figure 5.2, the *Time* variable grows linearly as the training data grow, leading to an unmanageable time of execution.

- *F1-Score*: in order to realize a real-world validation, this variable includes two other fundamental variables, namely precision and recall. This variable is fundamental since it reflects the performance of the model against the Validator. It is possible to see the evolution of the F1-Score estimated by BERT in Figure 5.3. The trend shown in this Figure is similar to the one of Figure 5.1, so it reflects the Accuracy of the model.

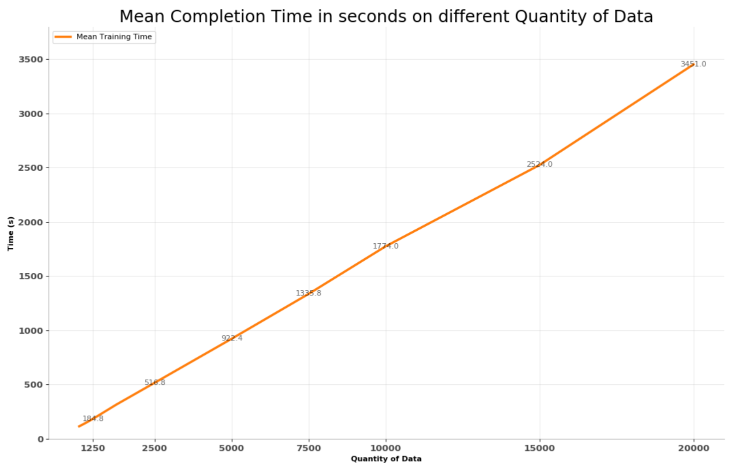


Figure 5.2 Training Time with Different Amount of Data

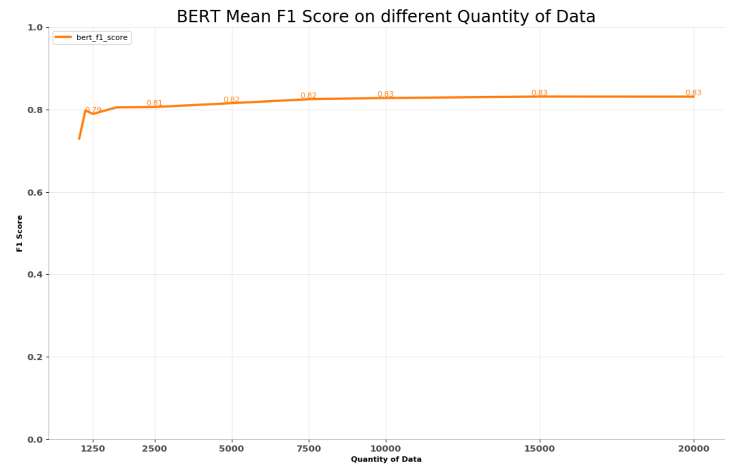


Figure 5.3 F1-Score with Different Amount of Data

To conclude this phase, as optimal amount of data, it was chosen to train BERT's model with *7500* data, resulting in an acceptable fine-tuning time and BERT accuracy.

5.3 Hyper-Parameters Fine-Tuning

This section should be viewed as completely from the previous section 5.2, as together with the data quantity analysis, it allows to obtain the best setup for the proposed model. By evaluating different hyper-parameter configurations it is possible to establish which values result in a more precise model and which, instead, worsen the performance.

Hyper-Parameters that can be manipulated on the pre-trained BERT model are shown in Figure 5.4:

```
batch_size
max_seq_length
epochs
learning_rate
warmup_proportion
```

Figure 5.4 BERT Hyper-Parameters

These hyper-parameters can take real values, so it was impossible to test all the possible configurations that these hyper-parameters could take. Consequently it was decided to vary the hyper-parameters as shown in Table 5.1, resulting in 150 different configurations, all of them repeated 5 times.

Variable	Value1	Value2	Value3	Value4	Value5	Value6
batch_size	**32**					
max_seq_length	**128**					
epochs	1	**3**	5	7	9	
learning_rate	2e-06	**2e-05**	5e-05	7e-05	2e-04	2e-05
warmup_proportion	0.001	0.01	0.05	**0.1**	0.2	

Table 5.1 Hyper-Parameters Values Estimated

It was not possible to vary two hyper-parameters, batch_size and max_seq_length, since, as previously mentioned, the value of these two parameters depends on the hardware characteristics of the machine used, i.e. a different value of these two hyper-parameters involves the exhaustion of machine resources. Indeed, some tests were done with a batch_size equal to 16, resulting in the same averaged value of accuracy but with a doubled time of fine-tuning, hence to avoid time consumption, this value was removed from the configurations. In fact, the time spent in this procedure is even greater than the one of the analysis on the amount of data, being 15 days, 23 hours and 15 minutes.

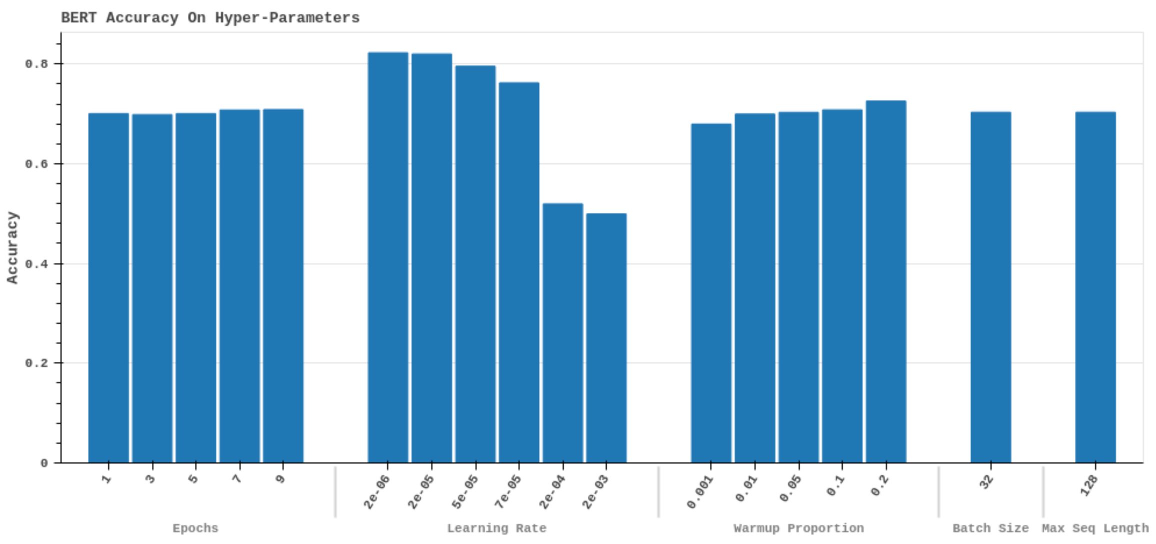


Figure 5.5 Hyper-Parameters and BERT Accuracy

Figure 5.5 shows averaged BERT accuracy for all the values referenced in Table 5.1. This can give an idea on which configuration could be the best, but an important variable not considered is fine-tuning Time, as stressed above. This variable has been considered in Figure 5.6, that displays all the configurations and respective fine-tuning time and accuracy.

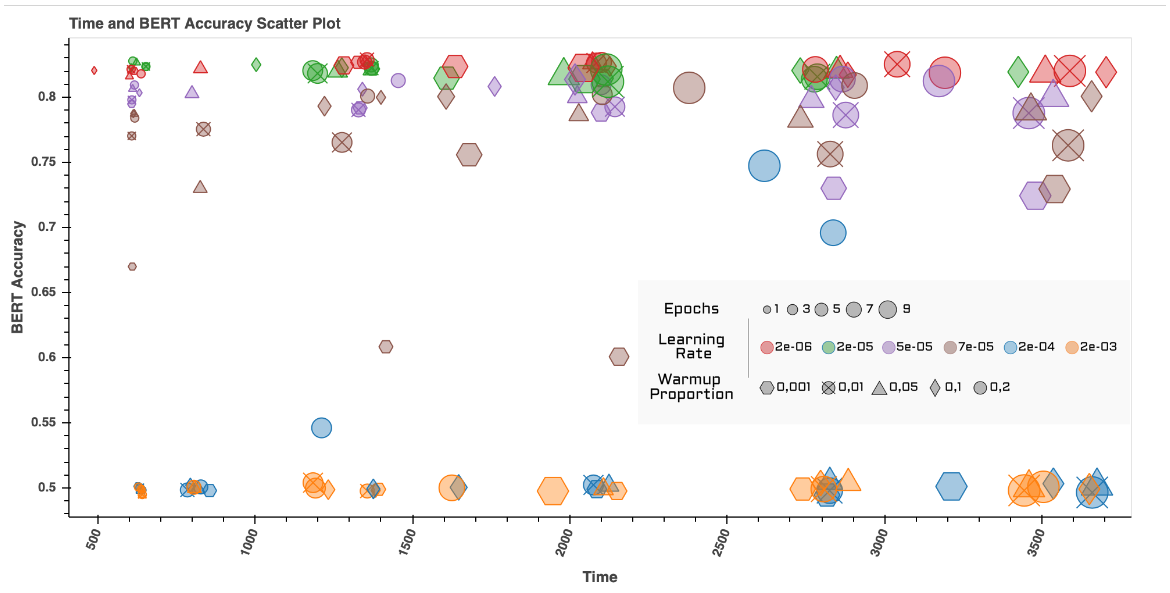


Figure 5.6 Hyper-Parameters

As it can be seen from Figure 5.6, epochs growing affects fine-tuning time, as expected, while accuracy isn't growing.

Another thing that can be inferred is that for smaller values of epochs there is not a drastic drop in accuracy, but rather it can be seen that the major discriminant is the learning rate. In fact, higher values of learning_rate correspond to a very low accuracy, both for a high and low epoch value.

In order to choose the best configuration that will be the basis for all the following work, it was decided to use BERT Accuracy as the discriminatory variable. This is not the unique solution, in fact, another analysis was carried out using a ranking to evaluate which was the best configuration to use. Several ranking strategies have been proposed, all based on vector similarity.

$$Ref\_Vector(Time, BERTAccuracy, BERTF1Score) = [0,1,1]$$

A Reference vector of the form above has been built and for each configuration has been computed a vector of the following form:

$$\begin{aligned} &Configuration\_Vector(Time, BERTAccuracy, BERTF1Score) = \\ &\quad [\frac{1}{4} * \frac{Time - min(Time)}{max(Time)}, \frac{1}{2} * BERTAccuracy, \frac{1}{2} * BERTF1Score] \end{aligned} \tag{5.1}$$

For each configuration has been compared the distance against Reference_Vector. As it can be seen from formulas, each vector is composed of 3 different variables, such as Time, Accuracy and F1-Score, for the configurations this vector is adjusted by weights. Different similarity functions have been used to generate a ranking, as in Table 5.2.

configuration	euclidian distance	cosine similarity	manhattan distance	minkowski distance	final rank
32.0_3.0_2e-06_128.0_0.05	0	0	6	0	1
32.0_1.0_2e-06_128.0_0.001	1	1	0	2	2
32.0_1.0_2e-06_128.0_0.05	5	2	5	7	3
32.0_3.0_2e-06_128.0_0.1	2	3	11	1	4
32.0_1.0_2e-06_128.0_0.2	11	4	7	13	5
32.0_1.0_2e-06_128.0_0.01	6	5	3	8	6
32.0_3.0_2e-06_128.0_0.2	3	6	12	4	7
32.0_1.0_2e-05_128.0_0.001	7	7	4	10	8
32.0_3.0_2e-06_128.0_0.01	4	8	13	6	9

Table 5.2 Ranking Based on Time, BERT Accuracy and BERT F1-Score

The best configuration with this ranking strategy is the one with the attributes shown in Listing 5.4.

```
batch_size = 32
max_seq_length = 128
epochs = 3
learning_rate = 2e-06
warmup_proportion = 0.05
```

```
BERT Accuracy = 0.821840
BERT F1-Score = 0.825373
Time = 826.4 (s)
```

Listing 5.4 Ranking Configuration

However, it was decided to use only BERT Accuracy to be the discriminator variable because the ranking was biased by the choice of the weights. Doing so resulted in a more impersonal choice and a different best configuration. Ranking only with Accuracy of BERT, the best resulting configuration proved to be that in Listing 5.5, that shows a greater fine-tuning Time with respect to the previous configuration, however acceptable.

```
batch_size = 32
max_seq_length = 128
epochs = 3
learning_rate = 2e-06
warmup_proportion = 0.01
```

```
BERT Accuracy = 0.828800
BERT F1-Score = 0.831677
Time = 1355.4 (s)
```

Listing 5.5 Best Configuration

This configuration of Hyper-Parameters will be used for all the following experiments presented in this thesis.

5.4 Data Collection and Exploration

After having defined the basis for the following phases, the next step is to select data to build ad-hoc scenarios for a Text Segmentation task. It was decided to choose two different sources of data: Books and News Articles.

5.4.1 News Articles

The first scenario that this work will analyze exploits *News Articles* coming from Harvard Dataverse[1] and, in particular, the dataset of news articles used can be downloaded manually from a persistent link[2]. The manual download presents some difficulties, because of the huge dimension of the file. To overcome these problems, they can be downloaded in seconds using the Google Colaboratory GPU.

[1]https://dataverse.harvard.edu

[2]https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/5XRZLH

However, the JSON objects have more information fields than required for the thesis purpose, such as the article's source name, keywords, publish date and many others. Since this thesis is not interested in these other JSON fields, they will be removed during Section 5.5.

5.4.2 Books

There aren't many sources of digitized books as one can think. Among the most complete collection of books, there are Project Gutenberg[3], with books in plain text or in electronic publication (epub) format, the Internet Archive Books[4] and Oxford Text Archive[5].

Since books in epub format have no standard structure, being written in HyperText Markup Language (HTML), it was decided to use plain text books from the above listed sources.

Using plain text books introduce a considerable complexity because this format is not structured. In this context, the goal of this thesis is to find when a new Text Segment is introduced, so, in order to have a reference, it was decided to understand when a new chapter begins, having as input a list of sentences.

5.5 Pre-Processing

The objective of this section is to present how data downloaded in Section 5.4 are transformed into a format suitable to fine-tune BERT pre-trained model.

This section is divided accordingly to the two sources of data, *Books* and *News Articles*, since, being different data structures, this phase assumes different characteristics.

5.5.1 News Articles

This dataset, given its large size, is downloaded directly into the machine offered by Google Colaboratory and is manipulated here in order to obtain a form that can be easily managed. The starting format is very easy to handle: the JSON format allows easy access to data, allowing easy manipulation.

Each JSON object is defined with a long list of fields, that are not object of interest for the work proposed in this thesis. In order to lighten these objects, many fields are discharged.

[3]http://www.gutenberg.org
[4]https://archive.org/details/internetarchivebooks
[5]http://ota.ox.ac.uk/catalogue/index.html

The following snippet presents the starting JSON object and how it is lightened by keeping only necessary fields, highlighted with bold formatting.

```
{
 {'_id': '2d61667465722d706f6c6963652',
  'authors': ['Example Author'],
  'category': 'world',
  'category_aggregate': 'world',
  'category_probability': 0.81751,
  'ground_truth': False,
  'keywords': ['keyword1', 'keyword2'],
  'language': 'en',
  'pipelined': False,
  'publish_date': '2018-08-28',
  'scrape_date': '2018-09-08',
  'source': 'https://www.example.com',
  'source_name': 'Example',
  'tags': [''],
  'text': 'example text',
  'title': 'Example Title',
  'url': 'https://www.example.com/example_link'},
}
```

Listing 5.6 News Article Starting JSON

After this field cleaning the resulting JSON is composed of only three fields, in particular, the field named `text` contains the whole article. In order to allow the creation of a dataset suitable for the targeted Text Segmentation task, it is necessary to split the text of these articles into lines, since the purpose is to understand whether a sentence belongs to a new or the same News Article.

As mentioned above, the following phases include the needed pre-processing steps to obtain an appropriate structure to build the dataset.

- Sentence Split: for each article, with the help of a regular expression, punctuation is identified and sentences are divided accordingly.
- Identifier Assignment: finally, the id of the article is substituted with an easier to read ID, created by the concatenation of characters and values, in order to ease the following manipulation.

Resulting article structure is shown in Table 5.3. This structure is saved as a DataFrame object using Pandas library.

article_id	article_num	sentence_num	category	text
a9s3	9	3	sport	Sentence3
a9s4	9	4	sport	Sentence4

Table 5.3 News Articles Structure After Pre-Processing

5.5.2 Books

Books, that have been downloaded as plain text from different sources in Section 5.4, are compressed into a unique archive. Then, in order to create a unique document, all these books are concatenated into one single structure, giving an identifier to each book.

In this way, a list of books is stored, ready to be further processed. The objective in this scenario is to identify whether a text segment belongs to a new or the same chapter in the book. In order to do this identification, it is necessary to split the books into chapters.

The Data Cleaning Pipeline consists of three main phases, below described.

- Sentence Split: for each book present in the raw list of books, with the help of a regular expression, punctuation is identified and sentences are divided accordingly.
- Chapter Division: then each book consisting of a list of sentences, is scanned to find where a chapter begins, using another regular expression.
- Identifier Assignment: finally, an identifier is assigned to each sentence, in order to ease the following manipulation. This identifier is created by concatenating the book to which the sentence belongs, the chapter to which the sentence belongs and the number in order of succession of the sentence.

Table 5.4 shows the resulting structure after the pre-processing of the book scenario. This structure is saved as a DataFrame object using Pandas library.

book	chapter	sentence_num	sentence_id	sentence
23	4	5	b23c4s5	Sentence5
23	4	6	b23c4s6	Sentence6
23	4	7	b23c4s7	Sentence7

Table 5.4 Resulting Book Structure

It was decided to not go deeper with further text mining phases since BERT itself in the fine-tuning performs all the needed tasks, like tokenization and lemmatization.

5.6 BERT

In order to reproduce a Text Segmentation task using BERT pre-trained model, the dataset used must be structured in a form that BERT can understand. The general dataset structure suitable for BERT is presented in Table 5.5.

id1	id2	sentence1	sentence2	new_segment
b135c13s1038	b210c25s513	example Text1	example Text2	1
b267c9s108	b267c9s117	example Text1	example Text2	0

Table 5.5 General Structure of the Dataset

How the scenario-specific dataset is made will be exhaustively explained in the following Chapter.

After this assumption, the first following steps is to download the dataset, previously stored on the well-known platform BitBucket [6]. Along with the dataset used to fine-tune BERT pre-trained model, it is also downloaded a file used for real-world validation, that consists of a full book or list of news articles, depending on the scenario.

Configuration When both the dataset and the validator file have been loaded, the subsequent step is to set up the best configuration obtained from Section 5.3 and to set the number of folds to be used for the K-Fold Cross Validation. Another variable to set is the pre-trained model to be used. Unless specified differently in the single experiment, the model used is the one mentioned above, BERT model uncased_L−12_H−768_A−12.

Training The library used to handle all the model lifecycle is bert library from tensorflow and tensorflow_hub, where the model is stored. With this library it is possible to call functions that format the dataset couples of sentences into an input example that BERT can understand, that call the tokenizer from the module_hub, that convert these input examples into features, that create the model and the estimator and, finally, that fine-tune the model with an additional training (Listing 5.7).

```
train_InputExamples = train.apply(
                lambda x: bert.run_classifier.InputExample(guid = None,
                            text_a = x[DATA_COLUMN_1], text_b = x[DATA_COLUMN_2],
                            label = x[LABEL_COLUMN]), axis = 1)
tokenizer = create_tokenizer_from_hub_module()
```

Listing 5.7 Extract of BERT Functions

[6]https://bitbucket.org

5.7 Validation and Result Analysis

The last phase is a real-world validation with a validator file, whose sentences are processed in subsequent couples to predict whether or not they belong to the same segment. These predictions are then evaluated and variable such as *Precision*, *Accuracy* and others are stored in JSON format for each iteration. In Listing 5.8 it is possible to see all the variable stored at each iteration.

The object name corresponds to the quantity of data used. Different quantities, based on the experiment, are used in K-Fold Cross Validation and refers to the total quantity of data. The amount of data used in training is the 80% of that value.

```
{
    "9375": {
        "iteration": 3.0,
        "bert_accuracy": 0.8138666749000549,
        "bert_precision": 0.7908163070678711,
        "bert_f1_score": 0.8162190318107605,
        "bert_recall": 0.8433079719543457,
        "bert_auc": 0.8144364356994629,
        "bert_loss": 0.4472725987434387,
        "bert_tp": 775.0,
        "bert_tn": 751.0,
        "bert_fp": 205.0,
        "bert_fn": 144.0,
        "prediction_right": 16.0,
        "prediction_total": 45.0,
        "prediction_threshold_1": 16.0,
        "prediction_threshold_3": 18.0,
        "prediction_threshold_6": 19.0,
        "val_accuracy": 0.9147286821705426,
        "val_precision": 0.35555555555555557,
        "val_f1_score": 0.4923076923076923,
        "val_recall": 0.8,
        "val_tp": 16.0,
        "val_tn": 338.0,
        "val_fp": 29.0,
        "val_fn": 4.0,
        "time": 1271.0
    }
}
```

Listing 5.8 Example Of Validator Results

To evaluate the performance, different variables have been used, like Precision, Recall, F1−Score and Accuracy.

Variables named as prediction_threshold_x correspond to the number of correct predictions in the x-neighbourhood of the real new segment.

Other variables computed are the following:

$$Precision = \frac{TP}{TP+FP} \qquad Recall = \frac{TP}{TP+FN}$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+TN} \qquad F1-Score = \frac{2*Precision*Recall}{Precision+Recall}$$

Where:

TP (true positive): segments predicted to belong to a new segment and that belong to a new segment.

FP (false positive): segments predicted as new segments that are not new segments.

FN (false negative): segments predicted as current segments that are new segments.

TN (true negative): segments predicted to belong to the current segment and that belong to the current segment.

These variables are very helpful in the evaluations of the performances obtained by each experiment and allow to make considerations about it.

5.8 Libraries

5.8.1 Tensorflow

Tensorflow [7] is the core open source library to help developing and training ML models.

5.8.2 Scipy

Scipy is a Python-based ecosystem of open-source software for mathematics, science, and engineering. Among its core packages we used:

- NumPy [8], the fundamental package for scientific computing with Python.
- Matplotlib [9], a Python 2D plotting library.
- Pandas [10], which provides easy-to-use data structures, the pandas dataframe, that are widely used.

[7]https://www.tensorflow.org
[8]http://www.numpy.org
[9]https://matplotlib.org/index.html
[10]http://pandas.pydata.org/index.html

5.8.3 Scikit-learn

Scikit-learn [11] is an open-source library that provides useful tools for data mining and machine learning tasks.

5.8.4 Bokeh

Bokeh [12] is an interactive visualization library that targets modern web browsers for presentation. Its goal is to provide elegant, concise construction of versatile graphics, and to extend this capability with high-performance interactivity over very large or streaming datasets. Bokeh can help anyone who would like to quickly and easily create interactive plots, dashboards, and data applications.

[11]https://scikit-learn.org/stable/
[12]https://docs.bokeh.org/en/latest/

Chapter 6

Experiments and results

The goal of this chapter is to show how the methodology and the implementation, discussed in the previous sections, have been put to the test in different experiments. Firstly, the used dataset is presented and analyzed. Then, we focus on the variations of the proposed approach, where several combinations of methods and different parameters are explored. Moreover, the results are discussed and compared, in order to understand which is the best solution to achieve the work objective.

6.1 Dataset

The dataset used is created by coupling sentences that are previously manipulated in order to obtain a suitable format for fine-tuning BERT pre-trained model. These couples of sentences are stored in a `tar.gz` archive, as suggested by the official implementation, uploaded on BitBucket to ease the download from external URL. Same results could be obtained by uploading directly the dataset into Google Colaboratory manually for each execution.

6.1.1 Sentences Coupling

In this context, the creation of the dataset takes place by coupling sentences coming from the same or different text segment. In this way the size of the dataset can be decided a priori, generating the pairs of established sentences.

It was decided to use different amounts of data in each of the experiments proposed below, in order to obtain a more complete view of each possible configuration used. The amounts of data used include the best amount of data obtained from the previous step, plus additional amounts of data are evaluated, all using K-Fold Cross Validation as a re-sampling procedure.

Under this assumption, sentence pairs are produced by coupling a number of sentences from the same text segment, together with the same number of pairs from a different segment. In this way, a data structure is obtained consisting of pairs of the same segment to which the new_segment label with value 0 is assigned, while for pairs coming from different text segments, new_segment with value 1 will be assigned.

The resulting dataset has the form described in Table 5.5.

In order to ease the computation, only the identifier of the sentence is used as a discriminator to check whether or not two sentences belong to the same segment, adding the full text of the sentence afterwards.

Hence, in order to generate X sentences pairs, $X/2$ pairs belonging to the same segment are appended to $X/2$ pairs belonging to different segments. The dataset undergoes a shuffling procedure and then a sampling based on the amount of data to be generated.

In this way, the archive is composed of as many folders as the amount of data to be used. The origin of this operation comes from the fact that in this way the same data are always used analyzing the same amount of data.

The dataset is then customized based on each experiment needs. The appropriate modifications, dictated by the characteristics of the experiment, are explained together with it.

6.1.2 Validation

The real-world validation, as explained in 5.7, is constructed by extrapolating a portion of data and saving it in Comma Separated Values (CSV) format consisting of a list of lines and new_segment values that are equal to 1 when a new segment starts. The purpose of this file is to provide a real-world use case to the fine-tuned model and to understand if the estimations on the model reflect reality.

The validator file is declined differently according to the scenario:

- News Articles: the validator file is composed of 20 different articles concatenated subsequently, for a total of about 400 sentences. These articles can be already used in training or not, depending on the experiment.
- Books: the validator file is composed of a variable amount of chapters, coming from different books and manipulated differently based on the experiment needs. The number of these chapters is always between 10 and 20 with a variable number of sentences in the order of magnitude of the thousand. This characteristic comes from the nature of the text segment used since chapters are longer than news articles.

6.2 News Articles Experiments

This section shows every experiment about the News Articles Scenario. All experiments have been named as ArtX, where *X* is the identifier of the experiment, a scalar number in ascending order.

Each experiment of this scenario was executed with a K-Fold Cross Validation *K* equal to 5 and for different quantities of data, such 4000, 9375, 15000, 30000. These quantities of data have been chosen under the considerations of Section 5.2 and are divided as 80% for the training phase and 20% for the test phase. The reference Data Quantity is 9375 since it results in 7500 data for the training phase.

In the following subsections, all the experiments of this scenario are exhaustively explained. Moreover, at the end of this section, a summary, that includes all the experiments, is commented.

6.2.1 Experiment Art1

This experiment is the least elaborate one. The dataset used is the one explained in Section 6.1 with no further additions, news articles are divided into lines and the sentences are coupled as they are. The dataset is built using a random sampling on 60000 couples, hence the percentage of couples that belong to the same article will be probably different from couples that belong to the different articles.

In this experiment, the BERT pre-trained model is fine-tuned with different quantity of data, including four iterations with 50000 couples (40000 in training and 10000 in testing). Results of this experiment can be evaluated in Table 6.1.

The validator file used is composed of 20 different articles, hence predictions right can be at most 20. This particular file is long exactly 428 lines.

Data	BERT Accuracy	BERT F1-Score	Right Predictions	Total Predictions	Validator Accuracy	Validator Precision	Validator F1-Score	Validator Recall
4000	0.83300	0.83966	15.2	61.8	0.8801864	0.2478313	0.373339	0.76
9375	0.82656	0.83223	15.4	54.0	0.8993006	0.2866983	0.417374	0.77
15000	0.82740	0.83056	15.0	51.4	0.9034965	0.2916866	0.419889	0.75
30000	0.83566	0.83637	14.6	52.8	0.8983682	0.2772215	0.401644	0.73
50000	0.84182	0.84360	14.25	49.75	0.9038461	0.2866394	0.408685	0.7125

Table 6.1 Results of Experiment Art1

All of these values are averaged in a total of 5 iterations (except for the data quantity 5000).

This experiment shows how the pre-trained BERT model behaves against a real-world validation. Besides, BERT Accuracy, computed with its estimator, shows a quite high value, even though real-world validation presents a high false positives value. The parameter Validator Accuracy shows a very high value: this is due to all the correct predictions for couples belonging to the same article (so the same text segment).

6.2.2 Experiment Art2 - Balanced

This experiment is characterized by a different composition of the dataset used in Experiment 6.2.1. This dataset is built by the same number of same article couples and different articles couples. From now on, to ease the reading, this type of dataset composition will be called ***Balanced***.

Data	BERT Accuracy	BERT F1-Score	Right Predictions	Total Predictions	Validator Accuracy	Validator Precision	Validator F1-Score	Validator Recall
4000	0.83725	0.83951	14.6	54.8	0.8937063	0.2671505	0.390954	0.73
9375	0.82176	0.82351	14.6	49.2	0.9067599	0.2970651	0.422207	0.73
15000	0.83113	0.83256	14.2	51.8	0.8988344	0.2752260	0.396322	0.71
30000	0.83646	0.83813	15.4	50.8	0.9067599	0.3033113	0.434950	0.77

Table 6.2 Results of Experiment Art2

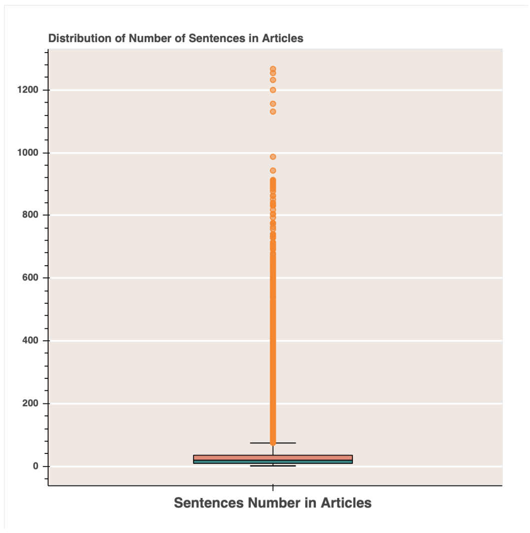
Results in Table 6.2 highlights that the balancing has no substantial effect on the model performance, although a slight improvement in all the variables is present.

Although the improvement is slight, the datasets of the next experiments will be constructed according to a 50%-50% balance (unless specified differently in the experiment description).

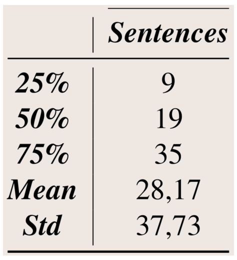
6.2.3 Experiment Art3 - Distribution-Wise

The following experiment's dataset is built using balanced data and under additional pre-processing. Before creating the sentence pairs, an analysis of the distribution of the lengths of all the articles was conducted. Figure 6.1a shows a box plot that describes the length of the articles in terms of the number of sentences. This box plot was used to reduce the number of items that were used as a pool of sentences from which to extract the pairs, in particular, all articles with fewer sentences than the value of the first quartile are removed.

To refer easily to this particular manipulation will be used the name ***Distribution-Wise***. Adding this particular processing results in a slight drop in performance as it can be seen in Table 6.3. This article has been run also with 50000 data.



(a) Box Plot



	Sentences
25%	9
50%	19
75%	35
Mean	28,17
Std	37,73

(b) Box Plot values

Figure 6.1 Number of Sentences Distribution in Articles

Data Quantity	BERT Accuracy	BERT F1-Score	Right Predictions	Total Predictions	Validator Accuracy	Validator Precision	Validator F1-Score	Validator Recall
4000	0.80575	0.80841	14.0	56.2	0.88764	0.24931	0.36743	0.7
9375	0.81952	0.82250	14.4	51.2	0.90116	0.28119	0.40439	0.72
15000	0.82213	0.82554	14.6	51.6	0.90116	0.28311	0.40786	0.73
30000	0.82290	0.82502	15.2	52.2	0.90256	0.29151	0.42130	0.76
50000	0.82580	0.82803	15.0	54.0	0.89743	0.27777	0.40540	0.75

Table 6.3 Results of Experiment Art3

6.2.4 Experiment Art4, Art5 and Art6 - Mono Category

The experiments that are described in this section share the same characteristics of the previous example, except for the use of the data quantity 50000, hence they are presented together.

Their peculiarity is the limitation of the pool of articles to the ones characterized by a specific category: an attribute of the News Articles full dataset that was not removed.

- Experiment Art4: characterized by articles of the category *Sports*
- Experiment Art5: characterized by articles of the category *Health*
- Experiment Art6: characterized by articles of the category *World*

All these experiments have been validated on a validator made by articles of the same category.

Experiment	Data Quantity	BERT Accuracy	BERT F1-Score	Right Predictions	Total Predictions	Validator Accuracy	Validator Precision	Validator F1-Score	Validator Recall
4	4000	0.75525	0.76360	17.6	82.8	0.85678	0.21499	0.34486	0.88
4	9375	0.75296	0.75593	17.0	81.2	0.85762	0.20987	0.33652	0.85
4	15000	0.77173	0.77471	16.6	83.0	0.85212	0.20009	0.32243	0.83
4	30000	0.77276	0.77339	17.0	81.6	0.85678	0.20856	0.33490	0.85
5	4000	0.71625	0.77836	18.2	157.4	0.63566	0.15477	0.25970	0.91
5	9375	0.77248	0.80708	18.0	106.0	0.76744	0.17321	0.28974	0.9
5	15000	0.79040	0.81824	17.8	89.8	0.80826	0.19836	0.32438	0.89
5	30000	0.79466	0.82244	17.8	93.4	0.79896	0.19090	0.31425	0.89
6	4000	0.81575	0.81862	17.2	47.0	0.91117	0.36730	0.51445	0.86
6	9375	0.81749	0.82026	17.8	42.2	0.92752	0.42253	0.57293	0.89
6	15000	0.82187	0.82381	17.8	37.8	0.93951	0.47110	0.61599	0.89
6	30000	0.82683	0.82771	18.2	41.2	0.93242	0.44325	0.59573	0.91

Table 6.4 Results of Experiments Art4, Art5 and Art6

Table 6.4 shows that some categories perform better than others. The number of *Total Predictions* is extremely high in Experiments Art4 and Art5, while Experiment Art6 has very good results in all of its parameters. These three experiments results should be worse than other experiments, due to the difficulty of the task: predicting the start of a new article of the same topic is more difficult than predicting one of different topics. Under this consideration, Experiment Art6 is characterized by a more general category of articles with respect to the other two experiments, labelled as *world*, hence with a more general dataset performs better than the others.

6.2.5 Experiment Art7 - Validation with New Data

This section describes experiment Art7, that is characterized by an additional feature: articles are divided on top of the dataset processing to be part of the dataset that fine-tunes the model and of the validator file. Previously the articles that compose the validator file could have been also part of the dataset. With this additional feature, it's sure that the validator file consists of data never used for the fine-tuning process.

Data Quantity	BERT Accuracy	BERT F1-Score	Right Predictions	Total Predictions	Validator Accuracy	Validator Precision	Validator F1-Score	Validator Recall
4000	0.75950	0.76496	15.4	54.6	0.88412	0.28268	0.41342	0.77
9375	0.78283	0.78913	15.4	56.6	0.87883	0.27220	0.402120	0.77
15000	0.78627	0.79117	15.6	52.4	0.89100	0.29899	0.43194	0.78
30000	0.79253	0.79648	15.4	48.8	0.89947	0.31602	0.44798	0.77

Table 6.5 Results of Experiment Art7

Table 6.5 shows a consistent drop in performance for the model, with lower *BERT Accuracy* and worse predictions on the validator. BERT Accuracy is lower than expected, but this is due to different sentences used for fine-tuning the model.

6.2.6 Experiment Art8, Art9, Art10 and Art11 - Different Balancing

These experiments share the same configuration of Section 6.2.5 but with a different balance between sentences from the same article and from different articles. The percentages used are presented in Table 6.6.

Experiment	From the Same Article	From Different Articles
Art8	80	20
Art9	60	40
Art10	40	60
Art11	20	80

Table 6.6 Percentages of Experiments Art8, Art9, Art10 and Art11

Using more sentences from the same article, *Total Predictions* are expected to be lower than the opposite, and vice versa.

This consideration is confirmed in Table 6.7, were results of these experiments are shown.

Experiment	Data Quantity	BERT Accuracy	BERT F1-Score	Right Predictions	Total Predictions	Validator Accuracy	Validator Precision	Validator F1-Score	Validator Recall
8	4000	0.80325	0.80806	15.8	47.6	0.90697	0.33261	0.46754	0.79
8	9375	0.81408	0.81746	15.2	45.8	0.90853	0.33216	0.46222	0.76
8	15000	0.81267	0.81612	16.0	44.0	0.91731	0.36416	0.50030	0.8
8	30000	0.82010	0.82171	15.0	37.8	0.92816	0.39698	0.51895	0.75
9	4000	0.80275	0.81704	16.4	65.4	0.86408	0.25412	0.38712	0.82
9	9375	0.80298	0.81756	16.6	57.8	0.88475	0.28773	0.42707	0.83
9	15000	0.81386	0.82218	16.0	54.0	0.89147	0.29653	0.43256	0.8
9	30000	0.81560	0.82593	15.6	57.6	0.88010	0.27254	0.40335	0.78
10	4000	0.79725	0.78792	14.0	36.8	0.92558	0.38128	0.49341	0.7
10	9375	0.79712	0.79107	14.6	33.0	0.93850	0.44272	0.55103	0.73
10	15000	0.81067	0.80038	14.8	31.8	0.94263	0.46643	0.57154	0.74
10	30000	0.81460	0.80829	13.8	30.8	0.94005	0.45139	0.54490	0.69
11	4000	0.75250	0.70201	10.2	16.2	0.95917	0.64394	0.56328	0.51
11	9375	0.76235	0.71090	9.2	16.2	0.95400	0.57218	0.50874	0.46
11	15000	0.77527	0.73438	10.4	18.0	0.95555	0.57875	0.54712	0.52
11	30000	0.77743	0.73831	10.6	18.6	0.95504	0.57134	0.54966	0.53

Table 6.7 Results of Experiment Art8, Art9, Art10 and Art11

6.2.7 Experiment Art12 - Cased Model

This last experiment shares all the settings of Experiment of Section 6.2.5, with only one substantial difference: a different pre-trained BERT model. The model used in this experiment is the base version of the cased model, namely bert_cased_L−12_H−768_A−12/1.

Data Quantity	BERT Accuracy	BERT F1-Score	Right Predictions	Total Predictions	Validator Accuracy	Validator Precision	Validator F1-Score	Validator Recall
4000	0.79450	0.79739	14.6	42.0	0.91524	0.34925	0.47215	0.73
9375	0.80960	0.81135	14.8	41.6	0.91731	0.35619	0.48068	0.74
15000	0.81327	0.81601	15.6	45.8	0.91059	0.34129	0.47449	0.78
30000	0.81603	0.81785	15.6	42.4	0.91938	0.36837	0.50025	0.78

Table 6.8 Results of Experiment Art12

Results of this experiment are available in Table 6.8 and reflects a general performance improvement with respect to Experiment of Section 6.2.5. Another important information is that the fine-tuning time is halved.

6.2.8 News Articles Experiment Comparisons

Summary of Experiments In order to recap each experiment setting, Table 6.9 is provided.

Experiment	50000 Data	Balancing Same-New	Mono Category	Distribution Wise	Different Data (Train/Validation)	Model
Art1	✓	Random	✗	✗	✗	Uncased
Art2	✗	50-50	✗	✗	✗	Uncased
Art3	✓	50-50	✗	✓	✗	Uncased
Art4	✗	50-50	✓	✓	✗	Uncased
Art5	✗	50-50	✓	✓	✗	Uncased
Art6	✗	50-50	✓	✓	✗	Uncased
Art7	✗	50-50	✗	✓	✓	Uncased
Art8	✗	20-80	✗	✓	✓	Uncased
Art9	✗	40-60	✗	✓	✓	Uncased
Art10	✗	60-40	✗	✓	✓	Uncased
Art11	✗	80-20	✗	✓	✓	Uncased
Art12	✗	50-50	✗	✓	✓	Cased

Table 6.9 Experiments News Articles Scenario Summary Table

Accuracy Figure 6.2 shows *BERT Accuracy* and *Validator Accuracy* of each experiment, in order to give a comparison between all experiments.

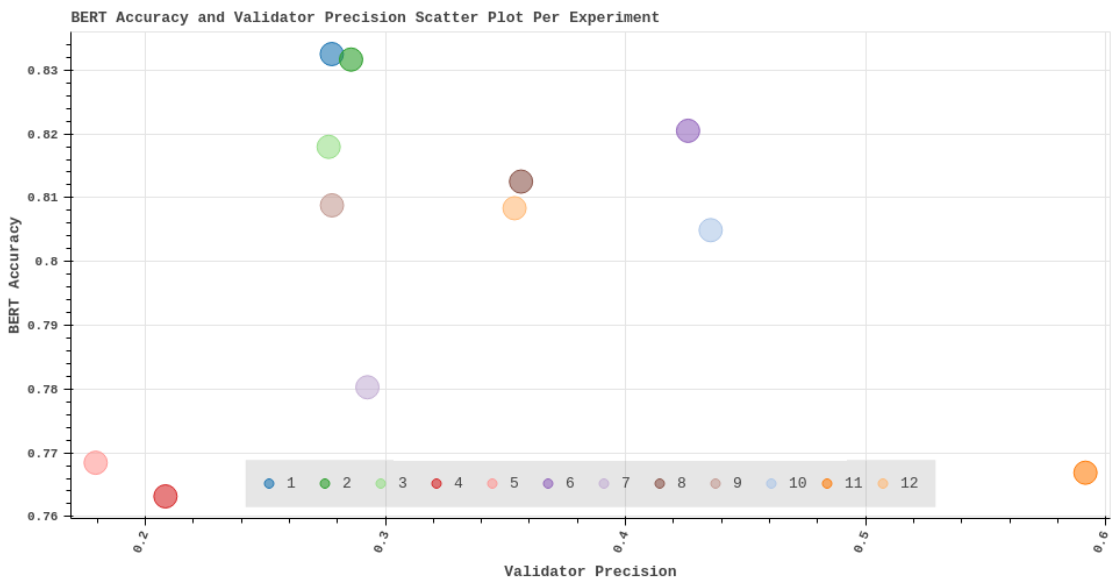


Figure 6.2 BERT Accuracy and Validator Precision per Experiment

Besides *BERT Accuracy*, the other variable chosen is *Validator Precision*, since this variable in this specific use-case is considered to be the most significant between Validator performance indexes because it shows the relationships between *Right Predictions* and *Total Predictions*. The importance of this relationship comes from the fact that a high value of total predictions influences the value of right predictions.

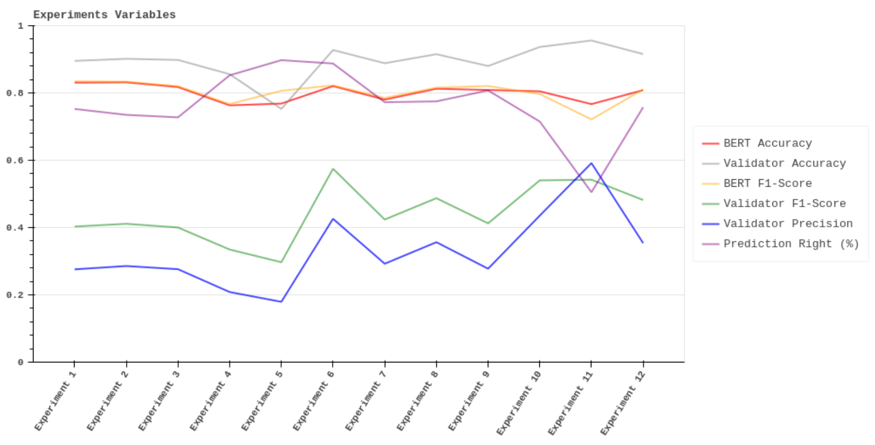


Figure 6.3 All Estimators per Experiment

Comparison Between All the Variables From Figure 6.3 all the variables included between 0 and 1 are shown and it is also included the value of right predictions in percentage. It is important to highlight that Experiment Art11 is characterized by the highest value of

Validator Precision having the least value of *Right Predictions*. *BERT Accuracy* is stable for each experiment, while the best values of *Right Predictions* is provided by the Mono-Category experiments.

Total Predictions Figure 6.4 shows the total number of predictions. From this figure it can be seen that *Experiment Art11* is the one with the lowest number of *Total Predictions*, reflected in its corresponding value of *Validator Precision* that is the highest between all the experiments.

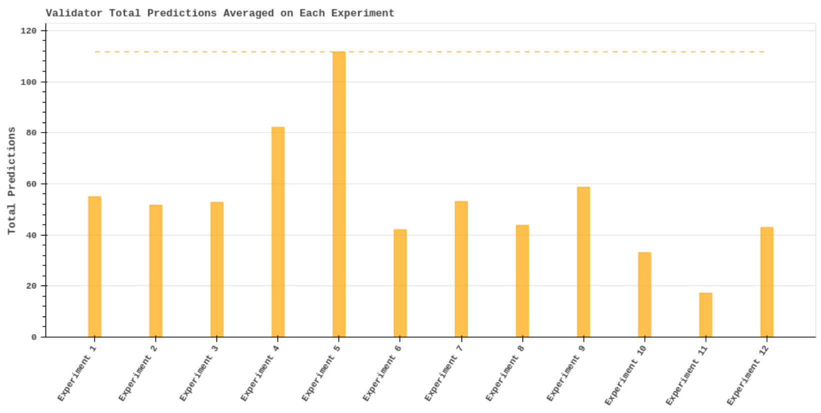


Figure 6.4 Validator Total Predictions per Experiment

WinDiff-Like Evaluation Another interesting evaluation can be seen in Figure 6.5, where a windiff-like plot is shown. This figure represents how distant the prediction of the model is with respect to the line of the start of the new segment, in this scenario a new article.

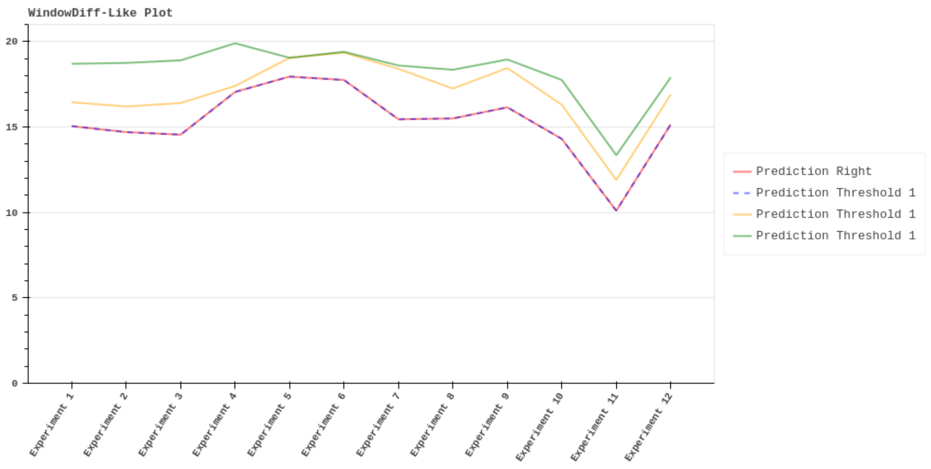


Figure 6.5 WinDiff-Like Plot per Experiment

Figure 6.5 shows that each model predicts the same number of new segments using a 1-sentence threshold of tolerance, while, increasing this threshold, the number of correct predictions increases.

Comparison Between Experiments Art7 and Art12 The last comparison that will be offered is the comparison between the same experiment with different models. Figure 6.6 shows how the two models perform with the same setting and the same dataset.

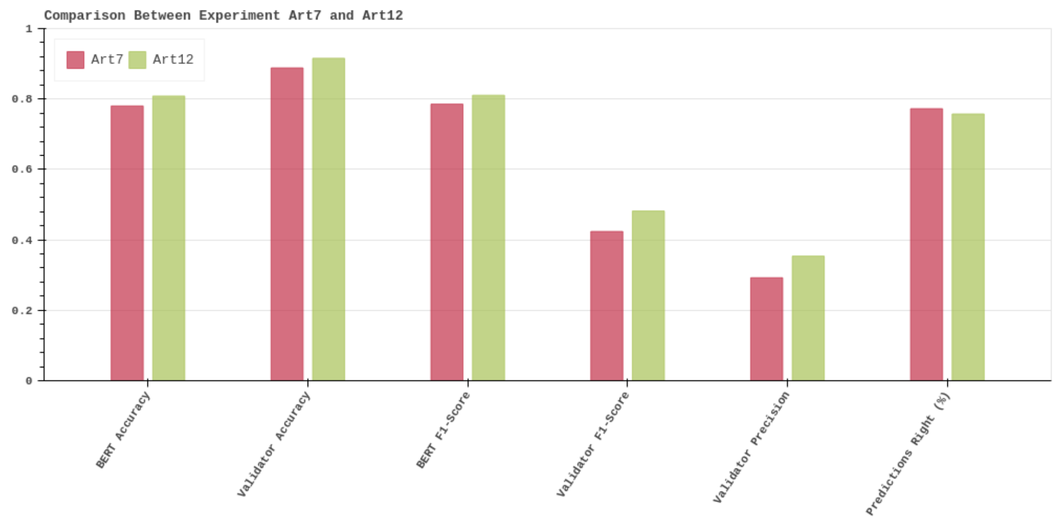


Figure 6.6 Comparison Between Cased and Uncased Model

Figure 6.6 compares all the significant variables for experiment Art7 and Art12. As it can be seen the **Cased** model performs slightly better that the **Uncased** version.

6.3 Books Experiments

This Section explains each experiment about the Books Scenario. All experiments have been named according to the following convention:

- Experiment concerning Chapters: all the experiments, which consider the chapters as new segments, are named with ChX, where X corresponds to an incremental number.
- Experiment concerning Paragraphs: all the experiments, which consider the paragraphs as new segments, are named with PX, where X corresponds to an incremental number.

Each experiment of this scenario was executed with a K-Fold Cross Validation K equal to 5 and for different quantities of data, such 4000, 9375, 15000, 30000. These quantities of data have been chosen under the considerations of Section 5.2 and are divided as 80% for the training phase and 20% for the test phase. The reference Data Quantity is 9375 since it results in 7500 data for the training phase.

For each experiment of this Scenario, a dataset with the following properties has been used to train the pre-trained BERT model: sentence coupling has been accomplished by assigning to each book a probability prob_book given by its size in terms of number of chapters, then another probability prob_chapter is assigned to each chapter of each book. Therefore, the probability of each sentence being chosen results to be defined as follows:

$$prob\_sentence = prob\_book * prob\_chapter$$

Under this assumption, the probability that a sentence, belonging to a book B and a chapter C, is chosen increases the more B contains chapters and the more C contains sentences. This particular addition has been introduced to avoid chapters formed by too less sentences.

In the following subsections, all the experiments of this scenario are exhaustively explained. Moreover, at the end of this section, a summary, that includes all the experiments, is commented.

6.3.1 Experiment Ch1 - Balanced

This experiment's dataset has been built using above described considerations. For this particular experiment, are provided also five iterations with 50000 couples of data (40000 of them used for training purposes and 10000 for testing).

Results of this experiment are shown in Table 6.10. Comparing these results with previous scenario experiments, it can be seen how this scenario performs drastically worse.

It is important to remind that this validator file consists of 13 chapters, for a total of 2875 sentences.

Data Quantity	BERT Accuracy	BERT F1-Score	Right Predictions	Total Predictions	Validator Accuracy	Validator Precision	Validator F1-Score	Validator Recall
4000	0.56400	0.66611	1.4	232.4	0.91422	0.00567	0.01059	0.08
9375	0.58762	0.66655	1.8	224.2	0.91735	0.00764	0.01423	0.11
15000	0.59133	0.66663	1.2	263.8	0.90316	0.00460	0.00864	0.07
30000	0.60097	0.66665	1.4	239.0	0.91193	0.00588	0.01097	0.08
50000	0.60608	0.66667	2.4	276.0	0.89976	0.00880	0.01652	0.14

Table 6.10 Results of Experiment Ch1

This experiment presents an high value of Total predictions with a very low value of Right predictions . These values are reflected by BERT parameters that show a very low value.

These worst results come from:

- A greater difficulty of the experiment due to the particular kind of data: a chapter is composed by a greater number of sentences of an article, resulting in a more general partition of the segments that influences negatively BERT performance.
- A worse quality of the data worsens BERT estimations. Book sentences in the books could have such peculiarities that they cannot be contextualized if taken out of context.

In this experiment sentences that form the dataset are taken as they are, no additional pre-processing (in addition to the split) was carried out on the data.

The following experiments will change these results by adding further pre-processing.

6.3.2 Experiment Ch2 - Distribution-Wise

Following the structure used for the scenario explained in Section 6.2, the second experiment's dataset is built under an analysis of the distribution of the data under different facets:

- Books length in terms of chapters: as it can be seen in Figure 6.7, the collection of books used to build the dataset is characterized by books around the 12 chapters, but with several outliers.
- Chapters length in terms of sentences number: Figure 6.7 shows that the number of sentences in each chapter is characterized by a diversified distribution, with many outliers, due to the fact that the books, to which the chapters in question belong, did not have a clear distinction of chapters and through regular expressions it was not possible to identify a pattern of division of the chapters.

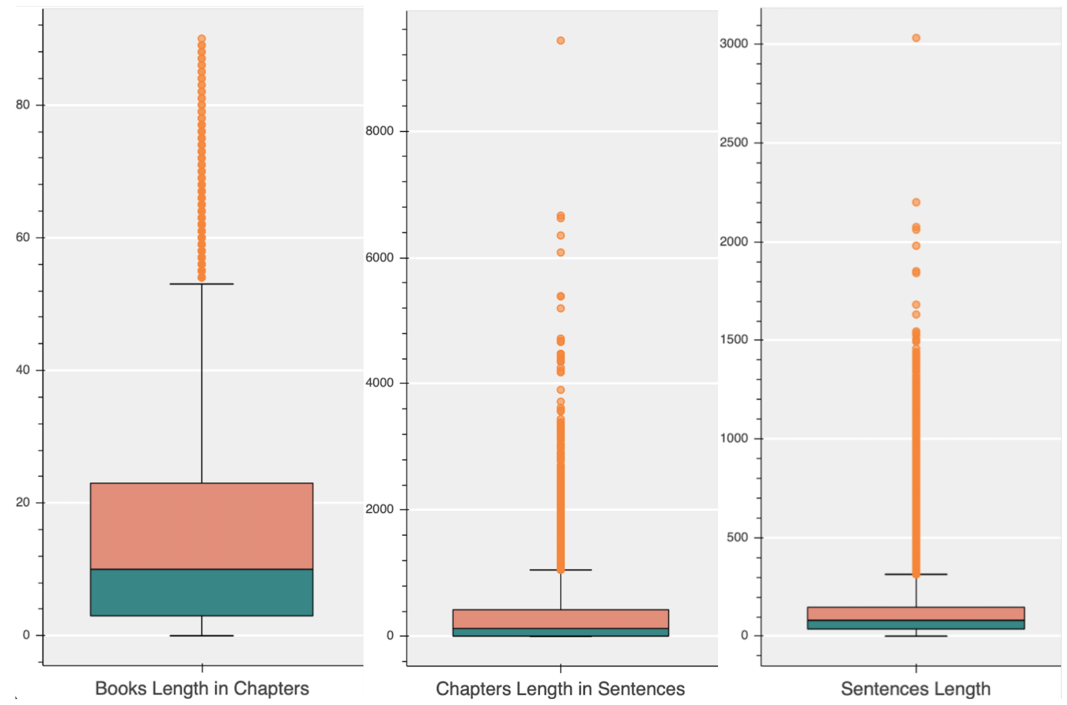


Figure 6.7 Books Distribution

- Sentences length: phrases that are too short or too long can negatively affect the fine-tuning of the BERT pre-trained model.

The reference values of Figure 6.7 can be seen in Table 6.11, where the quartiles together with the minimum and the maximum values are collected.

	25%	***50%***	***75%***	***mean***	***std***
Books Length in Chapters	3	10	23	16,06	17,09
Chapters Length in Sentences	4	127	423	376,23	693,52
Sentences Length	37	81	148	107,42	99,14

Table 6.11 Books Distribution Reference Values

After this analysis it was decided to remove:

- Books with fewer chapters than the 25 percentile and with more chapters than the 75 percentile, in order to bypass possible mistakes due from the regular expression.
- Chapters with fewer sentences than the 25 percentile and with more sentences than the 75 percentile, since with too many or too fewer lines the text segment loose coherence.

- Sentences too short or too long to provide a more general dataset in fine-tuning.

This additional pre-processing activity has had beneficial effects on the results obtained in training, demonstrating a general increase in all the estimating variables, in particular for the BERT estimates, that shows a 10% growth with respect to the previous experiment, as Table 6.12 shows.

Data Quantity	BERT Accuracy	BERT F1-Score	Right Predictions	Total Predictions	Validator Accuracy	Validator Precision	Validator F1-Score	Validator Recall
4000	0.72800	0.73607	2.6	29.8	0.93126	0.09013	0.12346	0.20
9375	0.74784	0.75419	1.2	23.4	0.93784	0.05123	0.06562	0.09
15000	0.76053	0.77076	2.2	27.8	0.93345	0.08148	0.10954	0.17
30000	0.75753	0.76463	1.8	21.0	0.94442	0.08612	0.10593	0.14

Table 6.12 Results of Experiment Ch2

Using the same methodology, a validator file of 17 chapters was built, corresponding to 547 sentences. This explains the lower number of *Total Predictions*.

6.3.3 Experiment Ch3 and Ch4 - Validation with New Data

In order to proceed in the same logical way of the Scenario 6.2, the data constituting the validation file are now obtained from a portion of data detached from those that create the dataset.

Just before the pairs of sentences are randomly matched, 10% of them are extracted and used later in order to obtain a validator file constructed with new data compared to the data used to create the training (fine-tuning) dataset.

The difference between these two experiments derives only from the validator file. In experiment Ch3, the validator file is built by a random sample of a book in the portion of data stored aside for the validation, while the validator file of the experiment Ch4 is built by concatenating random chapters from different books (always in the validation portion of the data), like the ones of the scenario of Section 6.2. Validator of Experiment Ch3 consists precisely of a book of 20 chapters and of 4326 sentences, while Validator of Experiment Ch4 consists of 20 chapters and 1458 sentences.

Table 6.13 shows how these two experiments behave, giving the opportunity to see in detail their difference.

In terms of BERT estimation, there is no significant difference between these experiments and the previous, because, using at most 50.000 pairs of sentences, the maximum number of sentences used is 100.000, a very low number compared to the total number of sentences

Experiment	Data Quantity	BERT Accuracy	BERT F1-Score	Right Predictions	Total Predictions	Validator Accuracy	Validator Precision	Validator F1-Score	Validator Recall
3	4000	0.73625	0.74629	1.2	24.0	0.94658	0.04311	0.05843	0.09
3	9375	0.74496	0.75246	0.6	19.4	0.95271	0.02727	0.03428	0.04
3	15000	0.75380	0.76015	1.2	24.0	0.94649	0.04110	0.05659	0.09
3	30000	0.75893	0.76632	0.6	16.8	0.96226	0.03131	0.03723	0.04
4	4000	0.72925	0.74177	6.8	66.0	0.95031	0.10349	0.15850	0.34
4	9375	0.74357	0.75246	8.0	65.8	0.95209	0.12164	0.18653	0.4
4	15000	0.74553	0.75378	8.4	67.8	0.95127	0.12406	0.19153	0.42
4	30000	0.75500	0.76373	8.6	72.0	0.94866	0.11966	0.18718	0.43

Table 6.13 Results of Experiment Ch3 and Ch4

(1.311.936 sentences). The validity of this experiment is however given by the use of probabilities in the choice of the latter: since each sentence has a different probability of being chosen to become part of a couple, the choice of couples could involve the same sentences every time. However, in pairs of sentences, there are no duplicate pairs, as they are removed when creating the dataset. In this way, the possibility of having the same data in training and validation has been removed.

Instead, in terms of predictions, the two experiments differ significantly. This difference comes from the different validator file, above described.

6.3.4 Experiment Ch5, Ch6, Ch7 and Ch8 - Different Balancing

All these experiments share the same setting used by Experiment Ch4 of the previous section. The only difference is about the percentages of couples of the same chapter or not.

In the following table, the different percentages used are shown.

Experiment	From the Same Chapter	From Different Chapters
Ch5	80	20
Ch6	60	40
Ch7	40	60
Ch8	20	80

Table 6.14 Percentages of Experiments Ch5, Ch6, Ch7 and Ch8

Table 6.15 shows the different results obtained by each couple of percentages. From this table it can be seen that moving away from a balanced percentage, BERT estimates deteriorate considerably.

Total Predictions on the validator decreases as the percentages of couples from different chapters increases; the same behaviour characterizes the Right Predictions. Another impor-

Experiment	Data Quantity	BERT Accuracy	BERT F1-Score	Right Predictions	Total Predictions	Validator Accuracy	Validator Precision	Validator F1-Score	Validator Recall
5	4000	0.62100	0.72340	13.0	368.2	0.75141	0.03847	0.07204	0.65
5	9375	0.70378	0.76319	10.0	175.8	0.87934	0.05724	0.10260	0.5
5	15000	0.69487	0.75335	9.8	198.4	0.86355	0.05184	0.09327	0.49
5	30000	0.71950	0.77000	9.8	158.2	0.89115	0.06195	0.10998	0.49
6	4000	0.74350	0.77237	8.8	89.0	0.93727	0.09970	0.16240	0.44
6	9375	0.74144	0.76205	8.6	90.4	0.93603	0.09499	0.15554	0.43
6	15000	0.74006	0.76394	9.2	109.4	0.92381	0.08967	0.14889	0.46
6	30000	0.75160	0.77117	8.8	93.8	0.93397	0.09394	0.15478	0.44
7	4000	0.74700	0.73736	5.6	43.0	0.96445	0.12979	0.17727	0.28
7	9375	0.73877	0.72750	6.2	51.4	0.95951	0.12169	0.17443	0.31
7	15000	0.73953	0.73112	8.2	53.2	0.96101	0.15439	0.22389	0.41
7	30000	0.74903	0.73747	6.8	50.0	0.96130	0.13712	0.19499	0.34
8	4000	0.63750	0.66256	1.2	14.0	0.97831	0.07612	0.06669	0.06
8	9375	0.64160	0.66406	1.6	13.2	0.97941	0.10172	0.08919	0.08
8	15000	0.67793	0.66484	3.0	20.0	0.97666	0.15325	0.15081	0.15
8	30000	0.69153	0.66968	2.6	17.0	0.97817	0.15354	0.14048	0.13

Table 6.15 Results of Experiments Ch5, Ch6, Ch7 and Ch8

tant consideration is that the validator performance indexes show better results the more the percentages are balanced. All these four experiments share the same validator file of Experiment Ch4.

6.3.5 Experiment Ch9 - Cased Model

As for the news articles scenario, a comparison is proposed between BERT's cased and uncased model. This experiment is the last dealing with chapters as text segment and its results are presented in Table 6.16. In the comparison section, it is possible to see the difference between the two equal experiments with different models.

Data Quantity	BERT Accuracy	BERT F1-Score	Right Predictions	Total Predictions	Validator Accuracy	Validator Precision	Validator F1-Score	Validator Recall
4000	0.74900	0.75370	6.0	67.6	0.94811	0.08899	0.13714	0.3
9375	0.74698	0.75337	6.2	69.8	0.94687	0.08889	0.13815	0.31
15000	0.75040	0.75543	8.2	74.6	0.94632	0.11013	0.17352	0.41
30000	0.75536	0.75959	7.6	74.2	0.94578	0.10256	0.16152	0.38

Table 6.16 Results of Experiment Ch9

6.3.6 Chapters Experiments Comparison

Summary of Experiments In order to recap each experiment setting, Table 6.17 is provided.

Experiment	50000 Data	Balancing Same-New	Distribution Wise	Different Data (Train/Validation)	Model
Ch1	✓	50-50	✗	✗	Uncased
Ch2	✗	50-50	✓	✗	Uncased
Ch3	✗	50-50	✓	✓	Uncased
Ch4	✗	50-50	✓	✓	Uncased
Ch5	✗	20-80	✓	✓	Uncased
Ch6	✗	40-60	✓	✓	Uncased
Ch7	✗	60-40	✓	✓	Uncased
Ch8	✗	80-20	✓	✓	Uncased
Ch9	✗	50-50	✓	✓	Cased

Table 6.17 Experiments Books Scenario Summary Table

Accuracy Figure 6.8 shows *BERT Accuracy* and *Validator Accuracy* of each experiment of the Books Scenario, in order to provide a comparison between all the experiments in the accuracy context.

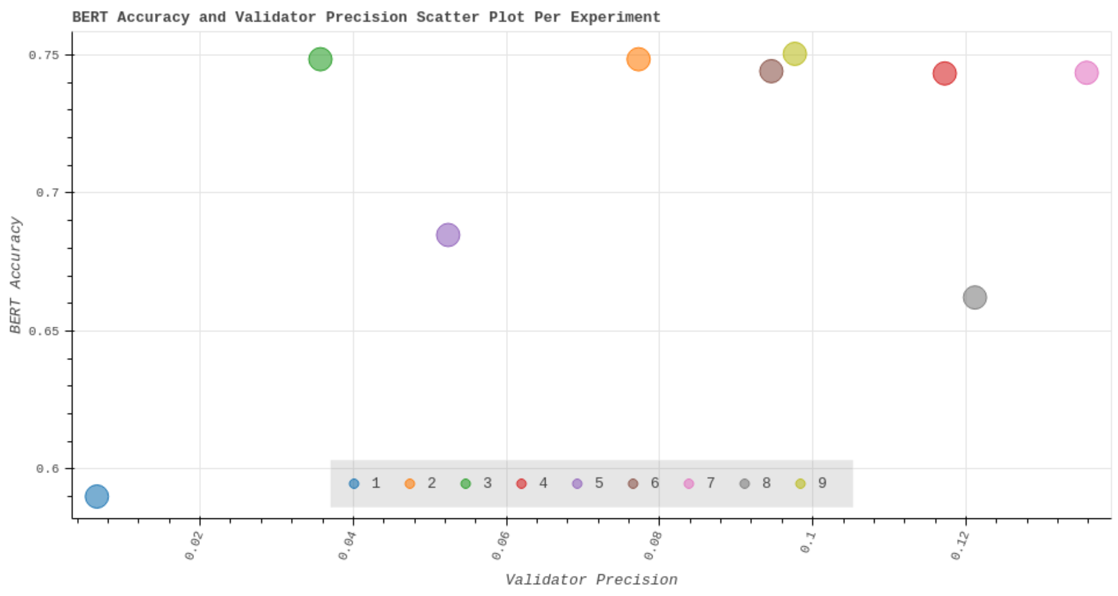


Figure 6.8 BERT Accuracy and Validator Precision per Experiment

Besides *BERT Accuracy*, the other variable chosen is *Validator Precision*, since this variable in this specific use-case is considered to be the most significant between Validator performance indexes because it shows the relationships between *Right Predictions* and *Total*

Predictions. The importance of this relationship comes from the fact that a high value of total predictions influences the value of the right predictions.

Comparison Between all the Variables Figure 6.9 shows all the variables included between 0 and 1 and it is also included the value of right predictions in percentage.

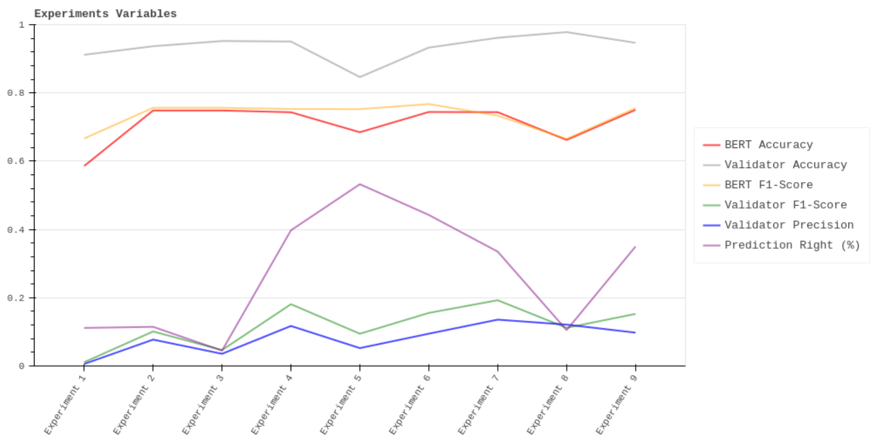


Figure 6.9 All Estimators per Experiment

Total Predictions Figure 6.10 shows the total number of predictions. From this figure, it can be seen that Experiment Ch8 is the one with the lowest number of Total Predictions.

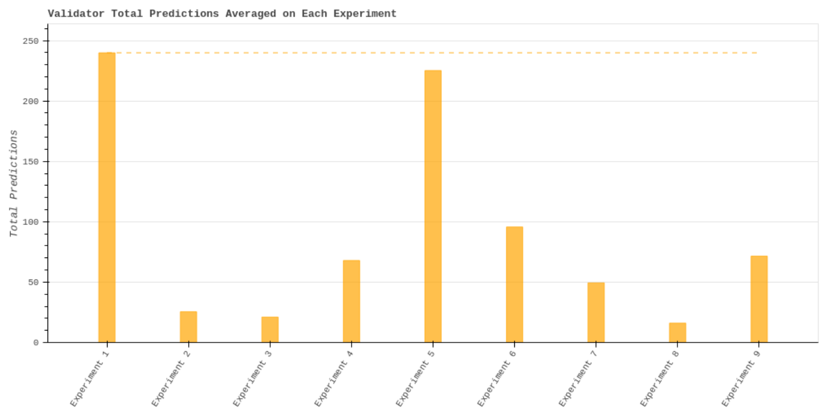


Figure 6.10 Validator Total Predictions per Experiment

WindDiff-Like Evaluation Another interesting evaluation can be seen in Figure 6.11, where a windiff-like plot is shown. This figure represents how distant the prediction of the model is with respect to the line of the start of the new segment, in this scenario a new chapter.

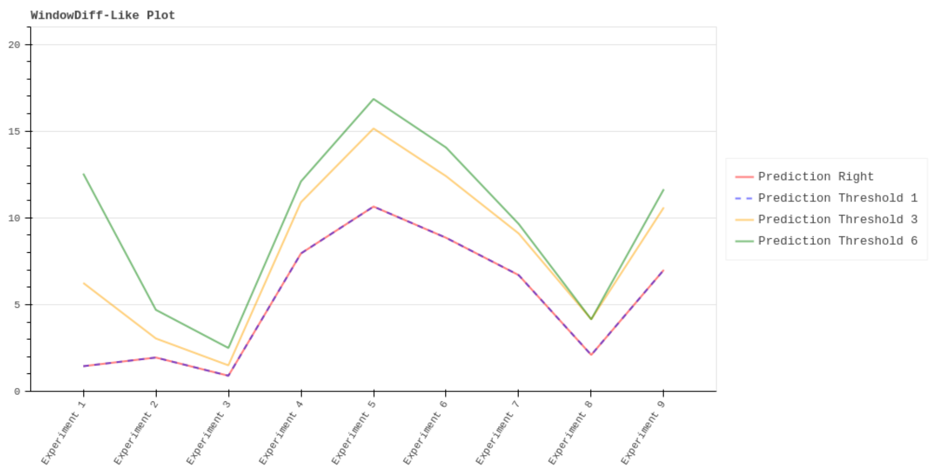


Figure 6.11 WinDiff-Like Plot per Experiment

Figure 6.11 shows that each model predicts the same number of new segments using a 1-sentence threshold of tolerance, while, increasing this threshold, the number of correct predictions increases.

Comparison Between Experiments Ch4 and Ch9 The last comparison that will be offered is the comparison between the same experiment with different models. Figure 6.12 shows how the two models perform with the same setting and the same dataset.

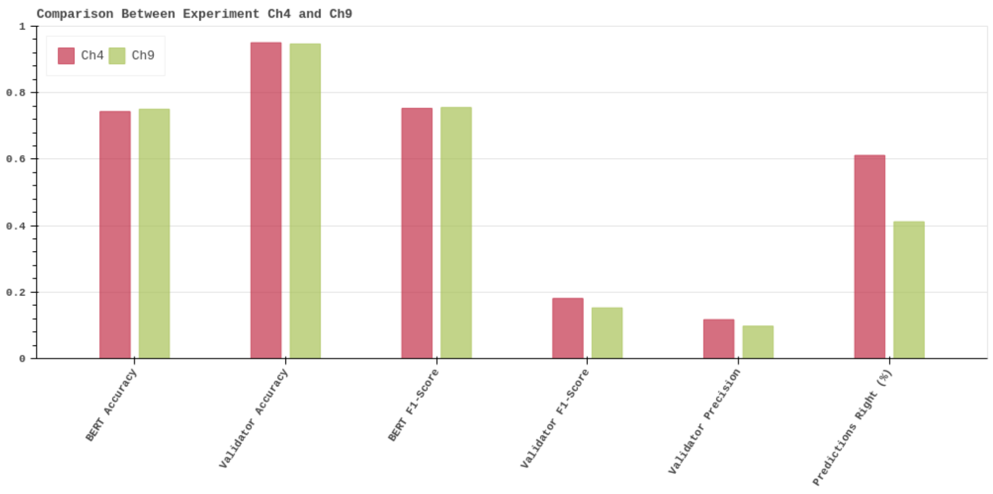


Figure 6.12 Comparison Between Cased and Uncased Model

Figure 6.12 compares all the significant variables for experiment Ch4 and Ch9. As it can be seen, unlike the previous scenario, the **Uncased** model performs slightly better than the **Cased** version.

6.3.7 Paragraphs Experiment P1 and P2

Experiment	Data Quantity	Time (s)	BERT Accuracy	BERT F1-Score	Right Predictions	Total Predictions	Validator Accuracy	Validator Precision	Validator F1-Score	Validator Recall
1	4000	573.2	0.73775	0.74831	27.0	52.2	0.89102	0.52063	0.60645	0.73
1	9375	1207.4	0.75210	0.75863	26.8	52.4	0.88916	0.51162	0.59960	0.72
1	15000	1869.2	0.74666	0.75521	26.6	59.2	0.86687	0.45760	0.55748	0.72
1	30000	3723.4	0.76010	0.76653	27.0	58.0	0.87306	0.46659	0.56891	0.73
2	4000	229.6	0.73825	0.74545	25.6	47.6	0.89660	0.54234	0.60678	0.69
2	9375	433.6	0.75051	0.75617	25.4	50.0	0.88792	0.50934	0.58428	0.69
2	15000	647.0	0.75013	0.75469	25.8	51.2	0.88669	0.50508	0.58556	0.70
2	30000	1215.2	0.76277	0.76844	26.8	56.2	0.87740	0.47717	0.57524	0.72

Table 6.18 Comparison Results of Experiment P1 and P2

These two experiments differ only in the choice of the model: Experiment P1 is executed with the *uncased* version of BERT model, while Experiment P2 with the *cased* version.

The configuration shared between the two consists in having to indicate a segment of text defined by a paragraph instead of a chapter.

There is no statistical evidence about the right length of a paragraph. For this reason, these text segment can be seen more like a batch of sentences rather than real paragraphs.

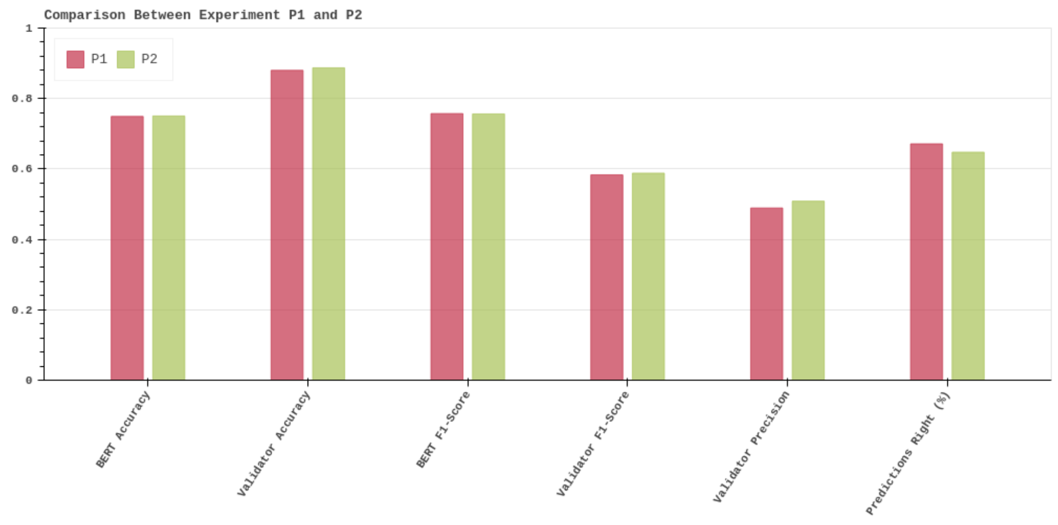


Figure 6.13 Comparison Between Cased and Uncased Model

However, the dataset consists of a couple of sentences that have been processed exactly like in experiments before, yet these couples are taken from batches of sentences of a variable, random length between 6 and 12 sentences.

Like the dataset, the validator has undergone the same pre-processing process and it is composed of 40 batches appended consequently.

	Accuracy	*Precision*	*F1-Score*	*TP*	*TN*	*FP*	*FN*	*All*
News Articles	0.89002	0	0	0	785	96	1	882
Chapters	0.87178	0.08333	0.04273	5	1518	50	174	1747

Table 6.19 Estimators on Attention-Based Neural Text Segmentation

In Table 6.18, the results about these two experiments are shown, in addition, Figure 6.13 graphically presents all their interesting variables.

From Table 6.18 and Figure 6.13 it is possible to establish that the cased model obtains practically same results as the uncased model, but with half the time taken by the uncased version of BERT model.

6.4 Comparison with Related Works

From the beginning, one of the objectives was to be able to establish whether the use of the BERT pre-training model was characterized by comparable, or even better, performance of solutions adopted so far, such as Attention-Based Neural Text Segmentation [2], in the area of Text Segmentation.

For this reason, a comparison of the results obtained by both implementations on the same dataset is proposed. Both scenarios proposed in the above experiments have been implemented according to the approach in question.

This solution is characterized by an higher value of accuracy, even if the predictions are scarce. From these results we can conclude that the method proposed in this thesis to find new segments of text provides better results in terms of predictions despite a lower accuracy.

Table 6.19 shows the results obtained to provide a better description of the comparison between the two solutions.

Chapter 7

Conclusion

This thesis proposes a fully-automatic method to segment texts in natural language. The approach consists in the application of a method of pre-training language representations, called BERT (Bidirectional Encoder Representation from Transformer), in order to understand when a new text segment starts in a flow of sentences. This approach allows partitioning a text into meaningful units by detecting the start of new text segments.

7.0.1 Summary of the Results

Among the different configurations that this work has tested, the best performance have been achieved by the experiments carried out on the News Articles Scenario. The difference in the results between the different scenarios also lies in the different complexity of the same. the length of the segments to be partitioned is a fundamental variable in the complexity of the experiment itself, as shown by the results, in which the chapter scenario, characterized by the greatest number of sentences per segment, provides the worst results. The other fundamental variable that decides the complexity of the scenario is the quality of the data, better as regards the scenario of the News Articles. Table 7.1 shows all the important variables averaged by scenario. These data reflect the considerations written above.

Scenario	*BERT Accuracy*	*BERT F1-Score*	*Right Predictions (%)*	*Validator Accuracy*	*Validator Precision*	*Validator F1-Score*
News Articles	80,24%	80,43%	76,41%	89,39%	32,80%	44,10%
Chapters	70,95%	73,22%	26,81%	93,42%	8%	11,36%
Paragraphs	74,98%	75,67%	65,94%	88,36%	49,88%	58,55%

Table 7.1 Mean Results for each Scenario

7.0.2 Contributions

With the huge amount of data available on the internet, the need to transform information to capture its meaning is always higher. Thanks to the many Text Segmentation tasks it is possible to manage this information to obtain usable and punctual content.

Text Segmentation can be used to improve many other Natural Language Processing tasks, such as Text Summarization: breaking a document into sections before summarizing will ensure that the summary includes all the topics that were covered in the document.

This work focuses on the usage of two different sources of data to understand the potential of the pre-trained model BERT, that outperforms previous models in simple Natural Language Processing tasks, but this approach can be extended to any source of data.

Furthermore, this method can be extended and improved in several ways, but the results obtained suggest that this could be a significant starting point for many other works concerning Text Segmentation using BERT pre-trained model.

7.0.3 Future Work

If considered as a starting point, this work can be extended in many directions. First of all, having the possibility to use more powerful hardware, it would be curious to try to use other BERT models among those proposed, above all the most recent Whole Word Masking (May 31st, 2019), which is proposed in a Cased and an Uncased version.

It would also be interesting to evaluate the results of the use of human supervised data. The fine-tuned model could be also used in addition to other models to perform different NLP tasks, such as Text Summarization. Text Segmentation tasks can also be used to improve other tasks by integrating image analysis or integrating with query systems for Information Retrieval: performing query similarity measures against a sections of document as supposed to the whole document; when displaying the search result, you can display the most relevant portion of the document to the query [1].

Bibliography

[1] Arivazhagan, M., Srinivasan, H., and Srihari, S. N. (2007). A statistical approach to line segmentation in handwritten documents. In *DRR*.

[2] Badjatiya, P., Kurisinkel, L. J., Gupta, M., and Varma, V. (2018). Attention-based neural text segmentation. *CoRR*, abs/1808.09935.

[3] Caudill, M. (1987). Neural networks primer, part i. *AI Expert*, 2(12):46–52.

[4] Dai, W., Yang, Q., Xue, G.-R., and Yu, Y. (2008). Self-taught clustering. In *Proceedings of the 25th international conference on Machine learning*, pages 200–207. ACM.

[5] Fan, W., Wallace, L., Rich, S., and Zhang, Z. (2006). Tapping the power of text mining. *Communications of the ACM*, 49:76–82.

[6] Fushiki, T. (2011). Estimation of prediction error by using k-fold cross-validation. *Statistics and Computing*, 21(2):137–146.

[7] Howard, J. and Ruder, S. (2018). Fine-tuned language models for text classification. *CoRR*, abs/1801.06146.

[8] III, H. D. and Marcu, D. (2011). Domain adaptation for statistical classifiers. *CoRR*, abs/1109.6341.

[9] Lioma, C., Larsen, B., Petersen, C., and Simonsen, J. (2016). Deep learning relevance: Creating relevant information (as opposed to retrieving it).

[10] Luntz, A. C. and Brailovsky, V. L. (1969). On estimation of characters obtained in statistical procedure of recognition.

[11] Malmasi, S., Dras, M., Johnson, M., Du, L., and Wolska, M. (2017). Unsupervised text segmentation based on native language characteristics. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1457–1469, Vancouver, Canada. Association for Computational Linguistics.

[12] Naili, M., Chaibi, A. H., and Ben Ghezala, H. H. (2017). Comparative study of word embedding methods in topic segmentation. *Procedia Comput. Sci.*, 112(C):340–349.

[13] Pak, I. and Teh, P. (2018). *Text Segmentation Techniques: A Critical Review*, volume 741, pages 167–181.

[14] Pardo, T. A. S. and das Graças Volpe Nunes, M. (2003). Segmentaçao textual automática: Uma revisão bibliográfica. *Universidade de São Paulo-USP Universidade Federal de São Carlos-UFSCar Universidade Estadual Paulista–UNESP (*, 2003:21.

[15] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *CoRR*, abs/1802.05365.

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

[17] Raina, R., Battle, A., Lee, H., Packer, B., and Y Ng, A. (2007). Self-taught learning: Transfer learning from unlabeled data. *Proceedings of the Twenty-fourth International Conference on Machine Learning*, 227.

[18] Ramos, J. et al. (2003). Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142.

[19] Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., and Liu, C. (2018). A survey on deep transfer learning. *CoRR*, abs/1808.01974.

[20] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention Is All You Need. *arXiv e-prints*, page arXiv:1706.03762.

[21] Wang, Z., Song, Y., and Zhang, C. (2008). Transferred dimensionality reduction. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 550–565. Springer.

[22] Wirth, R. and Hipp, J. (2000). Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, pages 29–39. Citeseer.