POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

Temporal Summarization: a Transformer-Based Approach

Supervisors

Prof. Luca CAGLIERO

Dott. Moreno LA QUATRA

Candidate

Carlo PELUSO

July 2022

Summary

Context description: During the last decades, along with the rise of the World Wide Web, exponentially increasing amounts of textual data have been produced from a wide range of sources, at very high rates. Even if - at first glance - we might have an infinite potential for learning, huge amounts of data actually create confusion and became easily outdated.

This unstoppable and rapidly growing data production trend has demanded researchers to develop and study approaches for automatic text summarization methods, in which from various textual sources shorter summaries are produced, while keeping unaltered (as much as possible) the overall content of the source documents.

Temporal Summarization: In the context of large unforeseen events (e.g. natural disasters) the data production rate reaches even higher peaks and, at the same time, the novelty of the data produced lasts even less than usual. For this particular cases, researchers have also introduced the task of Temporal Summarization, in which the automatic text summarization task is enriched with further constraints: in this context, the corpus that has to be summarized grows rapidly and a newly published document remains relevant for a short interval of time; indeed, the documents must be processed promptly in order to prevent the information from aging.

Thesis goal: This thesis presents a Temporal Summarization system able to process a timestamped corpus containing textual documents related to evolving events (such as natural disasters or mass protests) and produce concise and up-to-date sentences by capturing the relevant portions of the corpus.

The corpus exploited for both the development and test of the system proposed in this work is provided by the TREC Temporal Summarization conference, which is the major reference for this thesis both in terms of evaluation metrics and task objectives.

In particular, this work enriches the Sequential Update Summarization task objectives proposed within the TREC research contest, that originally requires to solely sample from a timestamped corpus a list of documents that are considered relevant: differently, the task addressed in this work aims at capturing relevant

portions from the documents - indeed, not only selecting entire documents - and finally to iteratively emit short, newly generated sentences that embed the relevant information acquired.

In order to address this task, the data is initially gathered, processed and collected on cloud along a computationally intensive dataset generation procedure.

Next, cutting-edge Deep Learning architectures are exploited through the fine-tuning procedure and evaluated among various experimental setups; during this phase emerged the most promising span extraction model, which is the core of the whole system, in charge of properly capture and identify relevant portions from the input texts.

Finally, the most performing model is exploited by working as the principal module of the system, conducted by means of a summaries production procedure that orchestrates the emission of freshly-generated sentences to the end users.

Experiments: Despite the task proposed in this work slightly differs from the Sequential Update Summarization task proposed by the TREC Temporal Summarization conference, the system proposed is evaluated on the official metrics proposed and compared to the participant systems.

The proposed method is always comparable to the top-tier participants of the TREC research contest and resulted to be very accurate in capturing relevant information from documents related to a wide variety of different topics and systematically identify significant overall proportions of available information.

Contribution outline: The main contributions of this work are the following:

- The Sequential Update Summarization task was channeled into a full downstream Abstractive Temporal Summarization task, in which newly generated data is emitted to the end users
- The Temporal Summarization task is unprecedented addressed through innovative Deep Learning architectures, with respect to traditional shallow architectures and optimization methods
- The Text Span Detection task for selecting (sub)sentences from a corpus is addressed through cutting-edge pre-trained Deep Learning models finetuned on the Token Classification task, resulting to be very effective.

Future works: Still, the system proposed in this works has some drawbacks: in particular, it lacks of an information relevance scoring method and indeed periodically loses focus on the most relevant information that can be retrieved from the text.

A possible solution to this problem, that could be investigated in further researches, is the enhancement of the summaries production system with a clustering-based method that aggregates similar information from sub-events and indicates the most relevant ones before the emission.

Acknowledgements

ACKNOWLEDGMENTS

Everybody wants to know what I'd do if I didn't win.
I guess we'll never know.

Table of Contents

List of Tables

List of Figures

Acronyms

AI

Artificial Intelligence

TS

Temporal Summarization

Chapter 1

Context description

1.1 From the printing press invention to online newspapers

The diffusion of newspapers during the 17^{th} century brought an inestimable value to society in terms of knowledge, participation and awareness: the chance to get to know the latest news (first weekly, then daily) has allowed the majority of the population - including the lowest social classes - to be aware of the financial and political happenings of their country. Therefore, the printing press invention led to a radical change in society and to the widening of social and political debate, even in the poorest social environments. It would have taken over two centuries to dethrone the monopoly of printed papers and to introduce alternative forms of news communication: the radio (1902) and - consequently - the television (1927) opened up to a faster, more efficient and up-to-date method of communication. These types of news broadcasting gave the possibility to stay updated on the evolution of a specific news several times in a day, unlike the daily cadence to which people was accustomed through the reading of newspapers.

In this scenario, especially when the television was fully adopted by the population, printed papers lost more and more their function of news issuers and focused their work on enriching their articles with comments from literati, in-depth studies and insights. Moreover, the large number of news that a newscast has to broadcast, in conjunction with the strict schedule of the television, does not allow deepening the news: for this reason, the newspaper remained the best way to digest news.

In the last decades, through the incredible diffusion of the World Wide Web, the world faced another radical shift in the way we receive, elaborate and digest news. Nowadays news are not only emitted by newspapers, radios and televisions: in fact, innumerable sources of information, more or less authoritarian, have been emerged.

Social networks, online newspapers, freelance journalists, bloggers and many

other actors contribute to enrich the data lake of news accessible through the World Wide Web. In 2016, 62% of adults in US declared to access their news on social media [1]: this percentage increased more and more over the following 4 years, becoming 68% in 2018 [2] and 71% in 2020 [3]. More generally, in 2020 it is estimated that the 86% of U.S. adults get news from digital devices (70% of them do it on a daily basis) [4].

Concurrently with the adoption of digital services as their principal source of information by the majority of the population, printed newspapers are facing an unstoppable decline. Since 1984, the total estimated circulation of daily newspapers in the U.S. is decreasing and, since 2004, the decreasing rate is even higher [5] (Figure 1.1).

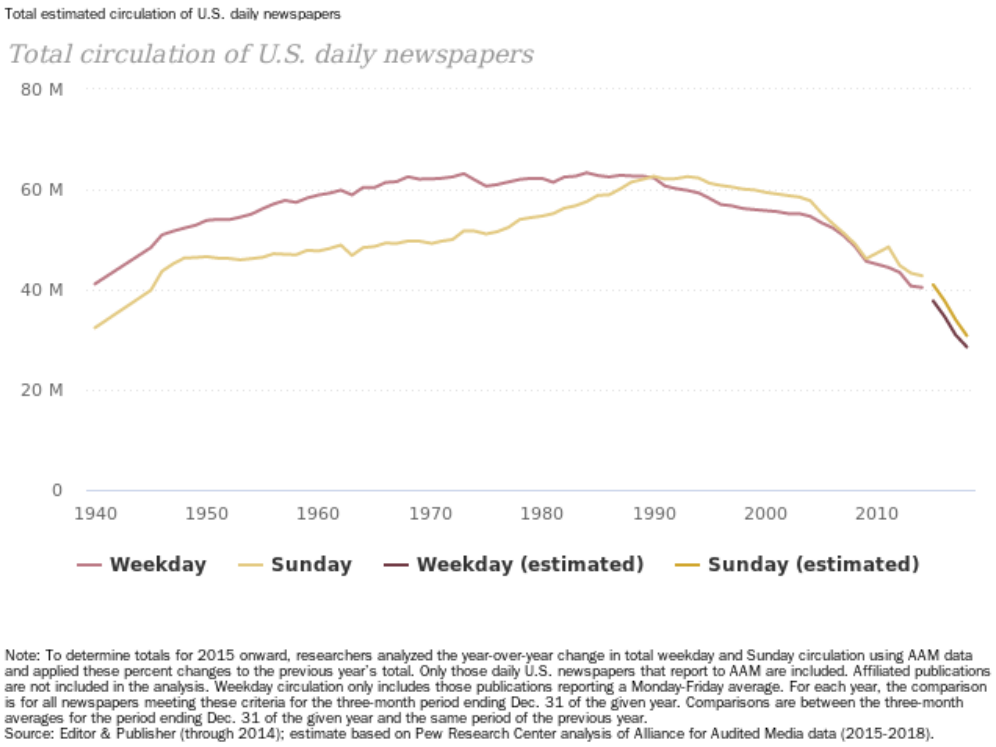

Figure 1.1: Total circulation of U.S. daily newspapers

1.2 The big data era: lots of news articles, lots of confusion

Thanks to the rising of the World Wide Web and the increasing computational power of IT devices, in the 21^{th} century an exponentially growing amount of data

(Figure 1.2) is being collected, processed and analyzed.

Among all those data - of different type, content and format - a significant percentage is textual: tweets, Facebook status updates, news articles, blog posts and so on. In 2015, every 60 seconds, 98.000 tweets and 670.000 Facebook posts were produced, and the total amount of data generated every minute was 1.820 TB [6].

According to the Internet Live Stats website, in 2021, 7.5 million posts are uploaded per day on the World Wide Web. Furthermore, according to different surveys, 60 authors out of 100 might reuse content from previous blog posts several times, leading to longer, not targeted news articles [7].

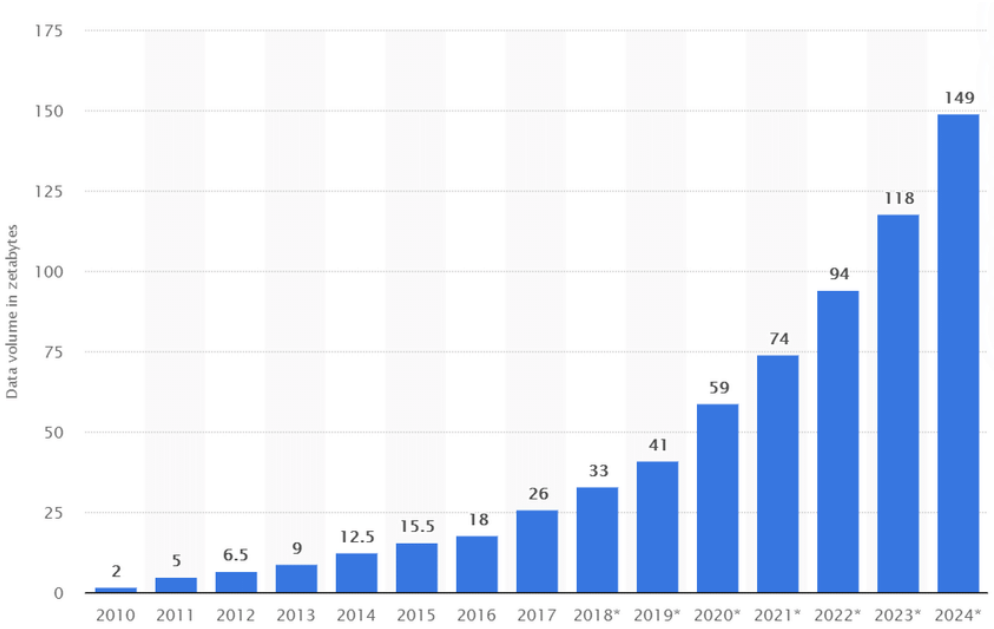

Figure 1.2: Volume of data created worldwide from 2010

The large amount of data produced through online articles suffers from redundancy problems and inaccuracy. Recent studies [3] have shown that most Americans think online news articles are largely inaccurate and that the news they get access to through social medias haven't improved their knowledge about the topic examined [3]. Only the 29% of the U.S. population benefited of news articles from social medias, and the remaining 23%, instead, accused to be even more confused after consulting them.

1.3 Temporal Summarization systems: timely sentence-length updates emission based on novelty

The abundance of online newspapers (and, consequently, the abundance of articles issued), brought us to an indefinite and messy amount of digital documents. Even if at first glance it might seem we have an infinite potential of learning, we soon realize that - to take full advantage of them - we should have the ability to read, understand and digest all the documents concerning a specific news.

As humans, we cannot process massive amounts of information in limited periods: however, thanks to the advancements of hardware and the introduction of cutting-edge learning algorithms, we are now able to teach machines how to learn relationships from data by feeding them with thousands (or millions) of labeled examples.

Natural Language Processing - a sub-field of linguistics, computer science and Artificial Intelligence [8] - encompasses several topics concerning computational processing and understanding of the human language [9]. The research community proposed lots of different architectures - such as Recurrent Neural Networks [10] [11], Long-Short Term Memory Networks [12] and Transformers [13] - through which hundreds of Natural Language Processing tasks are addressed.

In particular, the advent of the aforementioned architectures has allowed to produce more and more efficient Text Summarizers - in charge of coherently condensing pieces of a textual corpus into a shorter, easily digestible text.

Still, Text Summarizers have an impact whereas a corpus does not change over time and is collected in its definitive form; instead, in situations in which the corpus grows rapidly along with an evolving event, traditional Text Summarization approaches do not fit properly.

As part of language-related tasks, the Temporal Summarization task (TS), instead, aims to develop "systems which can detect useful, new, and timely sentence-length updates about a developing event to return to the user" [14]. In particular, the typical developing event is an unexpected news (for instance, a natural disaster or a mass protest) whose new information emerges rapidly.

In this work, state-of-the-art Deep Learning architectures are exploited to create a system that can automatically scan large sets of news articles, highlight the presence of novelties - with respect to the already known information - and produce concise update summaries with low latency.

Chapter 2

Temporal Summarization

2.1 TREC-TS Track Overview

As anticipated in the Introduction chapter, this work proposes a Temporal Summarization system able to emit relevant, concise and up-to-date sentences regarding an unforeseen event.

This work is highly inspired by a prior set of challenges proposed by the TREC conference that, in 2013, 2014 and 2015, came up with the Temporal Summarization track - among many other different challenges.

The TREC Temporal Summarization track (from here on, we will refer to it as TREC-TS) requests to develop a system that

1. can emit concise and pertinent updates about an ongoing event,

2. can track the metrics of important event's attributes, such as the number of deaths.

These goals are respectively named "Sequential Update Summarization" and "Value Tracking".

The **Sequential Update Summarization** sub-task proposed in the TREC-TS track has the aim of developing a system that, receiving in input a time-ordered corpus and a query, iterates over the documents in temporal order and decides whether to include or not a document into an update summary.

The **Value Tracking** sub-task, instead, requests to develop a system that, receiving in input a time-ordered corpus, a query and an attribute of interest, emits proper estimations of the attribute value for an event.

2.2 TREC-TS Track Metrics: Sequential Update Summarization

The evaluation metrics proposed within the TREC-TS Sequential Update Summarization sub-track rely on the 'nuggets' (or sub-events) definition and measure the system ability to cover these nuggets with low-latency.

Substantially, a 'nugget' (or sub-event) is a very short phrase / a keyword that highlights a specific information. The information highlighted could be a date, a place, or more generally an important sub-event that has to be captured. Each document within the provided corpus covers 0 to N nuggets.

Indeed, the evaluation metrics aim to measure precision, recall, timeliness and novelty of the documents selected by the system. In order to estimate these metrics, various auxiliary functions are defined - such as

- the **nugget relevance**, for measuring the importance of the nugget on a 0-3 scale.

  The nugget relevance function could be

  - **graded**: with graded relevance, it is intended that the relevance is normalized on an exponential scale.

    The actual function is defined as follows:

$$\mathbf{R}_g(n) = \frac{\exp n.i}{\exp \max_{n' \in \mathcal{N}} n'.i} \tag{2.1}$$

  - **binary**: with binary relevance, the relevance scale is restricted to the distinct values "not important" / "important".

    The actual function is defined as follows:

$$\mathbf{R}_b(n) = \begin{cases} 1, & \text{iff } n.i > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

- the **latency discount**, for measuring the update emission timeliness.

  The latency discount function is a monotonically decreasing function of $u.t, n.t$, where

  - $u.t$ is the moment in time in which the system decided to emit the update
  - $n.t$ is the moment in time in which the nugget became available

  The actual function is defined as follows:

$$\mathbf{L}(u.t, n.t) = 1 - \frac{2}{\pi} \arctan(\frac{u.t - n.t}{\alpha}) \tag{2.3}$$

where

$$\alpha = 3600 * 6 \tag{2.4}$$

is the latency-step (6 hours).

- the **verbosity normalization**, for penalizing unreasonably long updates emitted.

  The verbosity normalization function is a string length penalty function that monotonically increases with the number of words contained into an issued update.

  The actual function is defined as follows:

$$\mathbf{V}(u) = 1 + \frac{|\texttt{all\_words}_u| - |\texttt{nugget\_matching\_words}_u|}{\texttt{AVG}_n|\texttt{words}_n|} \tag{2.5}$$

  where u is the update and n is the nugget.

- the **update-nugget matching**, a key earliest matching function between a set of updates and a nugget that returns the first update that covers a specific nugget.

  The actual function is defined as follows:

$$\mathbf{M}(n, \mathcal{S}) = \underset{\{u \in \mathcal{S}: n \approx u\}}{\arg\min}\ u.t \tag{2.6}$$

  that could be $\emptyset$ if no update u matches n.

By means of the foregoing functions, several interesting measures of interest are defined. In particular, the TREC-TS track's official metric is the so-called **Combined**, which is the harmonic mean of **Expected Gain** and **Comprehensiveness**:

$$\texttt{Combined}(S) = 2 * \frac{\mathbf{C}(S) * \mathbf{G}(S)}{\mathbf{C}(S) + \mathbf{G}(S)} \tag{2.7}$$

where $\mathbf{C}(S)$ and $\mathbf{G}(S)$ are, respectively, the **Comprehensiveness** and **Expected Gain** metrics.

The **Expected Gain** metric is the sum of the relevance of each nugget matched by an update, defined as follows:

$$\texttt{ExpectedGain}(\mathcal{S}) = \mathbf{G}(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{u \in \mathcal{S}} \sum_{n \in \mathbf{M}(u)} \mathbf{g}(u, n) \tag{2.8}$$

where

- $\mathbf{M}(u)$ is the set of nuggets matching the update u,
- $\mathbf{g}(u, n)$ is the gain function (that could be discount-free or latency-discounted)

The **Comprehensiveness** metric is the proportions of nuggets matched by the emitted updates, defined as follows:

$$\texttt{Comprehensiveness}(\mathcal{S}) = \mathbf{C}(\mathcal{S}) = \frac{1}{|\mathcal{N}|} \sum_{u \in \mathcal{S}} \sum_{n \in \mathbf{M}(u)} \mathbf{g}(u, n) \tag{2.9}$$

where

- $\mathbf{M}(u)$ is the set of nuggets matching the update u,
- $\mathbf{g}(u, n)$ is the gain function (that could be discount-free or latency-discounted),
- $\mathcal{N}$ is the set of nuggets for the current event

In order to properly compare the performances of the Temporal Summarization system proposed in this work, the comparisons will be made on both the official leaderboards released by the TREC conference (for each year: 2013, 2014 and 2015) and the paper "Explicit Diversification of Event Aspects for Temporal Summarization" [15] (McCradie et al., 2018), a McCradie's more recent exceptional work in which several methods are discussed and evaluated.

2.3 Differences with the TREC-TS Track

In the previous section the TREC Temporal Summarization Track was introduced and discussed, along with official evaluation metrics proposed by the organizers themselves; in particular, in the introduction we focused on the Sequential Update Summarization sub-task proposed in the conferences, in which from a time-ordered corpus the goal is to develop a system able to decide whether to include or not a document from the corpus in an output update summary.

The purpose of this work takes huge inspiration from the Sequential Update Summarization sub-task and enriches it with further details: in particular, the task addressed in this work aims to develop a system able to automatically process a

textual time-ordered corpus (related to a specific event) and emit concise updates to the end users by selecting *portions* of the documents considered relevant.

Indeed, the most important difference is that the original TREC-TS challenge requests to "just" select *entire* documents from the corpus, whilst the system proposed in this work aims to select *relevant portions* from the documents, exploited by means of a text generation model that produces freshly paraphrased sentences to emit.

More generally, we can think of the TREC-TS Track as a Textual Information Retrieval task, whilst this work addresses an actual Abstractive Text Summarization task.

Despite these differences, the system proposed in this work is still evaluable through the official TREC-TS metrics; at the end of this work (see section 7, Conclusion) the results of this system with respect to the TREC-TS metrics are presented, along with a comparison with the other systems proposed solely for the TREC-TS challenge.

2.4 Differences with the Timeline Summarization task

As a note, it is worth mentioning the differences between the Temporal Summarization task and the Timeline Summarization task, since they are both similar in name and purposes.

The Timeline Summarization task (TLS) has the purpose of overviewing long-running events by producing summaries regarding the major sub-events for each key date found [16] [17].

As mentioned several times, instead, the Temporal Summarization task (TS) has the purpose of producing concise, relevant and up-to-date update summaries regarding an evolving event [14].

Both the TLS and TS tasks aim at producing summaries of long-lasting events; however, the Temporal Summarization task has the purpose of performing the analysis **online**, as soon as the new documents are indexed. Instead, the Timeline Summarization task aims at extracting timestamped summaries by analyzing the whole event.

Chapter 3

Literature review

3.1 Previous works

Temporal Summarization is a wide-range task that requires knowledge from various fields of Machine Learning, Computer Science and Linguistics. In this section, major contributions in the field of Text Summarization and Temporal Summarization are introduced, along with the presentation of state of the art architectures involved in Natural Language Processing tasks.

3.1.1 Text summarization

The exponentially increasing amount of texts produced over the last decades, along with the necessity of capturing information and extracting main concepts from unsorted and chaotic textual sources, has demanded researchers developing methods and approaches for addressing the task of automatic text summarization.

Hence, engineer procedures able to summarize documents while keeping the overall content and information unaffected is the main purpose of this task [18].

In the field of text summarization, it is possible to distinguish two major approaches: we say that a summarization system applies **extractive** approaches when a text is restricted to a subset of its sentences, leaving out phrases (or portions of text) not considered useful or informative for the summary. Instead, a summarization system applies **abstractive** approaches when a text is someway paraphrased into a summary, shorter than the source, having same notions and concepts.

Furthermore, it is possible to distinguish summarization systems from the number of sources: a summarization system that produces a summary from a single source (or document) performs the so-called "**single-document summarization**". Instead, a summarization system that produces a summary from different sources (or documents) performs the so-called "**multi-document summarization**".

In the field of **extractive text summarization**, several methods based on statistics and / or humanly-engineered features can be found in the literature. Such systems have the aim of enlighting salient sentences from source document(s) relying on various sentence score mechanisms, that enable the possibility of approaching this task through expert rule systems. Some of these methods relied on frequency-based procedures [19][20] (Luhn, 1958; Nenkova, 2006) or on the TF-IDF (Term Frequency - Inverse Document Frequency) procedure [21] (Christian et al., 2016); other methods, instead, are based on graph-based algorithms [22] (Kupiec et al., 1995), hidden Markov models for computing the likelihood of a sentence to be relevant for the summary [23] (Conroy et al., 2006), ILP (Integer Linear Programming) models operating on trees and graphs constructed from the source text [24] (Woodsend et al., 2010) or operating on discourse tree based on the Tree Knapsack Problem [25] (Hirao et al., 2013), and LSA (Latent Semantic Analysis) based methods for selecting highly ranked sentences over a document [26] [27] (Gong et al., 2001; John et al., 2017). Further summarization methods based on ontology features of the sentences can be also found in the literature [28] [29] [30] (Chen et al., 2006; Baralis et al., 2013; Hennig et al., 2016).

Over the last years, as for the majority of the other Natural Language Processing related tasks, the research studies focused on Neural Network based methods: the advent of Recurrent Networks [10] [11], LSTM-based architectures [12] and the Transformer architecture [13] provided tools able to model interactions between spans of text, extract informations from them and construct features not humanly engineered.

Recursive Networks and LSTM-based models have been successfully applied in various text summarization related tasks, like sentence ranking [31] (Cao et al., 2015), sentences and words extraction [32] [33] [34] (Cheng and Lapata, 2016; Nallapati et al., 2016; Xu and Durett, 2019), syntactic and discourse parsing [35] [36] (Socher et al., 2011; Li et al., 2014). Moreover, hierarchical bidirectional LSTM models recently resulted to outperform CNN and RNN approaches for extractive text summarization [37] (Jiang et al., 2021).

In the last lustrum, also application of Graph Neural Networks for extractive text summarization arised: Yasunaga et al. (2017) proposed a three stages approach for extracting summaries by means of RNN-based sentence embeddings, Graph Convolutional Networks and a greedy sentence selection heuristic [38]. Cui et al. (2020) and Wang et al. (2020) also proposed two approaches based on Graph Neural Networks enriched with methods for capturing inter-sentence relationships [39] [40].

Also worthy of mention are the works in the field of extractive text summarization via Reinforcement Learning: Mohsen et al. (2020), similarly to Narayan et al. (2018), proposed a self-attentive reinforced neural network-based model trained directly by optimizing the ROUGE evaluation metric of the sentences extracted

[41] [42].

Abstractive text summarization systems, unlike extractive systems, are considered more challenging to be built because of different reasons. An extractive system aims only to select relevant sentences from the source text; abstractive systems, instead, cope with problems like text generation which are harder than straightforward data-driven approaches like sentence extraction. Due to this, early approaches to text summarization were mostly extractive-like; instead, over the last years, the rise of more complex architectures enriched the researchers' toolkit and enabled the possibility of building quite reliable abstractive text summarizers. Intuitively, the overall goal of a learner that performs abstractive summarization is to digest text documents, learn the semantic representations behind them, and be able to - through the semantic representations of the important content of the text - generate a whole new text, shorter than the source, having same meaning and important concepts.

First abstractive text summarization researches were based on Abstraction Schemes [43] (Genest and Lapalme, 2012), tree-based methods [44] (Tanaka et al., 2009) and information-items extraction methods [45] (Genest and Lapalme, 2011).

Further initial stages approaches can be classified as sentence fusion and aggregation methods, in which plausible summary sentences are derived from the combination of different subtrees [46] [47] (Filippova and Strube, 2008; Cheung and Penn, 2013). Enhancements to this technique were proposed by Bing et al. [48] (2015).

In 2015, Rush et al. [49] - relying on the Bahdanau et al. attention mechanism [50], which already resulted to be successful in tasks like machine translation - proposed a neural attention model (convolutional encoder, feed-forward decoder) where every word to be added in the summary was conditioned on the original sentence through the attention mechanism. Chopra et al. [51] (2016) enhanced this architecture by using a Recurrent Neural Network instead of a feed-forward network for the decoder, leading to state-of-the-art results on the Gigaword and DUC datasets.

Temporal Summarization

The Sequential Update Summarization (see section 2.1) task proposed by the TREC conference can also be seen as a sentence selection / extractive text summarization task.

Several methods have been proposed during the years in which the TREC-TS challenge took place, exploiting keywords mining approaches through Latent Dirichlet Allocation (LDA) based procedures [52] (Tazibt and Aoughlis, 2019), clustering methods [53] (Liu et al.) and manifold-based text similarity algorithms [54] (Zhao et al.), along with sentence scoring procedures by means of the keywords

shooting method[55] (Zhang et al.), convex combinations of relevance, novelty and saliency features of the text [56] (Xu et al.), ranking methods on aggregated data [54] (Zhao et al.) and entity recognition based models [57] (Abbes et al.).

Furthermore, in 2018, McCradie et al. [15] proposed a system that relies on previous works in the field of Web search diversification [58], by proposing an approach that selects sentences depending on a predefined taxonomy that describes the most useful information retrievable from the type of event in analysis, along with the predicted relevancy of the n-th update.

The framework consists of two components that (1) automatically identifies the types of informations that could be relevant for a particular query (extracted from Wikipedia or generated via Crowdsourcing) and (2) estimates the score for each update analysed by means of the state-of-the-art explicit diversification framework xQuAD [59].

The aforementioned work - as anticipated - does not relies solely on the data provided by the TREC conference; still, it is evaluated on the TREC metrics (see section 2.2) and it is used as a comparison with respect to this work in chapter 6.

3.1.2 Transformers

Transformer [13] (Vaswani et al., 2017) is a cutting-edge deep architecture, widely adopted for addressing Natural Language Processing, Speech Processing and Computer Vision tasks. Originally, Transformer was introduced as Seq2Seq model for machine translation; yet, the rise of Transformer-based pretrained models like BERT [60] showed that it is possible to achieve state-of-the-art results on a wide range of tasks. For this reason, nowadays, Transformer-based pretrained models are the go-to architectures for several Machine Learning tasks.

Transformer Vanilla

The Vanilla Transformer [13] architecture is a Seq2Seq model consisting of a encoder-decoder structure. Both the encoder and the decoder are actually stacks of N identical encoder / decoder blocks, that slightly differ between each other because of additional cross-attention modules (in the decoder blocks).

The Vanilla Transformer uses $N = 6$ encoder blocks, each of them composed of a multi-head self-attention mechanism and a position-wise FFN (fully connected feed-forward network); moreover, throughout each sub-layer, a residual connection is implemented. Finally, the output of each sub-layer is $\texttt{NormalizationLayer}(\mathbf{x} + \texttt{SubLayer}(\mathbf{x}))$, where $\mathbf{x}$ is the input sequence. Both embedding and sub-layers build outputs of the dimension $D_m = 512$.

Similarly, the Vanilla Transformer uses $N = 6$ decoder blocks. The decoder's structure differs (from the encoder's structure) because of the application of an

additional multi-head attention layer placed at the very end of the encoder block. For every position, the multi-head attention layer masks all the following positions - indeed, the prediction for step j depends only on the $[0, j-1]$ positions.

The key element introduced with the Transformer architecture is the **attention mechanism** by means of the Query-Key-Value (QKV) model: Transformer parses the (input) word vectors[1] into key, query and value vectors and applies the so-called scaled dot-product attention, defined as

$$\texttt{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \texttt{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{D_k}})\mathbf{V} = \mathbf{A}\mathbf{V} \tag{3.1}$$

where

- $\mathbf{Q}$ is the matrix representation of the queries
- $\mathbf{K}$ is the matrix representation of the keys
- $\mathbf{V}$ is the matrix representation of the values
- D_k is the dimension of the keys

and

$$\mathbf{A} = \texttt{Attention Matrix} = \texttt{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{D_k}}) \tag{3.2}$$

$\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ are the result of the multiplication between input $\mathbf{X}$ and a set of learned matrices $\mathbf{W_Q}$, $\mathbf{W_K}$ and $\mathbf{W_V}$; obviously, hence, the dimensions of $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ are controlled by means of the dimensions of $\mathbf{W_Q}$, $\mathbf{W_K}$ and $\mathbf{W_V}$. Summing up, matrices $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ are weighted representations of the input $\mathbf{X}$.

The Vanilla Transformer instead of applying a single attention function uses the so-called multi-head attention, which is a manner for projecting queries, keys and values into higher dimensions learning different attention heads.

An attention head is essentially the output of a scaled dot-product attention (Equation 3.1); several attention heads are needed for capture information from several representation subspaces at different positions.

As anticipated before, both encoder and decoder blocks exploit a position-wise fully connected feed-forward network - placed right after the multi-head attention. The fully connected feed-forward network consists of two linear transformations:

[1]Obviously, mathematical operations cannot be applied on textual data: indeed, the Transformer architecture defines the **word vectors** as the embedded representation of each element of a sentence; such word embeddings (or tokens) encapsulate the semantic meaning of a word in a format that is understandable by Neural Networks and are used to feed the Transformer.

$$\texttt{FNN}(x) = \max(0, x \cdot W_1 + b_1) \cdot W_2 + b_2 \tag{3.3}$$

where

$$\max(0, x \cdot W_1 + b_1) \tag{3.4}$$

is nothing but a ReLU activation.

Finally, even though the Vanilla Transformer does not work by processing sequentially the inputs, it achieves the same result by enriching the input with a positional encoding - a vector containing values that provide information about the position of an element in the input:

$$\texttt{PE}(\texttt{pos}, 2i) = \sin(\texttt{pos}(10000^{\frac{i}{D_m}})) \tag{3.5}$$

The advantage of the Transformer architecture is that self-attention layer increase interpretability and can potentially understand relationships between elements that are far from each other. Furthermore, self-attention layers are faster than recurrent ones in cases in which the sequence length is smaller than the representation dimensionality.

BERT, RoBERTa and SpanBERT

Since the release of the Vanilla Transformer [13] (Vaswani et al., 2017), lots of transformer-based models were proposed. In particular, the BERT (Bidirectional Encoder Representations from Transformers) model [60] (Google Research, 2019) obtained state-of-the-art results on eleven Natural Language Processing tasks. BERT [60] was pretrained on 3.3 billion unlabeled texts and was designed to be finetuned adding an additional output layer, without task-specific architecture modifications.

Since its release, BERT has become an ubiquitous baseline in Natural Language Processing experiments, with more than 150 publications that improve the model [61] [62]. Two of the latter, along with BERT itself, are used in this work: RoBERTa [63] (Meta AI, 2019) - a "robustly optimized method for pre-training self-supervised NLP systems" - and SpanBERT [64] - a "pre-training method designed to better represent and predict spans of text" - were evaluated through multiple experiments.

BERT, as the name states, is a *Bidirectional Transformer*. In fact, it enhances the previous standard language models by proposing the "masked language model" (MLM) pretraining objective, which allows the input representations (a token sequence) to fuse the left and right context.

The MLM objective, substantially, forces BERT - given a partially masked input sequence - to reconstruct the input sequence. Indeed, BERT learns how to fill an incomplete text sequence with the correct tokens. It is worth mentioning that

the masked input sequences are defined once in the data processing phase (single static mask): hence, from a single input sequence, a single input masked sequence is created - for each sequence.

Furthermore, BERT is also pretrained on the Next Sequence Prediction task. Simply, BERT is fed with two sequences A and B: half the times, B is a random sequence sampled from the corpus. The other times, instead, B is the actual following sequence of A. The goal of BERT was to binary classify the inputs (and, in particular, B) as "NextSequence" or "NotNextSequence": results state that BERT reaches 97% of accuracy on the Next Sequence Prediction task.

As anticipated, when released, BERT achieved state-of-the-art results on eleven NLP tasks. In particular, BERT pushed forward the performances on

- the SQuAD v1.1 Question Answering Test F1, with 1.5 absolute improvement
- the Multi-Genre Natural Language Inference (MultiNLI), with 5.6% absolute accuracy improvement
- the GLUE benchmark to 80.4%, with a 7.6% absolute improvement.

RoBERTa [63], which stands for "Robustly Optimized BERT Pre-training Approach", is one of the most popular BERT-derived model that has been proposed in order to further enhance the performances of BERT itself.

Firstly, differently from BERT, RoBERTa is not trained on single static masks; instead, the training data used to feed RoBERTa is duplicated 10 times, each with a unique masking strategy.

Furthermore, the NSP (Next Sequence Prediction) task was removed since after several experiments it resulted that the model had slightly greater performances.

Moreover, RoBERTa was trained on bigger batch sizes and longer training sequences, increasing the difficulty of the Masked Language Model objective.

Finally, RoBERTa achieved higher scores on both the GLUE benchmark (88.5) and the RACE benchmark.

In 2020, Facebook Research also proposed SpanBERT [64] - a pretraining method designed to better predict spans of text.

Again, the masking strategy proposed by the authors is different with respect to the single (randomly generated) static mask introduced by BERT. In particular, SpanBERT is pretrained by masking contiguous random subsequent tokens (i.e., a span) instead of randomly chosen tokens. Moreover, SpanBERT was also trained to predict the entire masked subsequence by means of solely the span boundary representations.

SpanBERT achieved terrific results particularly on span selection tasks (such as Question Answering) and also on the GLUE benchmark.

In particular, the SpanBERT model obtains 94.6% on SQuAD 1.1 and 88.7% on SQuAD 2.0, along with state-of-the-art performances on the OntoNotes coreference resolution task.

Chapter 4

Methods

In this chapter the task of the Temporal Summarization addressed in this work will be explained in details, along with the tools, methodologies and procedures that have been applied to carry out this work.

4.1 Overview

4.1.1 Temporal Summarization

The pipeline proposed in this work for addressing the Temporal Summarization task consists of 3 major stages. Following, a brief introduction to the workflow, in order to give a general flavour of what will be discussed in the following sections:

- **Data processing stage**. In this phase, the data is collected and processed offline by means of some parameters. Through the variation of these parameters, a unique dataset is created (more details in section 4.2.2). Hence, the output of the processing phase are several datasets, unique among them, that are directly uploaded to an Amazon S3 bucket.
- **Models finetuning** (training phase). In this phase, BERT-derived pretrained models are finetuned in order to learn how to extract relevant portions of text from the update texts. The data used to finetune the model is downloaded from the Amazon S3 bucket, filled up previously. Once the finetuning phase is done, the model is saved and uploaded into another Amazon S3 bucket.
- **Summaries production** (inference phase). In this phase, a finetuned BERT-derived model is downloaded from an Amazon S3 bucket, along with the test data. The model is then fed with the test data, sorted cronologically in order to simulate the emission of updated news articles. The outputs of the model are relevant portions of the texts received, which are consequently passed

through a paraphrasing pretrained model and ensembled to produce an update summary.

4.1.2 Tools, integrations and services

For the development of this project, several tools, cloud services and packages were adopted. Briefly,

- GitHub was used during the development phase of this work, in order to do code-versioning.
- Google Drive was used as a cloud storage for the raw data, especially because of its integrations with Google Colab.
- Google Colab was used to develop interactive high level notebooks.
- The Huggingface library provided pretrained Transformers, and in particular:
  - The pretrained versions of BERT, RoBERTa and SpanBERT were supplied and successively finetuned for the Token Classification task. Moreover, in order to work with these models, also the pretrained versions of their Tokenizers were provided. More details on Token Classification and Tokenizers later.
  - The pretrained version of Pegasus and its Tokenizer were supplied and applied for the paraphrasing phase of the work.
- Amazon S3 was used as a cloud storage for the processed data and for the finetuned models (BERT, RoBERTa and SpanBERT).
- Weight & Biases was integrated for the purpose of logging, visualization and reporting.

4.1.3 Token Classification task

As anticipated, the core of the development orbits around the finetuning of BERT, RoBERTa and SpanBERT for the task of Token Classification. Hence, it is needed to clarify what is the purpose of this task. To do so, it is also useful to explain the behaviour of the Tokenizers.

A **Tokenizer** is in charge of preparing the inputs for the model, since it cannot digest standard textual data. In order to prepare the inputs, the Tokenizer splits the text into words and subwords, consequently mapped into a list of tokens - i.e., a list of integers that can be properly recognized by the model (see Figure 4.1).

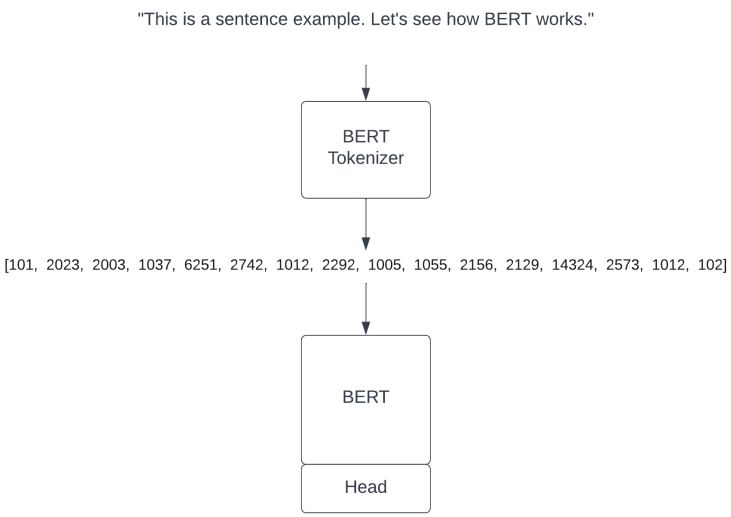

Figure 4.1: Tokenization process

A Tokenizer is strictly paired with a BERT-derived model. In this sense, BERT works correctly only with its own Tokenizer; the same for RoBERTa and SpanBERT.

The reason why a Tokenizer is bound to a particular model resides in the fact that BERT and its derived - as anticipated in the previous chapter - are pre-trained models and the Tokenizer is pretrained jointly with the model. Furthermore, they are not actually trained on a single task - instead, they are pre-trained with the purpose of learning a language representation.

Hence, when a pre-trained BERT is fed with a list of tokens, it can refer to its previously learnt mapped vectors and provide features to any head applied over it. Obviously, the tokens must came from the correct Tokenizer: otherwise, they could not be correctly interpretated by the model and hence lead to poor results.

The power of such models can be exploited all over several Natural Language Processing tasks, since they provide a solid backbone (also called features extractor, the first part of a Neural Network, where the features of the data are captured) able to provide effective language representations. Consequently, depending on the task that has to be addressed, a specific head (i.e., the last part of a Neural Network, carefully designed depending on the shape requested for the output) can be applied on top of the pretrained model. This process is certainly less time-expensive than training a custom model from scratch and, more importantly, is very likely to result better in performances.

In particular, lately, several researches were conducted on applications of fine-tuned BERT [60] [65] (and in general, transformers models) for **Token Classification** tasks, in which a label is assigned to each token. Popular examples of Token Classification sub-tasks are Named Entity Recognition (NER) and Part-of-Speech (POS) tagging: for instance, in the NER task, each token is labeled depending on its grammatical nature - e.g. a token represents a noun, a verb, etc.

BERT resulted to perform incredibly well in both Named Entity Recognition [66] [67] [68] [69] [70] and POS tagging [71] [72] [73], overcoming state-of-the-art results on several datasets and benchmarks.

Having stated how outstanding BERT is in addressing Token Classification tasks and how convenient is to finetune a cutting-edge model with respect to train a newly defined one from scratch, in this work the problem of "enlighting relevant spans from a text" is modeled as a Token Classification task, in which we would like to label the tokens as not relevant / relevant (respectively, with zero / not zero labels): Hence, in the data processing phase, the data is treated to relate a list of tokens to a mask (i.e., the ground truth, a list of labels that enlight the relevant spans). In Figure 4.2, a simple example of Token Classification:

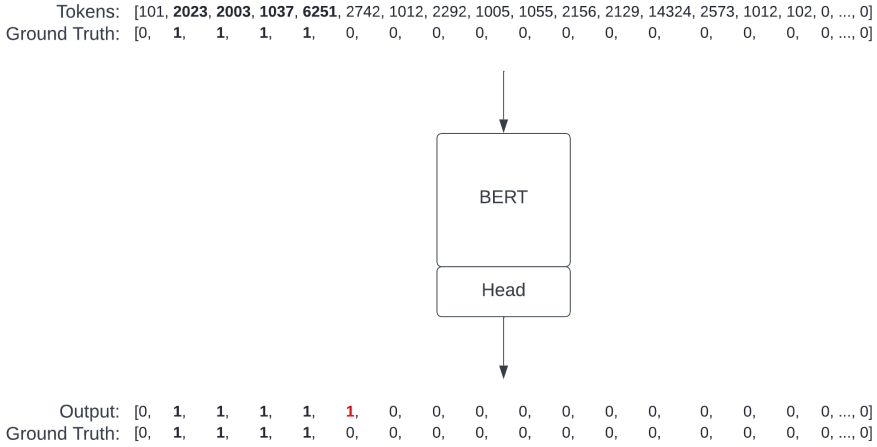

Figure 4.2: Token Classification example

4.2 Data

4.2.1 Data explanation

As for the majority of the Machine Learning / AI projects, the initial phases are mostly related to the data collection, understanding and management.

For this work, the data was downloaded from the TREC organization website [74][14]. In particular, since the Temporal Summarization challenge has taken place in years 2013, 2014 and 2015, the data downloaded refers to those years. For each year, the data is divided into 4 files (three .csv files, one .xml file):

- The **topics** file gives contextual information about the events treated in the dataset, such as a query example (e.g. "buenos aires train crash") and a time interval in which the event remained relevant.

- The **updates** file provides update texts (chunks of news articles) regarding a particular topic.

- The **nuggets** file provides the relationship between a timestamp and a nugget - i.e., a small chunk of text that encapsulate a small but relevant information. This relationship highlights that a given information (the nugget) has been available since the related timestamp.

- The **matches** file relates a nugget with an update text. Hence, it gives the relationship between "which new information is available" (nugget) and "where it can be found" (update text). Furthermore, it specifies the portion of the update text in which the information can be found through two indices.

An update can appear from 0 to N times into the matches table, meaning that not every update contains relevant informations, but some updates can contain more than one relevant information. This means that a single update can cover 0 nuggets, exactly 1 nugget, or N nuggets.

Similarly, a nugget can appear from 0 to M times into the matches table, meaning that not every discoverable information is covered by an update. At the same time, some informations can be found in multiple updates. This means that a nugget can be covered by 0 updates, exactly 1 update, or M updates.

In the following table, there are five examples of records that can be found in the topics table:

query_id	query
11	costa concordia
12	european cold wave
13	queensland floods
14	boston marathon bombing
15	egyptian riots

Regarding the "costa concordia" topic, an example of update is:

"*Most of the 3,200 passengers and 1,023 crew on board the ship had been evacuated to the island of Giglio and from there were being taken to the mainland.*".

Such update text resulted to be related to 2 different nuggets through the matches table. In particular, this is how the portions of the update texts are related to the nuggets:

update text portion	nugget
Most of the 3,200 passengers and 1,023 crew on board the ship had been evacuated	3,206 passengers and 1,023 crew members were on board
evacuated to the island of Giglio and from there were being taken to the mainland	Rescued passengers huddle ashore

However, as specified earlier, not every update contains informations that are relevant for the summarization task. Let's look at some examples regarding the "costa concordia" topic:

update_id	update text
1326952200-...-124	Family Travel
1326502200-...-50	Southeast Asia Pte Ltd.
1326502200-...-51	(Co. Reg. No. 199700735D).

Obviously, such updates do not bring any information regarding the topic. Summarizing, the dataset contains both updates with relevant content and updates that are useless.

The overall goal of the work is to extract - as accurately as possible - relevant spans from pertinent updates. This problem is addressed by means of BERT, RoBERTa and SpanBERT, finetuned for the Token Classification: in this sense, the predictions made by these models will work as masks enlighting the relevant portions only. Their predictions are then used to fed a paraphrasing model that

will actually emit the updates to the end user.

4.2.2 Data preparation

The Data Processing phase is one of the crucial steps that a Machine Learning pipeline has to address, in order to properly serve the correct data to the models that will be trained or finetuned.

As anticipated in the previous section, the BERT-derived models utilized in this work are finetuned for the task of Token Classification: as imaginable, lots of operations must be applied to the data in order to prepare them to fit this task.

Because of these heavy operations, the processing phase was forced to be offline - in the sense that the data is not processed right before a training / inference phase. In fact, a processing phase used to last at least 10 minutes, which is not a reasonable time to wait for. Instead, having a dataset already processed and stored on a cloud service like Amazon S3 is way more convenient.

The data processing phase is conducted by an offline Python script that iterates through the combinations of 4 parameters (as anticipated in section 4.1). In this sense, a unique dataset is produced for each of these combinations - leading to several datasets stored into the Amazon S3 bucket. The parameters are the following:

- The **tokenizer** parameter indicates which tokenizer should be used during the processing phase. Acceptable values for this parameter are the ones related to the models that will be finetuned - BERT, SpanBERT and RoBERTa.
- The **tokenizer type** parameter indicates if the previously selected tokenizer should be instantiated in its "cased" or "uncased" version. It is possible that a tokenizer is available only in one of the two versions.
- The **binary** flag indicates if the ground truth masks should be produced as binary or not-binary lists. Further details are provided in the following sections.
- The **contextual** flag indicates if the tokenized data should be filled up also with contextual informations or not - hence, if the topic of an update must be taken in consideration.

The reason behind the necessity of processing the data with these parameters resides in the fact that

1. The experiments are carried out by means of BERT, RoBERTa and SpanBERT, that - as anticipated - need their own Tokenizer. Hence, during the processing phase, the tokenization is applied and its results are different depending on the **tokenizer** and **tokenizer type** parameters.

2. The experiments are carried out both with binary and not-binary ground truths. In fact, in some experiments the model tries to predict labels having 0 / 1 values; in other experiments, instead, the model tries to predict labels having values in the [0, 4] interval. The meaning behind this differentiation is explained in the following sections.

3. In some experiments, the context (i.e., the topic) is used to enrich the input data for the model.

Despite these varying parameters, there are 2 additional parameters that are fixed:

- The **datasets** parameter is a list that indicates the years for which the data must be processed. It is fixed to ["2013", "2014", "2015"] since we would like to use all the data available to train and test the models.

- The **only relevant** flag is a boolean indicating if the processed dataset must contain (or not) only relevant updates. It is set to True (hence, only relevant updates are used to finetune the model) for different reasons: firstly, by using also not relevant updates the dataset is strongly imbalanced, since the majority of the records are not relevant. Furthermore, managing also not relevant texts is somehow outside of the scope of this project and probably needs even more architectural complexity - in terms of workflow stages and models to be integrated.

The data processing pipeline, conducted by a Python script, follows the upcoming steps:

1. **Data loading phase**:

   (a) The 4 parameters are chosen: [tokenizer, tokenizer type, binary, contextual]

   (b) The raw data, for each year, is loaded from the file system.

   (c) For each year, a single dataset is produced from the 4 source files. Each dataset has columns ['update_text', 'match_start', 'match_end', 'query', 'timestamp', 'relevant'].

   (d) The datasets are merged together, resulting in a single dataset containing all the available data.

2. **Data pre-processing phase**:

   (a) Every row in which "update_text" or "query" are not strings, are deleted.

(b) If using an "uncased" tokenizer, the text is converted to lowercase.

(c) Several regular expressions are applied to the data, in order to fit the tokenizer.

(d) The spans (i.e., the interval of the text considered relevant) are aligned, avoiding referencing internal characters of a word by match_start or match_end.

(e) The binary / not binary masks (the ground truth) are produced.

(f) The data is collapsed: in fact, the same update_text can have several relevant portions. During this phase, the update_text is maintained unique, and the masks are merged together through a boolean OR (in the case of binary masks) or through a more complex operation (in the case of non binary masks).

3. **Tokenization phase**:

   (a) The texts are tokenized through the correct Tokenizer.

   (b) If requested, the context is encoded too.

4. **Data post-processing phase**:

   (a) The first character of a sentence is deleted, if it is a space.

   (b) If the binary flag is False, the masks related to updates having more than one relevant spans are adjusted (if in the mask there is a situation like [0,1,2,2,2,2,2,3,2,2,2,2,3,0], the first 3 is replaced with a 2. This because 3 represents the end of a span: however, here, 2 masks are overlapping.)

5. **Testing phase**: in this phase, the effectiveness of the regular expressions is tested.

Data loading phase

The **data loading** phase is responsible for the actual preparation of the raw data, in terms of loading and application of standard base operations - like merges, base filterings, etc.

As follows, a detailed explanation of the operations:

1. For each dataset requested (i.e., for each year), **the raw data is read** - hence, 4 dataframes (topics, matches, nuggets, updates) are collected.

2. **A simple join operation is performed over the 4 dataframes**: the matches dataframe is joined with the updates dataframe and then with the nuggets dataframe. Recall that the updates dataframe contains every update

text available, and the matches dataframe contains the references only to the relevant update texts. The nuggets dataframe contains the nugget and the timestamp in which the information became public for the first time. In the matches dataframe, nugget and update text are related. By applying the join operation, the resulting dataframe has only relevant update texts (the others are disregarded through the join operation), the relevant interval, the topic and the timestamp in which the information emerged (coming from the nuggets dataframe).

3. Once the previous operation is performed on every requested dataset, **the 3 datasets are merged together**, leading to a single datasets having columns ['update_text', 'match_start', 'match_end', 'query', 'timestamp', 'relevant'].

Data pre-processing phase

The **data pre-processing phase** is responsible for the real processing of the data. Indeed, the most complex and heavy operations are performed in this stage.

The operations carried out in this stage are the following:

1. **The data types of each record for the columns "update_text" and "query" are checked**: if they are not strings, the whole record is deleted.

2. If the Tokenizer requested is "uncased", **the text is converted to lower case**.

3. **Almost 300 regular expressions are applied to the text**.
   The reasons behind this operation reside in the fact that it is necessary to assert that the encode-decode operation is performed consistently for the whole dataset. This means that every text, once encoded through the Tokenizer, must be reconverted correctly to the source text once the predictions on the relevant spans are made. Unfortunately, the Tokenizer very likely fails in this operation - i.e., the decoded tokens do not construct again the original text that produced them. In order to avoid this problem, the regexes are applied in such a way that the Tokenizer is already fed with texts that are equal to the decoded version of them.
   For instance, the phrase *"this is an example of phrase that will be wrongly encoded-decoded !"*, once encoded and decoded, results in the phrase *"this is an example of phrase that will be wrongly encoded - decoded!"*. This is a problem, since we would like to keep the text as the original because of the mapping with the ground truth masks (that could be hence disaligned). By applying these regexes, instead, the original phrase is already reformatted as *"this is an example of phrase that will be wrongly encoded - decoded!"*, leading to no problems after the decoding phase.

4. **The relevant spans indices are aligned to the text**.
   Recall that the relevant spans are highlighted through 2 indices, match_start and match_end, that indicate from which to which character the interesting portion of text can be found. In the dataset, there are cases in which one of these indices points out to internal characters of a word. In this phase, these indices are realigned in order to comprehend the whole portion.

5. **The ground truth masks are produced**.
   Starting from the match_start and match_end indices, the ground truths are calculated. Depending on the input parameter "**binary**", two types of ground truths are returned:

   (a) if the parameter is True, binary masks are returned. In particular, the mask takes 0 value if the i-th word is outside the span. Otherwise, if the i-th is inside the relevant span, it takes value 1.

   (b) if the parameter is False, not-binary masks are returned. In particular, the mask takes (for each i-th element):

   - value 0 if the i-th word is not a word belonging to the span
   - value 1 if the i-th word is the first word belonging to the span
   - value 2 if the i-th word belongs to the span but it is neither the first, nor the last word
   - value 3 if the i-th word is the last word belonging to the span
   - value 4 if the i-th word is the only word belonging to the span

6. **The masks are collapsed**.
   As explained in the previous sections, an update text can have more than 1 relevant span. Hence, in the dataset there could be records with same update text and different masks. During this operation, the masks are treated as follows:

   (a) If the masks are binary, they are collapsed into a single mask by applying a simple boolean OR operation between the masks.

   (b) If the masks are not binary, they are collapsed into a single mask and the overlapping points (cases in which the mask is like [..., 2,**3**,2,2,3,0]) are reconducted to inter-span points (hence leading to [..., 2,**2**,2,2,3,0]).

Tokenization phase

Once the data is prepared, ready to fit the Tokenizer and collapsed, the tokenization process can start.

During the tokenization process, the textual information in the dataset is replaced by the tokenized versions of the texts.

The encoding phase is driven by the

- **tokenizer name**, that identifies the name of the Tokenizer used for the encoding.
- **tokenizer type**, that identifies the type of Tokenizer used for the encoding.

Moreover, if the **contextual** flag is set to True, the context of the text is also encoded: for instance, if the context is *"costa concordia"* and the update text is *"500 deaths in costa concordia disaster"*, the resulting text to be encoded is *"500 deaths in costa concordia disaster"* + Separator + *"costa concordia"*.

Finally, the Tokenizer has a maximum number of words that can consider during the encoding phase: in this work, this parameter is set to 512 words.

Because of the variable number of words that each record can contain, the Tokenizer returns - along with the list of tokens - an attention mask. An attention mask is a binary list, indicating whether a token should be considered (taking value 1) or not (taking value 0). The reason behind the necessity of the attention masks resides in the application of the maximum number of words value: in fact, since the list of tokens are long exactly 512 - i.e., the max number of words - a padding of useless tokens is applied at the end of each list of tokens. Obviously, such tokens should not be taken into consideration by the model: hence, the attention mask in this indices takes 0 value and the model disregard the tokens automatically.

Testing phase

At the end of the actual processing phase of the data, there is also a testing phase in which the encoding-decoding problem is checked: in this phase, the processed dataset is verified by comparing the original texts with the decoded tokens. If every list of tokens can be decoded into the original texts, the test is passed and the dataset is ready to be uploaded to the Amazon S3 bucket.

Chapter 5

Experiments

In the previous chapter the data processing pipeline was deeply explained, starting from the processing parameters that conduct the preparation phase itself to the procedure applied over the data. Some keys takeaways are crucial to be clear before focusing on the finetuning and inference phase: in the processing phase, a set of unique parameters conduct the processing of a dataset. Indeed, by varying those parameters, multiple datasets are produced and uploaded onto an Amazon S3 Bucket. Those datasets lay the basis to an extensive set of experiments in which BERT, SpanBERT and RoBERTa are finetuned in different experimental setups. In fact, each dataset is strictly paired to an unique combination of processing parameters, and each combination of these parameters is strictly paired to an unique experimental setup.

In this chapter we introduce the methodologies behind the training of the Temporal Summarization system developed, that, for the best of our knowledge, is the first system in which BERT-derived pretrained models are finetuned for the task of Temporal Summarization. Moreover, the results of the whole downstream task are presented along with the inference phase, in which the most promising finetuned models are qualitatively evaluated on the full Temporal Summarization task.

Hence, in the following paragraphs we will explore the finetuning phase in details, along with the results of each model in each experimental setup. Moreover, the inference phase - in which summaries are effectively produced - will be presented, along with the choices made for efficiently produce and update the summary.

5.1 Finetuning: Experimental setups

5.1.1 Overview

As anticipated in the previous chapters, this work can be divided into 3 major stages: the data preparation phase, the transformers finetuning phase, and the inference phase.

In this paragraph, we will focus on the finetuning phase: hence, we will inspect every experimental setup in which BERT, SpanBERT and RoBERTa were finetuned for the task of Token Classification.

Every finetuning phase is conducted by means of a worker that prepares the environment for the single experiment through the acquisition of several parameters.

These parameters belong to three different categories:

1. Several **parameters** are **strictly related to the data and to the model chosen**, paired to the ones used during the processing phase, needed for retrieving the correct dataset from the cloud storage and for initialize the correct model.

2. A single parameter is the only **variable hyperparameter** of the model chosen, tuned during different runs of the same experiment.

3. The remaining parameters are **fixed, default hyperparameters**. Some of them are fixed for convenience; meanwhile, other hyperparameters are fixed for resource limits reasons.

Let's dive into the parameters related to model and the data:

- the **model** parameter is paired to the **tokenizer** and **tokenizer type** parameters, by which a dataset is generated. It indicates which model will be finetuned in the current experiment and can take the following values:
  - "bert-uncased", i.e. the uncased version of BERT
  - "bert-cased", i.e. the cased version of BERT
  - "spanbert-cased", i.e. the cased version of SpanBERT
  - "roberta-uncased", i.e. the uncased version of RoBERTa
  - "roberta-cased", i.e. the cased version of RoBERTa

  As explained before, the tokenizer is strictly related to the model, since they are pretrained together. Hence, a combination of **tokenizer** and **tokenizer type** refers uniquely to a single **model**. During the data preparation phase, indeed, we needed to process the data with particular combinations of **tokenizer** and **tokenizer type** in order to be able to finetune a particular **model**.

- the **dataset** parameter indicates which dataset will be used for finetune the model selected. It is fixed to "full", meaning that all the data is used - hence, the processed data related to the TREC-TS competitions of 2013, 2014 and 2015.
- the **full dataset** parameter indicates whether the dataset should contain or not also not relevant updates. It is the inverse of the **only relevant** parameter used during the processing phase. Its default value is set to false.
- the **binary** parameter indicates if the dataset should contain binary ground truths or not. It is paired to the **binary** parameter using during the processing phase.
- the **contextual** parameter indicates whether the dataset should contain or not the context information, appended to the text. It is paired to the **contextual** parameter using the processing phase.
- the **split on topics** parameter conducts the splitting phase of the dataset. As usual in Machine Learning works, the full dataset is divided into training, evaluation and testing sets: if this parameter is set to False, the dataset is randomly splitted according to some defined proportions. Otherwise, if this parameter is set to True, a more reasonable splitting procedure is applied. Let's consider the case in which the dataset is requested to be splitted as 80%/10%/10%: in such a case, the training set will contain the 80% of topics that can be found on the dataset. Hence, the evaluation and testing sets will contain, respectively, the 10% of topics. In such a way, the dataset is not really splitted according to the given proportions; however, this split philosophy is way more reasonable for the type of task addressed in this work, since the model should be tested on topics for which it was not trained on.
- the **save model** parameter, if set to True, triggers the upload of the finetuned model into the Amazon S3 bucket.

Summarizing, through this first part of parameters, the worker will construct an experimental setup in which the **model** will be finetuned on a particular **dataset**. Such **dataset** could be **full** or not, meaning that it could contain or not only relevant updates. Moreover, every row of the **dataset** can be paired with **binary** masks or not; also, every row can be enriched with **context** related information or not. Finally, the **dataset** could be **split on topics** for evaluation reasons, or - instead -, the **model** could be tested on random updates. The finetuned **model** is then uploaded to an Amazon S3 bucket if requested through the **save model** parameter.

Let's now analyze the hyperparameters:

- the **learning rate** parameter is the only hyperparameter left modifiable during a finetuning run. Its default value is 1e-5, which resulted to be a good learning rate value for the three models finetuned; however, since these three models are slightly different from each other, it is possible to tune it for experimentation reasons.
- the **sizes** of the training set, evaluation set and testing set are fixed parameters. Their default values are, respectively, 80%/10%/10%.
- the **batch size** value of the training, evaluation and testing sets are fixed. Their values are, respectively:
  - 2 for the training set
  - 1 for the evaluation set
  - 1 for the testing set

  Such values were fixed because of resources limits, since the GPUs available in Google Colab could not handle larger batches.
- the **epochs** parameter is fixed to 2. However, a single epoch is considered a completed finetuning run.
- the **number of logging steps** indicates after how many training / evaluation steps the worker must log a row into Weight & Biases. The default value for this parameter is set to 100.

The reason behind having a fixed **size** for the training, evaluation and training sets resides in the fact that the full dataset is quite small (12K records circa), especially for finetune huge models like BERT and its derived. For this reason, in order to fed the models with as much data as possible, the training set is the 80% of the total capacity.

Moreover, this reasoning partially explains also the choice of a small default value for the epochs parameter: they last for a not negligible period of time - 4 hours each - but are still enough for accomplish the Token Classification task in a reliable way.

5.1.2 BERT, RoBERTa and SpanBERT for Token Classification

In this section we will analyze every experimental setup that has been tested. In particular, we will focus on the details of the finetuning phase, the reasoning behind the necessity of every setup proposed, the evaluation metrics, along with some expectations about the results of the models.

Finetuning procedure

The **finetuning phase** starts once the worker has collected the parameters needed. Indeed, the worker has already

1. Downloaded the correct dataset from Amazon S3
2. Instantiated a connection with Weight & Biases, where all the logs of the run will be collected
3. Downloaded and initialized a pretrained BERT-derived model for the task of Token Classification (provided by Huggingface)
4. Instantiated an Adam optimizer

Hence, everything is ready for the finetuning of the model.

The finetuning algorithm follows the typical procedure of training of a Neural Network, where for each epoch the training batches are firstly forwarded through the network and then the loss is used for updating the weights through the backpropagation algorithm. A detailed explanation could be found at Algorithm 2.

Token Classification Task: evaluation metrics

The finetuning of BERT, RoBERTa and SpanBERT is the main, more delicate phase of this work, being these models the core of the "relevant text extraction" procedure. The whole Temporal Summarization system rounds around a finetuned BERT-derived model; indeed, without a properly finetuned model the system can only lead to poor overall results. In order to assess the quality of the finetuned models, a custom evaluation metric was defined.

Note that every evaluation metric, except for the losses, are not influent on the actual finetuning of the model - since they do not interfere in the backpropagation step. Hence, every evaluation metric attached has only purpose of providing quantitative results to the system owner, even though they do not provide actually useful insights for the finetuning of the model (unlike the loss, that is crucial for the finetuning).

As introduced in section 4.1.2, a Token Classifier aims at correctly predicting a list of labels, i.e. a list of integers with categorical meaning, against a list of ground truths, i.e. the correct list of integers.

Indeed, at every finetuning step, the model provides a list of predicted labels (for each element inside the batch); moreover, the batch itself already contains the list of correct labels for each element.

In order to extract a quantitative measure that could represent the goodness of the model, the metric used is a modified accuracy score in which the element-wise comparisons between ground truth labels and predicted labels is made only for the

Algorithm 2 Finetuning algorithm

procedure FINETUNING($\mathcal{B}_T, \mathcal{B}_V, n_{ls}$)
  while Epochs E are not finished **do**
    $l_0 \leftarrow 0$ ▷ The loss value is initialized at 0
    $a_0 \leftarrow 0$ ▷ The accuracy value is initialized at 0
    $e_0 \leftarrow 0$ ▷ The number of examples seen is initialized at 0
    $s_0 \leftarrow 0$ ▷ The number of steps made is initialized at 0
    for each Training batch $(b, b_i) \in \mathcal{B}_T$ **do**
      ▷ For each batch b (with batch index b_i).
      $i, m, y \leftarrow$ `UnpackBatch(`b`)`
      ▷ The batch is unpacked.
      ▷ i, m, y are the tokens, attention masks and labels.
      $l, \hat{y} \leftarrow$ `ForwardPass(`i, m, y`)`
      ▷ The batch is passed through the network.
      ▷ Batch loss and predicted values $l, \hat{y}$ are returned.
      $l \leftarrow l + l_0$
      ▷ The batch loss l is added to the epoch loss l_0.
      $s_0 \leftarrow s_0 + 1$
      ▷ The number of steps made s_0 is incremented.
      $e_0 \leftarrow e_0 +$ `b.size()`
      ▷ The number of examples seen e_0 is incremented by batch size.
      $a \leftarrow$ `ComputeAccuracy(`$y, \hat{y}, a$`)`
      ▷ The epoch accuracy a is updated.
      if b_i % $n_{ls} = 0$ **then**
        ▷ If the batch index b_i is divisible by
        ▷ the number of logging steps n_{ls}
        `LogLossValue(`l_0/s_0`)`
        ▷ The current loss value is logged into W&B.
        `LogAccuracyValue(`a_0/s_0`)`
        ▷ The current accuracy value is logged into W&B.
        `EvaluateBatch(`$b, \hat{y}$`)`
        ▷ The batch is qualitatively evaluated
        ▷ by logging the portion of text selected by the model.
      end if
      `BackwardPass(`l`)`
      ▷ The loss is used to finetune the model via backpropagation.
    end for
    `LogLossValue(`l_0/s_0`)`
    ▷ The epoch's loss value is logged into W&B.
    `LogAccuracyValue(`a_0/s_0`)`
    ▷ The epoch's accuracy value is logged into W&B.
    `Validate(`$\mathcal{B}_V$`)`
    ▷ The model is tested on the validation set.
  end while
end procedure

indexes in which the ground truth is not 0. This means that the actual accuracy is calculated by comparing only the ground truth labels that are "active" - in this work, they are the labels referencing an useful portion of text. By doing that, we analyze how many tokens that are relevant are effectively captured and predicted as relevant also by the model.

At a first sight this accuracy could be seen as not completely fair, since only the active labels are compared. Indeed, if the models always predict lists of all ones, the accuracy will be 100%.

However, of course, the models are neither forced nor tempted to predict all ones labels. Moreover, as explained in section 4.2.2, most of the tokens lists used to fed the model in the finetuning phase are strongly padded. In particular, BERT-derived models should be trained (or finetuned) by providing them fixed-size lists of tokens - the maximum length of the tokens lists is 512, which is the chosen capacity of the tokens lists for this work: as imaginable, few texts need such an high number of tokens to be represented; hence, the vast majority of tokens lists in the dataset are highly padded to the right.

For instance, let's consider the source update text
"buenos aires train crash kills 49, injuries over 600".

The BERT-uncased Tokenizer converts this into the following tokens list:
$[101, 9204, 9149, 3345, 5823, 8563, 4749, 1010, 6441, 2058, 5174, 1012, 102, 0, ..., 0]$.

Such tokens list contains 13 not null values - hence, the text can be represented by means of 13 distinct tokens - and 499 null values.

As imaginable, the attention mask of the considered token list is made up of 13 elements set to 1 and 499 elements set to 0:
$[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, ..., 0]$.

Indeed, the models will take care only of the first 13 elements of the tokens list and will not consider the remaining 499. For this reason, we can conclude that it is actually useless to have an evaluation metric that compares element-wise the whole ground truth list with the predicted labels, since the elements we actually care about are concentrated in a small portion at the beginning of the lists.

5.1.3 Finetuning experiments

Once presented the parameters that conduct the finetuning phase, along with a detailed description of the finetuning procedure itself and the evaluation metric that sustain the work, it is possible to start inspecting every finetuning experimental setup in which the models have been applied.

In the previous chapter we mentioned a list of several parameters, both for the processing phase and the worker initialization. However, it is important to state that the experimental setups we are about to analyze are not constructed by varying every parameter proposed.

Let's firstly look at the parameters that are fixed for every experimental setup:

- The **dataset** parameter is fixed to "full". Hence, in every experimental setup, the model will be finetuned on datasets constructed by all the raw data available. More details about this choice on section 4.2.2.
- The **full dataset** parameter is fixed to False. Hence, in every experimental setup, only the texts having relevant portions are used to finetune the model. More details about this choice on section 4.2.2.
- The **split on topics** parameter is fixed to True. Hence, in every experimental setup, the training, evaluation and testing sets contain, respectively, the 80% of topics / 10% of topics / 10% of topics.

The remaining parameters define univocally an experimental setup:

- The **model** parameter defines which model will be finetuned on the Token Classification task. It could vary and take values
  - "bert-uncased", i.e. the uncased version of BERT
  - "bert-cased", i.e. the cased version of BERT
  - "spanbert-cased", i.e. the cased version of SpanBERT
  - "roberta-uncased", i.e. the uncased version of RoBERTa
  - "roberta-cased", i.e. the cased version of RoBERTa
- The **binary** parameter indicates if the ground truths of an experimental setup are binary or not. It could take value True - hence, finetuning the model with binary ground truths - or False.
- The **contextual** parameter indicates if the dataset used in an experimental setup is enriched with contextual information about the updates. It could take value True - hence, finetuning the model with texts containing also contextual informations - or False.

Once stated how an experimental setup is built, let's briefly recall how are composed the datasets that will be used for both the finetuning and inference phases.

The datasets, stored on cloud on an Amazon S3 bucket, contain the following columns:

- The **text** column contains the processed update texts *in textual form*. This column is somehow useless, having stated that from the input_ids (i.e., the list of tokens coming from the Tokenizers) it is possible to re-obtain the original

update text (for more, see section 4.2.2). However, such data resulted to be very useful for speeding up the qualitative evaluation process: in fact, as mentioned in the finetuning algorithm (2, line 30), every "number of logging steps" the predictions of the model are qualitatively evaluated by logging the portion of text selected by the model. This could be achieved by having a Tokenizer that, from the input_ids, reconstructs the original text and then selects the relevant portion from the output of the model. Instead, by having the text information itself, this process is sped up since it is not required to decode the tokens.

- The **mask** column represents the ground truth labels, that can be binary or not, depending on the processing parameter. Such data is used for finetuning purposes only, in order to compare the predictions of the models and calculate evaluation metrics and losses.

- The **timestamp** column contains the information about the moment in time in which a news became available (for more, see section 4.2.2). It is needed especially for testing reasons, since we would like to sort the testing topics texts by means of this column in order to simulate the emission of updates.

- The **input_ids** column and the **attention_mask** column are the actual data that is used during the finetuning phase, used to fed the BERT-derived model. As anticipated, the **input_ids** are the tokens associated with an update text and the **attention_mask** are binary lists used by the model to focus on the meaningful tokens (for more, see section 4.2.2).

In the following sections, every experimental setup is presented and analyzed in details, along with the performances of every finetuned model in every environment. Finally, a brief discussion about the results achieved by the models and an analysis of the most promising ones, that will be used during the summaries production phase.

Binary masks, not contextual data

Let's dive into the experimental setups covered in this work by starting from the simplest one. The experimental setup presented in this paragraph is identified by the following parameters:

Parameter	Value
binary	True
contextual	False

This experimental setup is the "simplest" one, meaning that it is a base-point for starting to conduct further researches. There are at least two reasons for stating that:

- Firstly, being the ground truths list of zeros and ones, the model itself is somehow facilitated in the classification task: in fact, if the model randomly guesses the label of the n-th token, it has the 50% of probability of getting it right. In the concrete, it has to choose only between labeling a token as a 0 or as a 1.
- Furthermore, the actual data used to finetune the models is the simplest possible. In fact, the models receive only the tokenized texts and no other information.

Such an experimental setup is surely a useful evaluation baseline for further improvements of this work, at least for those based on binary ground truths. The results of the models on this environment will be fundamental to evaluate the goodness of the future enhancements, by inspecting if future tweaks of such experimental setup lead to better performances.

Of course, as anticipated, this environment is less comparable to not-binary ground truths setups, being their assumptions on the random guessing of the models completely different.

Model	**Training Accuracy**	**Validation Accuracy**
BERT uncased	0.5481	0.8502
BERT cased	0.5765	0.8339
RoBERTa uncased	0.5613	0.8324
RoBERTa cased	0.5911	0.7918
SpanBERT cased	0.5808	0.6707

Table 5.1: Binary masks, not contextual data: Training and Validation accuracies.

In table 5.1 are presented the results of the models in this experimental setup in terms of accuracy, calculated both in the training and testing set, after 1 epoch of finetuning.

Despite the fact that every model performs similarly in the training set (the values range within the $[0.5481, 0.5911]$ interval), it is possible to extract some interesting takeaways by looking at the results in the validation set.

In fact, in the training set, the most promising models are RoBERTa cased and SpanBERT cased: however, those 2 models are the worst performing on the validation set. This could be interpreted as a lack of generalization capabilities

of these 2 models, especially if compared with the best performing model on the validation set, BERT uncased, that has an accuracy delta of 0.3021 between validation and training accuracies (Table 5.2).

Model	Validation Accuracy - Training Accuracy
BERT uncased	0.3021
BERT cased	0.2574
RoBERTa uncased	0.2711
RoBERTa cased	0.2007
SpanBERT cased	0.0899

Table 5.2: Binary masks, not contextual data: Variation of accuracy between Validation and Training sets.

BERT uncased outperforms every other model in the validation set, leading to an accuracy of 0.8502, which is quantitatively pretty high.

In order to verify the actual goodness of this model in the validation set - hence, to prove the actual text extraction capabilities on unseen data -, in the following table (5.3) are compared some real relevant portions of updates and some predictions of BERT uncased:

Correct portion	Predicted portion
seven storey building collapses in thane	seven storey building collapses in thane
series of blasts	series of blasts hits iraqi capital
explosion has struck the main university in the northern city of aleppo, causing casualties	explosion has struck the main university in the northern of aleppo

Table 5.3: BERT uncased: Real portions of relevant text vs Predicted portions of relevant text.

As we can see from table 5.3, BERT uncased does not always perfectly predict the correct portion of text. However, its predictions are still pretty accurate and reasonable: the predicted portions are very likely to contain at least the correct portion and some other informative notions that enrich the portion itself and are still useful.

Binary masks, contextual data

In the previous paragraph we analyzed the simplest experimental setup possible, in which, nevertheless, BERT uncased performed very well.

In the following paragraph, we will tweak a little bit the previous experimental setup by trying to enrich the information contained in the data, by adding the context information.

Hence, this experimental setup parameters are:

Parameter	Value
binary	True
contextual	True

Having inspected the results of the models in the *not contextual* setup, we expect this setup to perform similarly. Still, we will be able to evaluate the benefits of the context information; moreover, having intuited that RoBERTa cased and SpanBERT cased lack of generalization in the validation set, we will understand if there are different needs with respect to the data used to train the models: RoBERTa and SpanBERT could possibly perform better when fed with richer data.

Model	Training Accuracy	Validation Accuracy
BERT uncased	0.5588	0.8219
BERT cased	0.5634	0.7034
RoBERTa uncased	0.5501	0.7866
RoBERTa cased	0.5596	0.6844
SpanBERT cased	0.5951	0.7393

Table 5.4: Binary masks, contextual data: Training and Validation accuracies.

From table 5.4, again, we can see that the results on the training set are compressed in a similar interval of accuracies, ranging in [0.5501,0.5951]. These results on the training set are slightly better than the ones with not-contextual data: however, as in the previous environment, the training accuracies do not reflect the actual goodness of the models on unseen data. In fact, in this setup, we notice an even smaller difference between validation accuracies and training accuracies (Table 5.5) that still awards BERT uncased as the best model, but with a lower accuracy (0.8219, with respect to the previous 0.8502).

Still, we can enlight some useful insights from SpanBERT cased, which was the worst performing model on the previous experimental setup. In this environment, instead, it highlights stronger generalization capabilities (the delta between validation accuracy and training accuracy grew from 0.0899 to 0.1442) and better overall

results: the validation accuracy in this setup is 0.7393, with an absolute increment with respect to the previous setup of 0.0686.

Furthermore, we cannot derive the same insights from RoBERTa cased, that did not perform well both on the previous and on the current experimental setups: in particular, it even gets worse, decreasing its overall validation accuracy from 0.7918 to 0.6844.

Finally, BERT uncased confirms its generalization capabilities by having both the best delta between validation and training accuracy (0.2631) and the best overall validation accuracy (0.8219).

Model	Validation Accuracy - Training Accuracy
BERT uncased	0.2631
BERT cased	0.14
RoBERTa uncased	0.2365
RoBERTa cased	0.1248
SpanBERT cased	0.1442

Table 5.5: Binary masks, contextual data: Variation of accuracy between Validation and Training sets.

Even if in absolute terms BERT uncased performs better with non contextual data, let's make again some comparisons between the correct relevant portions and the predicted relevant portions, on the validation set:

Correct portion	Predicted portion
seven storey building collapses in thane	seven storey building collapses in thane
series of blasts	series of blasts hits iraqi capital
explosion has struck the main university in the northern city of aleppo, causing casualties	explosion has struck the main university in the northern city of aleppo, causing casualties

Table 5.6: BERT uncased: Real portions of relevant text vs Predicted portions of relevant text.

As we can see, the predictions made in this environment are qualitatively comparable (and slightly better) than the ones made in the not-contextual environment. Hence, BERT uncased in this environment is still a well performing, reliable model.

Not binary masks, not contextual data

In the previous paragraphs we extracted some insights by analyzing the performances of the models finetuned on the Binary Token Classification task. In particular, we highlighted the most promising model - BERT uncased - and we qualitatively evaluated its results on the validation set.

In the following paragraphs, we will analyze the goodness of the BERT-derived finetuned models on a Not-Binary Token Classification task. As anticipated earlier, even if the task is similar, we have to consider lower baselines for the evaluation of our models in this environment. In fact, since the model has now the chance of predicting between 5 different labels, the probability of wrongly predict the label of the n-th token is way higher than before. Accordingly, also the probability of randomly guess the label of the n-th element correctly (20%) is lower than before (50%).

In order to recap, this experimental setup is conducted by the following parameters:

Parameter	Value
binary	False
contextual	False

Hence, the models will be finetuned on a Token Classification task with not-binary ground truths. As deeply discussed in section 4.2.2, the not-binary labels mentioned here can take the following values:

- value 0 if the i-th word is not a word belonging to the span
- value 1 if the i-th word is the first word belonging to the span
- value 2 if the i-th word belongs to the span but it is neither the first, nor the last word
- value 3 if the i-th word is the last word belonging to the span
- value 4 if the i-th word is the only word belonging to the span

Indeed, the models are requested to solve an harder task (with respect to the Binary Token Classification), since they have not only the aim of selecting the relevant words; the models have also to learn where a relevant portion starts, where it ends and which token is relevant by itself. If properly exploited, this setup could lead to qualitatively better results - since it defines the problem in a stronger, more strict manner.

From table 5.7, at first sight, it emerges that the training accuracies are more concentrated than before, even if in a lower interval ([0.3686,0.3805]). However, as

Model	Training Accuracy	Validation Accuracy
BERT uncased	0.3717	0.7539
BERT cased	0.3707	0.5072
RoBERTa uncased	0.3718	0.4656
RoBERTa cased	0.3686	0.5458
SpanBERT cased	0.3805	0.5383

Table 5.7: Not binary masks, not contextual data: Training and Validation accuracies.

previously anticipated, those results were expected to be worse with respect to the binary experiment 5.1.3.

Again, BERT uncased results as the better performing model on the validation set - reaching an accuracy value comparable to the other models on the binary experiment. Still, BERT uncased emerges as the most capable in generalizing, contrarily to SpanBERT cased and RoBERTa uncased - as demonstrated in table 5.8.

Model	Validation Accuracy - Training Accuracy
BERT uncased	0.3822
BERT cased	0.1365
RoBERTa uncased	0.0938
RoBERTa cased	0.1772
SpanBERT cased	0.1578

Table 5.8: Not binary masks, not contextual data: Variation of accuracy between Validation and Training sets.

As we did previously, let us analyze some examples of qualitative evaluation of BERT uncased on the validation set:

Correct portion	Predicted portion
seven storey building collapses in thane	seven storey building collapses in thane
series of blasts	series of blasts hits iraqi capital
(s) more than a dozen (e) (s) bombs went off in baghdad (e)	(s) more than a dozen bombs went off in baghdad just days after the last of the u. s. troops left iraq, leaving scores dead.

Table 5.9: BERT uncased: Real portions of relevant text vs Predicted portions of relevant text.

Note that the (s) and (e) identifiers are placed in the examples in order to signal which token was the start (or the end) of the portion.

Qualitatively, the model still does reasonably its job. However, quite often it misses the starting and end points of the relevant portion; this problem could be addressed by finetuning the model with more data and for more epochs.

Not binary masks, contextual data

In this last paragraph we will explore the results of the BERT-derived models on the experimental setup conducted by the following parameters:

Parameter	Value
binary	False
contextual	True

As seen earlier, the models struggle on the Not-Binary Token Classification because of the increased number of possible values each element can be labelled as. In this experimental setup, we will analyze if the context information is useful to improve the performances of the models.

Model	Training Accuracy	Validation Accuracy
BERT uncased	0.3599	0.4114
BERT cased	0.3616	0.4495
RoBERTa uncased	0.3473	0.4343
RoBERTa cased	0.3515	0.5225
SpanBERT cased	0.3723	0.4381

Table 5.10: Not binary masks, contextual data: Training and Validation accuracies.

However, as in the binary environment - in which we already experienced a loss of performances when adding the contextual information - also in the not binary environment we aggravate the performances of the models by enriching the data with the context guidance.

Model	Validation Accuracy - Training Accuracy
BERT uncased	0.0515
BERT cased	0.0879
RoBERTa uncased	0.087
RoBERTa cased	0.171
SpanBERT cased	0.0658

Table 5.11: Not binary masks, contextual data: Variation of accuracy between Validation and Training sets.

As we can see from table 5.10, the models perform the worst in this experimental setup. Along with very low training accuracies, we register also very low validation accuracies and hence very low increments between training and validation (Table 5.11).

In the previous paragraph we qualitatively evaluated the three best models, that had as validation accuracies - respectively - 0.8502, 0.8219 and 0.7539.

Let us now take a look at the predictions made with the best model of this environment, RoBERTa cased:

Correct portion	Predicted portion
(s) Seven storey building collapses in Thane (e)	(s) Seven storey building collapses in (s)
(s) Mumbra building collapse a fallout of illegal construction boom (e)	(s) Mumbra building collapse
(s) more than a dozen (e) (s) bombs went off in baghdad (e)	(s) more than a dozen bombs went off in baghdad just days after the last of the u. s. troops left iraq, leaving scores dead.

Table 5.12: RoBERTa cased: Real portions of relevant text vs Predicted portions of relevant text.

As anticipated by the value of our evaluation metric, that in this case is way lower than before, it emerges that RoBERTa cased does not perform well on the

validation set: in the examples we can see how the model is very likely to not correctly find the ending of the portions, indeed producing trimmed / incomplete sentences.

This (last) experimental setup was the most ambitious one and the models did not pay off for several reasons. In the following sections, we will inspect the results presented in these paragraphs - along with some further improvements that could be made in the future in order to enhance the performances of the models.

5.1.4 Finetuning results

In the last paragraphs we presented and analyzed the finetuning results of every model in every finetuning experimental setup presented in this work. For each experimental setup, we evaluated the goodness of the models by inspecting their validation accuracies, along with the qualitative evaluation of some predicted portions.

It resulted that by increasing the complexity of the experimental setup (i.e., by adding contextual information to the data or by having to perform not-binary classifications) the performances of the models get worse.

This insight opens up to wider considerations about these experiments: on one side, having a leaner process that requires little, concentrated amounts of data is very convenient. Having to manage less data[1] and having a simpler procedure that is able to consistently extract relevant portions of text from the updates is advantageous for several reasons related to storage-optimization, maintenance and intuitiveness.

On the other side, simple methods are not always suitable for generalized, cross-domain tasks in which it is requested high reliability across a wide range of data sources and topics.

However, in order to build a strong, fail-safe and trustworthy system able to extract relevant portions of text from different data sources, are needed

- **an higher variety of data and data sources**: in this work, the finetuning phase is approached by feeding the pretrained model with very few data (with respect also to the amount of data the model is pretrained on) coming from a single data source. Instead, in order to more robustly finetune the model,

[1]Here, with "less data", it is not intended to say that less data records are better; instead, with this statement, it is intended to say that being independent from other text-related information is crucial for having a leaner process. For example, being independent from the "context" information is fundamental when scaling up the number of data sources: in fact, our current data source provides such information, but it is not obvious that every other data source that will be integrated in the future can provide such attribute.

several data sources are needed to provide the model a wider range of examples both in terms of domain, context and writing style.

- **greater computational resources**: this work has been addressed by means of freely provided services (such as Google Colab). Obviously, the computational resources provided by free services are not enough to develop production-ready systems (the provided GPU is a K80 Tesla, that only support batch sizes up to 2 for this data): in fact, as mentioned earlier, the finetuning process of the models ends in just a single, long-lasting epoch (2 hours). Moreover, in order to capture as much extraction capabilities as possible within the duration of an epoch, the learning rate is set as higher as possible. Instead, having more performing computational resources and a wider range of data sources, it could be possible to finetune these models in more epochs, with a lower learning rate; moreover, by having more processing capabilities, it could be possible to parallelize the finetuning phase of several models, enabling the possibility of tune the hyperparameters of the model with smarter, more sustainable approaches.

Despite these conditions are needed for developing a more robust text extraction system, the results of the finetuning phase are very satisfactory: by means of a limited set of data, concentrated across less than 50 topics, and a limited resources capacity, this work proposes a reliable (at least within the testing scope of the data proposed by the TREC institute) BERT finetuned model able to capture consistently the relevant portion of a text.

The most promising finetuned model will be further exploited in the last step of this work as the core of the Temporal Summarization system.

5.2 Inference phase: summaries production

5.2.1 Overview

In the previous sections, we analyzed both the data processing phase and the finetuning phase proposed in this work. Both these phases are fundamental for the development of the Temporal Summarization system:

1. The data preparation phase is in charge of retrieving, processing and collecting the raw data, in order to easily serve the models in the finetuning phase.

2. The finetuning phase consists of a series of experiments - made in various, different environments - in which pretrained BERT-derived model are finetuned for extracting relevant portions of text from incoming updates. By means of evaluation metrics and qualitative analysis of the results, the most promising

finetuned model is chosen as the core of the whole Temporal Summarization system. However, by itself, this model is not able to produce up-to-date summaries about an evolving event. Instead, it is only able to capture the most important spans of a text.

In order to effectively exploit and take advantage of the relevant portions of text captured by the finetuned model, it is needed a production logic that conducts and regulates the whole system.

The aforementioned production logic is in charge of handling the predictions made by the finetuned model (i.e., the relevant spans of text suggested by the model) and timely update the end users with newly acquired information about a developing event.

5.2.2 Summaries production procedure

The summaries production procedure proposed in this work conducts the Temporal Summarization system itself and requires

1. **A working BERT model** able to capture relevant portion of texts from incoming updates.

2. **A test dataset**, **sorted by timestamp and concerning a single topic** (or context), from which are extrapolated the updates.

Internally, the production procedure has the purpose of receiving small, timely-ordered, relevant chunks of text and iteratively update the end users by emitting self-explanatory, syntactically correct sentences about the newly discovered information.

The production procedure relies on a pretrained Pegasus model for text generation, applied for paraphrasing purposes, and a cosine-similarity based decision metric, used to evaluate the redundancy of the updates.

Cosine similarity based decision metric

As anticipated, the production procedure proposed in this work receives timely-sorted relevant spans of texts from the finetuned BERT model. Despite the fact that the data comes from a single data source, it is very likely that an information that becomes available at timestamp T_i is reported in more than one update text.

However, the Temporal Summarization system - and in particular, the production procedure - must be able to avoid redundancy within the sentences it emits.

In order to disregard redundant updates, the production procedure embeds a cosine similarity based decision metric able to recognize redundancy between the already emitted updates (i.e., the update summary) and a given candidate update.

The cosine similarity metric measures the text similarity between two (vectorized)[2] documents A, B, with values ranging from 0 to 1: if the cosine similarity between two documents A, B is 1, it means that A, B have exactly the same meaning.

The mathematical equation of the Cosine Similarity metric can be computed as follows:

$$\texttt{Cosine Similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{5.1}$$

In order to exploit the cosine similarity metric for avoiding redundancy between the emitted sentences and the candidate, this work proposes a moving threshold that, if exceeded, indicates that the text should be discarded. Consider

- A **base threshold**, T_B
- An **upper bound threshold**, T_{UB}
- A **lower bound threshold**, T_{LB}
- The **number of emitted sentences** at time i, N_i
- A **threshold decay coefficient**, T_D

Consider also the following starting values:

Parameter	**Value**
Base threshold T_B	0.5
Upper bound threshold T_{UB}	0.5
Lower bound threshold T_{LB}	0.2
Threshold decay coefficient T_D	0.005

The moving threshold works as follows: whenever a candidate is received from the BERT model and paraphrased by Pegasus, the maximum cosine similarity metric[3] is computed between the candidate itself and the already emitted sentences.

[2]With the term "vectorized" it is intended that both the documents A and B are represented in vector form: hence, the cosine similarity metric is not computed directly on textual data; instead, it is calculated through the vector representations of the documents A, B.

[3]Here, with maximum cosine similarity metric, we refer to the maximum cosine similarity value between the candidate and any of the emitted sentences.

If this value is greater than the lower bound threshold T_{LB} and lower than the upper bound threshold T_{UB}, the candidate sentence is emitted to the end users. Then, the upper bound threshold is updated through the following mechanism:

$$T_{UB} = T_B \cdot \exp\left(-T_D \cdot N_i\right) \tag{5.2}$$

This equation represents a monotonically decreasing function that reassigns the value T_{UB} each time N_i grows. N_i grows every time a candidate sentence is emitted to the end users.

Indeed, through this moving threshold, the production procedure redefines the maximum value of similarity (i.e., the upper bound threshold T_{UB} each time a candidate is emitted) acceptable for emitting a new candidate. Hence, the moving threshold is stricter when no updates are emitted to the end users: this results in very distinct updates emitted at first. Instead, when the event is no more very recent - and indeed some updates are already emitted - the deciding threshold is lower, enabling the possibility of eviscerate the event by informing the end users with more similar updates.

Chapter 6

Results: TREC-TS Metrics

In the previous chapter, we analyzed both procedures and results of the Temporal Summarization system's finetuning and inference phases.

As anticipated in section 2.3, this work enriches the Sequential Update Summarization task proposed by the TREC conference: the original task requests to develop a system able to decide whether to emit or not sentences from a time-ordered corpus to the end users, in a timely manner; instead, in this work is presented a system that from a time-ordered corpus extracts relevant portions of text and emits concise, newly produced sentences to the end user.

Yet, the system proposed is still evaluated on the original metrics, presented in section 2.2:

- The **Expected Gain** metric, the sum of the relevance of each nugget matched by the emitted updates:

$$\texttt{ExpectedGain}(\mathcal{S}) = \mathbf{G}(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{u \in \mathcal{S}} \sum_{n \in \mathbf{M}(u)} \mathbf{g}(u, n) \tag{6.1}$$

  In the system proposed in this work, the emitted updates are the ones that originates the output sentences provided to the end users.

- The **Comprehensiveness** metric, the proportions of available information matched by the emitted updates:

$$\texttt{Comprehensiveness}(\mathcal{S}) = \mathbf{C}(\mathcal{S}) = \frac{1}{|\mathcal{N}|} \sum_{u \in \mathcal{S}} \sum_{n \in \mathbf{M}(u)} \mathbf{g}(u, n) \tag{6.2}$$

- **And in particular**, the harmonic mean of Expected Gain and Comprehensiveness, also referred to as **Combined**:

$$\texttt{Combined}(S) = 2 * \frac{\mathbf{C}(S) * \mathbf{G}(S)}{\mathbf{C}(S) + \mathbf{G}(S)} \tag{6.3}$$

Tables 6.1, 6.2 and 6.3 present the leaderboards of the TREC-TS Tracks for the years 2013, 2014 and 2015.

The tables were provided by the TREC institute along with the official Temporal Summarization Track Overviews and are enriched with the results of this work.

Each table presents the Expected Gain and Comprehensiveness metrics for each Team that submitted results; moreover, the Combined metric is computed by means of these metrics through equation 2.7. Finally, the results are sorted from the highest Combined value to the lowest, as an indication of the overall performances.

The metrics results proposed in tables 6.1, 6.2 and 6.3 are comprehensive of the performances of each system on every query proposed for each year. Indeed, results proposed in table 6.1 are the average results achieved by each system on the 9 events proposed in the 2013 challenge - the same for tables 6.2 and 6.3.

Before discussing the results of this work, it is also worth mentioning that tables 6.1, 6.2 and 6.3 *do not* cover every result proposed along the following years; indeed, in these tables, the results of this work are only compared to the results of the participant systems.

The Temporal Summarization system proposed in this work resulted to be well performing on each year data, always reaching a good balance between Expected Gain and Comprehensiveness results. Differently from other systems, our system achieves results comparable to the top-tier participants, both for the Expected Gain and the Comprehensiveness metrics; instead, other systems metrics result imbalanced on one of these two: for instance, "UWaterlooMDS-*" in 2013, "BPUT PRIS" and "ICTNET" in 2014, "udel fang" and "USI" in 2015.

In particular, our system systematically achieves good information coverage proportions, resulting in a good Comprehensiveness metrics; moreover, the relevance of the information extracted is comparable to the top-standing participants.

- In the 2013 leaderboard, the system proposed in this work stands 4th, reaching very good results on the Comprehensiveness metric but lacking in information relevance.
- In the 2014 leaderboard, instead, the system outperformed the other participants - achieving both the best results on the Expected Gain and Combined metrics. Moreover, the system results also comparable to the top 4 participants in the Comprehensiveness metric.
- In the 2015 leaderboard, the system achieves relevant performances on the Comprehensiveness metric but, as for the 2013 results, the system lacks in capturing all the relevant information.

TeamID	Expected Gain	Comprehensiveness	**Combined**
ICTNET-run2	0.102	0.192	**0.1332**
ICTNET-run1	0.101	0.194	0.1328
hltcoe-TuneExternal2*	0.109	0.162	0.1303
This work	0.0752	0.3409	0.1232
hltcoe-TuneBasePred2*	0.093	0.18	0.1226
PRIS-cluster5	**0.149**	0.099	0.119
PRIS-cluster3	0.103	0.131	0.1153
uogTr-uogTrNMTm1MM3	0.077	0.201	0.1113
PRIS-cluster2	0.071	0.204	0.1053
PRIS-cluster1	0.065	0.224	0.1008
PRIS-cluster4	0.065	0.224	0.1008
uogTr-uogTrNSQ1	0.069	0.174	0.0988
uogTr-uogTrNMTm3FMM4	0.062	0.183	0.0926
uogTr-uogTrNMM	0.057	0.244	0.0924
hltcoe-BasePred	0.053	0.281	0.0892
ALL–	0.055	0.231	0.0888
hltcoe-Baseline	0.049	0.291	0.0839
uogTr-uogTrEMMQ2	0.05	0.241	0.0828
hltcoe-EXTERNAL	0.047	0.318	0.0819
wim GY 2013-SUS1	0.037	0.128	0.0574
UWaterlooMDS-rg4	0.025	0.386	0.047
UWaterlooMDS-rg3	0.024	0.384	0.0452
UWaterlooMDS-rg2	0.021	0.441	0.0401
UWaterlooMDS-rg1	0.02	**0.445**	0.0383
UWaterlooMDS-UWMDSqlec2t25	0.016	0.433	0.0309
UWaterlooMDS-UWMDSqlec4t50	0.016	0.423	0.0308
UWaterlooMDS-CosineEgrep	0.007	0.013	0.0091
UWaterlooMDS-NormEgrep	0.001	0.05	0.002
BJUT-Q0*	0.0	0.0	0.0

Table 6.1: TREC-TS2013: Leaderboard results.

TeamID	Expected Gain	Comprehensiveness	**Combined**
This work	**0.0864**	0.3763	**0.1378**
BJUT	0.0657	0.4088	0.1132
BJUT	0.0632	0.3979	0.1091
BJUT	0.0632	0.3979	0.1091
cunlp	0.0631	0.322	0.1055
uogTr	0.0467	0.4453	0.0845
ICTNET	0.0531	0.1081	0.0712
uogTr	0.0387	0.3691	0.0701
IRIT	0.0383	0.3521	0.0691
IRIT	0.0378	0.3538	0.0683
uogTr	0.0347	0.4539	0.0645
average	0.0327	0.3615	0.06
cunlp	0.0325	0.3058	0.0588
ICTNET	0.0418	0.0934	0.0578
IRIT	0.0298	0.378	0.0552
IRIT	0.0289	0.3764	0.0537
uogTr	0.0281	**0.4733**	0.0531
IRIT	0.0285	0.3806	0.053
IRIT	0.0225	0.4012	0.0426
cunlp	0.0174	0.4265	0.0334
BUPT PRIS	0.0155	0.2692	0.0293
BUPT PRIS	0.0115	0.338	0.0222
ICTNET	0.0079	0.407	0.0155
ICTNET	0.007	0.409	0.0138
BUPT PRIS	0.0059	0.3728	0.0116
BUPT PRIS	0.0033	0.4369	0.0066

Table 6.2: TREC-TS2014: Leaderboard results.

TeamID	Expected Gain	Comprehensiveness	**Combined**
WaterlooClarke	0.235	0.352	**0.2818**
WaterlooClarke	**0.2872**	0.2584	0.272
WaterlooClarke	0.2252	0.3421	0.2716
cunlp	0.1371	0.487	0.214
cunlp	0.1203	0.5372	0.1966
cunlp	0.1011	0.4584	0.1657
IRIT	0.0849	0.4959	0.145
This work	0.0843	0.4366	0.1232
Mean	0.0666	0.4342	0.1155
IRIT	0.0518	0.5899	0.0952
BJUT	0.0445	0.6123	0.083
BJUT	0.0413	**0.6155**	0.0774
IRIT	0.0422	0.2939	0.0738
l3sattrec15	0.0408	0.3612	0.0733
l3sattrec15	0.04	0.3669	0.0721
IRIT	0.0306	0.3391	0.0561
l3sattrec15	0.0283	0.2276	0.0503
udel fang	0.0258	0.5294	0.0492
udel fang	0.0206	0.5819	0.0398
udel fang	0.0189	0.5965	0.0366
USI	0.0182	0.5713	0.0353
USI	0.0179	0.5837	0.0347
USI	0.0171	0.6133	0.0333
USI	0.0169	0.5758	0.0328

Table 6.3: TREC-TS2015: Leaderboard results.

As anticipated in section 2.2, this work is also compared with the system proposed by McCradie et al. in the publication "Explicit Diversification of Event Aspects for Temporal Summarization" [15], in which the most performing methods introduced reached exceptional results.

Before diving into the comparison of the system proposed in this work and the systems proposed in the aforementioned publication, some comments are needed:

- The publication proposed different systems addressing the TS task. Still, only the two most performing systems are compared to this work in this section, being the base-systems a combination of known approaches to Temporal Summarization, already exploited by the other participants. Instead, the two most performing systems rely on the xQuAD [59] and IASelect [75] explicit diversification frameworks, exploited for the sentence selection sub-task.

- Concerning the most promising methods proposed by the publication (henceforth we will refer to them as IASelect-Wikipedia, IASelect-Crowd, xQuAD-Wikipedia, xQuAD-Crowd), they take advantage of both semi-manually (via crowdsourcing) and automatically (from Wikipedia) generated additional data sources for finetuning the systems. Indeed, the method proposed in this work is comparable through the TREC-TS metrics (see section 2.2) with IASelect-Wiki/Crowd and xQuAD-Wiki/Crowd, but still having in mind that further information and data were crawled by the latters.

Table 6.4 presents the comparisons between the system proposed in this work and IASelect-Wikipedia, IASelect-Crowd, xQuAD-Wikipedia, xQuAD-Crowd on all 46 events present in the TREC-TS datasets. Again, the results are sorted from the most performing method to the least performing method; moreover, the best results for each metric (Expected Gain, Comprehensiveness and Combined) are highlighted in bold.

System	Aspects	Expected Gain	Comprehensiveness	**Combined**
xQuAD	Wikipedia	0.3864	**0.4450**	**0.4136**
xQuAD	Crowdsourcing	0.5369	0.3252	0.4050
IASelect	Crowdsourcing	0.6519	0.2584	0.3700
IASelect	Wikipedia	**0.7955**	0.2148	0.3382
This work	None	0.0842	0.4366	0.1412

Table 6.4: TREC-TS2013-2015: comparison of the system proposed in this work with IASelect-Wikipedia, IASelect-Crowd, xQuAD-Wikipedia, xQuAD-Crowd.

The most promising method is the xQuAD-Wikipedia system, which combines good results both on the Expected Gain and Comprehensiveness metrics. Hence,

xQuAD-Wikipedia captures a good proportion of **relevant** information through the update it emits (see section 2.2). Note that, among the methods proposed by McCradie [15], xQuAD-Wikipedia is the less performing in terms of Expected Gain and the best performing in terms of Comprehensiveness.

At the same time, IASelect-based methods result top performing on the Expected Gain metric - hence, these systems are optimal in capturing the most relevant updates in a corpus; however, they achieve poor performances on the Comprehensiveness metric, meaning that they tend to emit only updates that are very relevant.

The system proposed in this work results comparable to the Comprehensiveness performances of the most performing method xQuAD-Wikipedia, with a difference of 0.0084 in absolute terms. Indeed, our model covers almost the same number of information covered by the best method presented.

The key difference between our system and the systems proposed by McCradie is that the latters introduce a very accurate sentence scoring method that exploiting additional data sources can consistently and reliably select the sentences that are most informative. Instead, the system proposed in this work has a much simpler decision metric that relies on the cosine similarity, based solely on the redundancy of the candidate sentence with respect to the update summary. Indeed, our method lacks in capturing the key informations to be emitted, whereas the methods proposed by McCradie perform incredibly well thanks to the explicit diversification based sentence selection methods.

Chapter 7

Conclusion and future works

This work proposes a Temporal Summarization system able to emit not redundant, newly generated updates about a developing event from news articles and / or textual documents coming from the Internet.

Furthermore, it has the purpose of enriching the TREC Sequential Update Summarization task (see section 2.1) by adding a text abstraction layer to the originally requested system: indeed, this work aims at covering the whole Temporal Summarization downstream task, instead of stopping at the Information Retrieval stage.

The development of this system evolved along three major stages in which the process took shape.

Firstly, the data was gathered, processed, collected and maintained on cloud. Afterwards, several experiments were made by means of cutting-edge Machine Learning models, in which they were finetuned for capturing significant portions of text from the input documents. Following a long-lasting period of experimentation in which the models were quantitatively and qualitatively evaluated, the most promising architecture - resulting highly consistent and precise in the identification of relevant parts of a news article - was further employed as the core mechanism of the whole system, conducted by a summaries production logic that timely emits concise updates to the end users.

The system proposed in this work achieves results comparable to the top participants, for each edition of the TREC-TS challenge. Systematically, our system achieves great comprehensiveness (see definition at Equation 2.9) performances, indeed covering adequate proportions of available information through the emission of freshly generated sentences.

Moreover, our system is very reliable in capturing relevant information from a sentence: the solid backbone this system is built on, BERT [60] [61] is extraordinary in extracting text-related features and representing relationships between words; indeed, BERT confirms its versatility in tackling the vast majority of Natural

Language Processing tasks, even though with slender finetuning phases and few data.

Yet, this system has problems that could be addressed in further researches. In particular, the system sometimes results weak in interpreting the actual relevance of a predicted span of text. In particular, if we consider a given sub-event, the system is inherently tempted to emit the relevant portion of text of the first document (related to the sub-event) it receives, in favour of an increased emission timeliness. This is mostly due to the production logic, that relies solely on a cosine similarity based decision metric (see sections 5.2, 5.2.2). This decision metric penalizes the emission of updates similar to already emitted sentences: however, the first document of a sub-event is very likely to result different from the already emitted sentences - and indeed, it is very likely that it is delivered. In order to solve this problem, it could be interesting to apply online clustering-based methods on the predicted relevant portions of text, in order to choose from a sub-event cluster the appropriate, most relevant span of text. Moreover, given the results of explicit diversification based approaches in selecting relevant sentences, it could be useful to apply these methods for the sake of online-filtering the texts the update summary is based on.

Appendix A

Costa Concordia: generated summary

Below, an example of update summary produced by a Temporal Summarization system based on finetuned BERT for the "Costa Concordia" event.

1. Most of the passengers and crew on board the ship were evacuated to the island of giglio.
2. Costa cruises has confirmed that there are 3000 passengers and 1000 crew members on board.
3. According to the coastguard, the ship had taken on water and was listing at 20 degrees.
4. The costa concordia hit a reef near the isola del while on a trip around the Mediterranean.
5. There was a cruise ship that ran aground near the italian island.
6. It was proving difficult to rescue the last few passengers, and the mayor feared more casualties.
7. The ship shuddered to a halt for electrical reasons after hearing a large boom, but they were initially told it was because of a boom.
8. Luckily there's an island near us.
9. jeanne marie de champ said that faced with the chaotic scene at the lifeboats, she decided
10. As passengers scrambled to get off the ship, it was chaos.

11. The costa crociera company said that it was not yet possible to say what caused the problem, but that they had evacuated the vessel.

12. Most of the passengers were Italian, as well as some french and german.

13. The costa concordia's hull was torn open on friday and investigators are trying to figure out what went wrong and how far human error was to blame.

14. More than a dozen people died when the massive ship hit a reef in January.

15. There are 8 dead and 30 injured.

16. Many people are still looking for bodies after two elderly people were found on a cruise ship.

17. There were scenes of panic like on the titanic.

18. The night time rescue operation was assisted by the helicopters with search lights.

19. The costa concordia had a gross internal tonnage of more than 112000 metric tons.

20. As a second in command he was promoted to the rank of captain and in 2006 he was given command of the newly launched, 114500 tonne costa

21. Rescue workers have been working through the night on the stricken costa concordia as 29 people are now feared missing.

22. The captain of the ship has been arrested and accused of manslaughter, abandoning the vessel and causing the shipwreck.

23. The costa concordia cruise liner captain is in custody.

24. The national vice president of government relations for the largest union of merchant marines officers in the united states said that schettino should never have fled the costa concordia after it capsized.

25. The commander should leave at the end.

26. As long as the weather holds and stays, the search will go on, according to the coast guard and fire rescue teams.

27. The company is charged with removing 2535 tons of fuel from the ill fated ship before it leaks into the sea.

28. 300 meters from the rocks was where awe were navigating.

29. The growing size of cruise liners poses risks.
30. Carnival cruise lines, the parent company of costa cruises and owners of the ship, saw its stock fall almost 14 percent Tuesday.
31. Costa has started the process of taking bids for the recovery operation.
32. A judge in Italy delayed a decision on whether to release schettino from jail.
33. Costa crociere are trying to stave off lawsuits that are expected to cost hundreds of millions of euros.
34. The concordia's owner, Costa crociere, denied that people were allowed on the plane without following proper procedures.
35. Germany's two biggest reinsurers are bracing for tens of millions of dollars in payouts following the capsizing of the costa concordia cruise ship near Italy.
36. After the boat was lowered, goduti papa, for about 45 minutes.
37. Before pumping it out of the vesse fuel tanks, teams hope to apply heat to make it less.
38. The bad weather on Saturday forced the Dutch company that was contracted to pump out the heavy fuel and diesel oil from the costa concordia to delay the work.

Bibliography

[1] J. Gottfried and E. Shearer. *News Use Across Social Media Platforms in 2016*. 2016 (cit. on p. 2).

[2] E. Shearer and K. E. Matsa. *News Use Across Social Media Platforms in 2018*. 2018 (cit. on p. 2).

[3] E. Shearer and A. Mitchell. *News Use Across Social Media Platforms in 2020*. 2020 (cit. on pp. 2, 3).

[4] E. Shearer. *More than eight-in-ten Americans get news from digital devices*. 2020 (cit. on p. 2).

[5] M. Barthel and K. Worden. *Newspapers Fact Sheet*. 2021 (cit. on p. 2).

[6] Khalid Adam. «Big Data Analysis and Storage». In: Sept. 2015 (cit. on p. 3).

[7] *Most Surprising Blog Statistics: How Many Blog Posts Are Published per Day?* `https://letter.ly/how-many-blog-posts-are-published-per-day/`. Accessed: 2021-03-25 (cit. on p. 3).

[8] Wikipedia. *Natural Language Processing — Wikipedia, The Free Encyclopedia*. 2013. URL: `https://en.wikipedia.org/wiki/Natural_language_processing` (cit. on p. 4).

[9] Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. «A Survey of the Usages of Deep Learning for Natural Language Processing». In: *IEEE Transactions on Neural Networks and Learning Systems* 32.2 (2021), pp. 604–624. DOI: `10.1109/TNNLS.2020.2979670` (cit. on p. 4).

[10] David E. Rumelhart and James L. McClelland. «Learning Internal Representations by Error Propagation». In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987, pp. 318–362 (cit. on pp. 4, 11).

[11] Michael I. Jordan. «Chapter 25 - Serial Order: A Parallel Distributed Processing Approach». In: *Neural-Network Models of Cognition*. Ed. by John W. Donahoe and Vivian Packard Dorsel. Vol. 121. Advances in Psychology. North-Holland, 1997, pp. 471–495. DOI: https://doi.org/10.1016/S0166-4115(97)80111-2. URL: https://www.sciencedirect.com/science/article/pii/S0166411597801112 (cit. on pp. 4, 11).

[12] Sepp Hochreiter and Jürgen Schmidhuber. «Long Short-term Memory». In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735 (cit. on pp. 4, 11).

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. «Attention is All you Need». In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf (cit. on pp. 4, 11, 13, 15).

[14] J. Aslam, F. Diaz, M. Ekstrand-Abueg, R. McCreadie, V. Pavlu, and T. Sakai. «TREC 2015 Temporal Summarization Track Overview». In: 2015 (cit. on pp. 4, 9, 22).

[15] Richard McCreadie, Rodrygo L. T. Santos, Craig Macdonald, and Iadh Ounis. «Explicit Diversification of Event Aspects for Temporal Summarization». In: *ACM Trans. Inf. Syst.* 36.3 (Feb. 2018). ISSN: 1046-8188. DOI: 10.1145/3158671. URL: https://doi.org/10.1145/3158671 (cit. on pp. 8, 13, 57, 58).

[16] Manling Li, Tengfei Ma, Mo Yu, Lingfei Wu, Tian Gao, Heng Ji, and Kathleen McKeown. «Timeline Summarization based on Event Graph Compression via Time-Aware Optimal Transport». In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 6443–6456. DOI: 10.18653/v1/2021.emnlp-main.519. URL: https://aclanthology.org/2021.emnlp-main.519 (cit. on p. 9).

[17] Julius Steen and Katja Markert. «Abstractive Timeline Summarization». In: *Proceedings of the 2nd Workshop on New Frontiers in Summarization*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 21–31. DOI: 10.18653/v1/D19-5403. URL: https://aclanthology.org/D19-5403 (cit. on p. 9).

[18] Adhika Pramita Widyassari, Supriadi Rustad, Guruh Fajar Shidik, Edi Noersasongko, Abdul Syukur, Affandy Affandy, and De Rosal Ignatius Moses Setiadi. «Review of automatic text summarization techniques methods». In: *Journal of King Saud University - Computer and Information Sciences* (2020). ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2020.05.006. URL: https://www.sciencedirect.com/science/article/pii/S1319157820303712 (cit. on p. 10).

[19] H. P. Luhn. «The Automatic Creation of Literature Abstracts». In: *IBM Journal of Research and Development* 2.2 (1958), pp. 159–165. DOI: 10.1147/rd.22.0159 (cit. on p. 11).

[20] Ani Nenkova. «Summarization evaluation for text and speech: issues and approaches.» In: Jan. 2006 (cit. on p. 11).

[21] Hans Christian, Mikhael Agus, and Derwin Suhartono. «Single Document Automatic Text Summarization using Term Frequency-Inverse Document Frequency (TF-IDF)». In: *ComTech: Computer, Mathematics and Engineering Applications* 7 (Dec. 2016), p. 285. DOI: 10.21512/comtech.v7i4.3746 (cit. on p. 11).

[22] Julian Kupiec, Jan O. Pedersen, and Francine R. Chen. «A trainable document summarizer». In: *SIGIR '95*. 1995 (cit. on p. 11).

[23] John M. Conroy and Dianne P. O'leary. «Text Summarization via Hidden Markov Models». In: New York, NY, USA: Association for Computing Machinery, 2001. ISBN: 1581133316. DOI: 10.1145/383952.384042. URL: https://doi.org/10.1145/383952.384042 (cit. on p. 11).

[24] Kristian Woodsend and Mirella Lapata. «Automatic Generation of Story Highlights». In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden: Association for Computational Linguistics, July 2010, pp. 565–574. URL: https://aclanthology.org/P10-1058 (cit. on p. 11).

[25] Tsutomu Hirao, Yasuhisa Yoshida, Masaaki Nishino, Norihito Yasuda, and Masaaki Nagata. «Single-Document Summarization as a Tree Knapsack Problem». In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1515–1520. URL: https://aclanthology.org/D13-1158 (cit. on p. 11).

[26] Yihong Gong and Xin Liu. «Generic Text Summarization Using Relevance Measure and Latent Semantic Analysis». In: SIGIR '01. New Orleans, Louisiana, USA: Association for Computing Machinery, 2001, pp. 19–25. ISBN: 1581133316. DOI: 10.1145/383952.383955. URL: https://doi.org/10.1145/383952.383955 (cit. on p. 11).

[27] Ansamma John, P.S. Premjith, and M. Wilscy. «Extractive multi-document summarization using population-based multicriteria optimization». In: *Expert Systems with Applications* 86 (2017), pp. 385–397. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2017.05.075. URL: https://www.sciencedirect.com/science/article/pii/S0957417417304049 (cit. on p. 11).

[28] Ping Chen and R. Verma. «A Query-Based Medical Information Summarization System Using Ontology Knowledge». In: *19th IEEE Symposium on Computer-Based Medical Systems (CBMS'06)*. 2006, pp. 37–42. DOI: 10.1109/CBMS.2006.25 (cit. on p. 11).

[29] Elena Baralis, Luca Cagliero, Saima Jabeen, Alessandro Fiori, and Sajid Shah. «Multi-Document Summarization Based on the Yago Ontology». In: 40.17 (Dec. 2013), pp. 6976–6984. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2013.06.047. URL: https://doi.org/10.1016/j.eswa.2013.06.047 (cit. on p. 11).

[30] Leonhard Hennig, Winfried Umbrath, and Robert Wetzker. «An Ontology-Based Approach to Text Summarization». In: WI-IAT '08. USA: IEEE Computer Society, 2008, pp. 291–294. ISBN: 9780769534961. DOI: 10.1109/WIIAT.2008.175. URL: https://doi.org/10.1109/WIIAT.2008.175 (cit. on p. 11).

[31] Ziqiang Cao, Furu Wei, Li Dong, Sujian Li, and Ming Zhou. «Ranking with Recursive Neural Networks and Its Application to Multi-Document Summarization». In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI'15. Austin, Texas: AAAI Press, 2015, pp. 2153–2159. ISBN: 0262511290 (cit. on p. 11).

[32] Jianpeng Cheng and Mirella Lapata. «Neural Summarization by Extracting Sentences and Words». In: *CoRR* abs/1603.07252 (2016). arXiv: 1603.07252. URL: http://arxiv.org/abs/1603.07252 (cit. on p. 11).

[33] Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. «SummaRuNNer: A Recurrent Neural Network based Sequence Model for Extractive Summarization of Documents». In: *CoRR* abs/1611.04230 (2016). arXiv: 1611.04230. URL: http://arxiv.org/abs/1611.04230 (cit. on p. 11).

[34] Jiacheng Xu and Greg Durrett. «Neural Extractive Text Summarization with Syntactic Compression». In: *CoRR* abs/1902.00863 (2019). arXiv: 1902.00863. URL: http://arxiv.org/abs/1902.00863 (cit. on p. 11).

[35] Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning. «Parsing Natural Scenes and Natural Language with Recursive Neural Networks». In: *Proceedings of the 28th International Conference on*

International Conference on Machine Learning. ICML'11. Bellevue, Washington, USA: Omnipress, 2011, pp. 129–136. ISBN: 9781450306195 (cit. on p. 11).

[36] Jiwei Li and Eduard Hovy. «A Model of Coherence Based on Distributed Sentence Representation». In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 2039–2048. DOI: 10.3115/v1/D14-1218. URL: https://aclanthology.org/D14-1218 (cit. on p. 11).

[37] Wei Jiang, Yunfeng Zou, Ting Zhao, Qiang Zhang, and Yinglong Ma. «A Hierarchical Bidirectional LSTM Sequence Model for Extractive Text Summarization in Electric Power Systems». In: *2020 13th International Symposium on Computational Intelligence and Design (ISCID)*. 2020, pp. 290–294. DOI: 10.1109/ISCID51228.2020.00071 (cit. on p. 11).

[38] Michihiro Yasunaga, Rui Zhang, Kshitijh Meelu, Ayush Pareek, Krishnan Srinivasan, and Dragomir Radev. «Graph-based Neural Multi-Document Summarization». In: *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*. Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, pp. 452–462. DOI: 10.18653/v1/K17-1045. URL: https://aclanthology.org/K17-1045 (cit. on p. 11).

[39] Peng Cui, Le Hu, and Yuanchao Liu. «Enhancing Extractive Text Summarization with Topic-Aware Graph Neural Networks». In: *CoRR* abs/2010.06253 (2020). arXiv: 2010.06253. URL: https://arxiv.org/abs/2010.06253 (cit. on p. 11).

[40] Danqing Wang, Pengfei Liu, Yining Zheng, Xipeng Qiu, and Xuanjing Huang. «Heterogeneous Graph Neural Networks for Extractive Document Summarization». In: *CoRR* abs/2004.12393 (2020). arXiv: 2004.12393. URL: https://arxiv.org/abs/2004.12393 (cit. on p. 11).

[41] Farida Mohsen, Jiayang Wang, and Kamal Al-Sabahi. «A hierarchical self-attentive neural extractive summarizer via reinforcement learning (HSASRL)». In: *Applied Intelligence* 50 (Sept. 2020). DOI: 10.1007/s10489-020-01669-5 (cit. on p. 12).

[42] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. «Ranking Sentences for Extractive Summarization with Reinforcement Learning». In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 1747–1759. DOI: 10.18653/v1/N18-1158. URL: https://aclanthology.org/N18-1158 (cit. on p. 12).

[43] Pierre-Etienne Genest and Guy Lapalme. «Fully Abstractive Approach to Guided Summarization». In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Jeju Island, Korea: Association for Computational Linguistics, July 2012, pp. 354–358. URL: https://aclanthology.org/P12-2069 (cit. on p. 12).

[44] Hideki Tanaka, Akinori Kinoshita, Takeshi Kobayakawa, Tadashi Kumano, and Naoto Katoh. «Syntax-Driven Sentence Revision for Broadcast News Summarization». In: *Proceedings of the 2009 Workshop on Language Generation and Summarisation (UCNLG+Sum 2009)*. Suntec, Singapore: Association for Computational Linguistics, Aug. 2009, pp. 39–47. URL: https://aclanthology.org/W09-2808 (cit. on p. 12).

[45] Pierre-Etienne Genest and Guy Lapalme. «Framework for Abstractive Summarization using Text-to-Text Generation». In: *Proceedings of the Workshop on Monolingual Text-To-Text Generation*. Portland, Oregon: Association for Computational Linguistics, June 2011, pp. 64–73. URL: https://aclanthology.org/W11-1608 (cit. on p. 12).

[46] Katja Filippova and Michael Strube. «Sentence Fusion via Dependency Graph Compression». In: *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*. Honolulu, Hawaii: Association for Computational Linguistics, Oct. 2008, pp. 177–185. URL: https://aclanthology.org/D08-1019 (cit. on p. 12).

[47] Jackie Chi Kit Cheung and Gerald Penn. «Unsupervised Sentence Enhancement for Automatic Summarization». In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 775–786. DOI: 10.3115/v1/D14-1085. URL: https://aclanthology.org/D14-1085 (cit. on p. 12).

[48] Lidong Bing, Piji Li, Yi Liao, Wai Lam, Weiwei Guo, and Rebecca Passonneau. «Abstractive Multi-Document Summarization via Phrase Selection and Merging». In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 1587–1597. DOI: 10.3115/v1/P15-1153. URL: https://aclanthology.org/P15-1153 (cit. on p. 12).

[49] Alexander M. Rush, Sumit Chopra, and Jason Weston. *A Neural Attention Model for Abstractive Sentence Summarization*. 2015. DOI: 10.48550/ARXIV.1509.00685. URL: https://arxiv.org/abs/1509.00685 (cit. on p. 12).

[50] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. DOI: 10.48550/ARXIV.1409.0473. URL: https://arxiv.org/abs/1409.0473 (cit. on p. 12).

[51] Sumit Chopra, Michael Auli, and Alexander M. Rush. «Abstractive Sentence Summarization with Attentive Recurrent Neural Networks». In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, June 2016, pp. 93–98. DOI: 10.18653/v1/N16-1012. URL: https://aclanthology.org/N16-1012 (cit. on p. 12).

[52] Ahmed Amir Tazibt and Farida Aoughlis. «Latent Dirichlet allocation-based temporal summarization». In: *International Journal of Web Information Systems* 15.1 (Jan. 2019), pp. 83–102. ISSN: 1744-0084. DOI: 10.1108/IJWIS-04-2018-0023. URL: https://doi.org/10.1108/IJWIS-04-2018-0023 (cit. on p. 12).

[53] Qian Liu, Yue Liu, Dayong Wu, and Xueqi Cheng. «ICTNET at Temporal Summarization Track TREC 2013». In: *Proceedings of The Twenty-Second Text REtrieval Conference, TREC 2013, Gaithersburg, Maryland, USA, November 19-22, 2013*. Ed. by Ellen M. Voorhees. Vol. 500-302. NIST Special Publication. National Institute of Standards and Technology (NIST), 2013. URL: http://trec.nist.gov/pubs/trec22/papers/ICTNET-ts.pdf (cit. on p. 12).

[54] Yun Zhao, Fei Yao, Huayang Sun, and Zhen Yang. «BJUT at TREC 2014 Temporal Summarization Track». In: *Proceedings of The Twenty-Third Text REtrieval Conference, TREC 2014, Gaithersburg, Maryland, USA, November 19-21, 2014*. Ed. by Ellen M. Voorhees and Angela Ellis. Vol. 500-308. NIST Special Publication. National Institute of Standards and Technology (NIST), 2014. URL: http://trec.nist.gov/pubs/trec23/papers/pro-BJUT%5C_ts.pdf (cit. on pp. 12, 13).

[55] Chunyun Zhang, Weiyan Xu, Fanyu Meng, Hongyan Li, Tong Wu, and Lixin Xu. *The Information Extraction systems of PRIS at Temporal Summarization Track* (cit. on p. 13).

[56] Tan Xu, Douglas W. Oard, and Paul McNamee. «HLTCOE at TREC 2013: Temporal Summarization». In: *Proceedings of The Twenty-Second Text REtrieval Conference, TREC 2013, Gaithersburg, Maryland, USA, November 19-22, 2013*. Ed. by Ellen M. Voorhees. Vol. 500-302. NIST Special Publication. National Institute of Standards and Technology (NIST), 2013. URL: http://trec.nist.gov/pubs/trec22/papers/hltcoe-ts.pdf (cit. on p. 13).

[57] Rafik Abbes, Bilel Moulahi, Abdelhamid Chellal, Karen Pinel-Sauvagnat, Nathalie Hernandez, Mohand Boughanem, Lynda Tamine, and Sadok Ben Yahia. «IRIT at TREC Temporal Summarization 2015». In: *34th Text REtrieval Conference (TREC 2015)*. Thanks to National Institute of Standards and Technology (NIST) publisher. The definitive version is available at: http://trec.nist.gov/pubs/trec24/papers/IRIT-TS.pdf. Gaithersburg, Maryland, US: National Institute of Standards and Technology (NIST), 2015, pp. 1–10. URL: https://oatao.univ-toulouse.fr/15470/ (cit. on p. 13).

[58] Rodrygo L. T. Santos, Craig Macdonald, and Iadh Ounis. «Search Result Diversification». In: *Foundations and Trends® in Information Retrieval* 9.1 (2015), pp. 1–90. ISSN: 1554-0669. DOI: 10.1561/1500000040. URL: http://dx.doi.org/10.1561/1500000040 (cit. on p. 13).

[59] Rodrygo L.T. Santos, Craig Macdonald, and Iadh Ounis. «Exploiting Query Reformulations for Web Search Result Diversification». In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, pp. 881–890. ISBN: 9781605587998. DOI: 10.1145/1772690.1772780. URL: https://doi.org/10.1145/1772690.1772780 (cit. on pp. 13, 57).

[60] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423 (cit. on pp. 13, 15, 21, 59).

[61] Wikipedia contributors. *BERT (language model) — Wikipedia, The Free Encyclopedia*. [Online; accessed 30-December-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=BERT_(language_model)&oldid=1052964320 (cit. on pp. 15, 59).

[62] Anna Rogers, Olga ovaleva, and Anna Rumshisky. «A Primer in BERTology: What We Know About How BERT Works». In: *Transactions of the Association for Computational Linguistics* 8 (2020), pp. 842–866. DOI: 10.1162/tacl_a_00349. URL: https://aclanthology.org/2020.tacl-1.54 (cit. on p. 15).

[63] Yinhan Liu et al. «RoBERTa: A Robustly Optimized BERT Pretraining Approach». In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692. URL: http://arxiv.org/abs/1907.11692 (cit. on pp. 15, 16).

[64] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. «SpanBERT: Improving Pre-training by Representing and Predicting Spans». In: *CoRR* abs/1907.10529 (2019). arXiv: 1907.10529. URL: http://arxiv.org/abs/1907.10529 (cit. on pp. 15, 16).

[65] Ian Tenney, Dipanjan Das, and Ellie Pavlick. «BERT Rediscovers the Classical NLP Pipeline». In: *CoRR* abs/1905.05950 (2019). arXiv: 1905.05950. URL: http://arxiv.org/abs/1905.05950 (cit. on p. 21).

[66] Fábio Souza, Rodrigo Frassetto Nogueira, and Roberto de Alencar Lotufo. «Portuguese Named Entity Recognition using BERT-CRF». In: *CoRR* abs/1909.10649 (2019). arXiv: 1909.10649. URL: http://arxiv.org/abs/1909.10649 (cit. on p. 21).

[67] Kai Hakala and Sampo Pyysalo. «Biomedical Named Entity Recognition with Multilingual BERT». In: *Proceedings of The 5th Workshop on BioNLP Open Shared Tasks*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 56–61. DOI: 10.18653/v1/D19-5709. URL: https://aclanthology.org/D19-5709 (cit. on p. 21).

[68] Kai Labusch, Staatsbibliothek Zu, Berlin Kulturbesitz, Clemens Neudecker, and David Zellhöfer. «BERT for Named Entity Recognition in Contemporary and Historical German». In: Oct. 2019 (cit. on p. 21).

[69] Jouni Luoma and Sampo Pyysalo. «Exploring Cross-sentence Contexts for Named Entity Recognition with BERT». In: *CoRR* abs/2006.01563 (2020). arXiv: 2006.01563. URL: https://arxiv.org/abs/2006.01563 (cit. on p. 21).

[70] Xiangyang Li, Huan Zhang, and Xiao-Hua Zhou. «Chinese clinical named entity recognition with variant neural structures based on BERT methods». In: *Journal of Biomedical Informatics* 107 (2020), p. 103422. ISSN: 1532-0464. DOI: https://doi.org/10.1016/j.jbi.2020.103422. URL: https://www.sciencedirect.com/science/article/pii/S1532046420300502 (cit. on p. 21).

[71] Henry Tsai, Jason Riesa, Melvin Johnson, Naveen Arivazhagan, Xin Li, and Amelia Archer. «Small and Practical BERT Models for Sequence Labeling». In: *CoRR* abs/1909.00100 (2019). arXiv: 1909.00100. URL: http://arxiv.org/abs/1909.00100 (cit. on p. 21).

[72] Magnus Jacobsen, Mikkel H. Sørensen, and Leon Derczynski. «Optimal Size-Performance Tradeoffs: Weighing PoS Tagger Models». In: *CoRR* abs/2104.07951 (2021). arXiv: 2104.07951. URL: https://arxiv.org/abs/2104.07951 (cit. on p. 21).

[73] Rakia Saidi, Fethi Jarray, and Mahmud Mansour. «A BERT Based Approach for Arabic POS Tagging». In: *Advances in Computational Intelligence*. Ed. by Ignacio Rojas, Gonzalo Joya, and Andreu Català. Cham: Springer International Publishing, 2021, pp. 311–321. ISBN: 978-3-030-85030-2 (cit. on p. 21).

[74] *Text REtrieval Conference (TREC) Website*. `https://trec.nist.gov/` (cit. on p. 22).

[75] Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. «Diversifying Search Results». In: *Proceedings of the Second ACM International Conference on Web Search and Data Mining*. WSDM '09. Barcelona, Spain: Association for Computing Machinery, 2009, pp. 5–14. ISBN: 9781605583907. DOI: `10.1145/1498759.1498766`. URL: `https://doi.org/10.1145/1498759.1498766` (cit. on p. 57).