POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

An Extraction-Abstraction Hybrid Approach for Financial Document Summarization

Supervisors

Prof. Luca CAGLIERO

Dott. Moreno LA QUATRA

Dott. Jacopo FIOR

Candidate

Sofia PEROSIN

December 2021

Abstract

Nowadays, the quantity of data that a company manages is huge, and it is expected that it will increase in the future. For this reason, it is particularly appealing to design data-driven tools and methodologies capable of managing these pieces of information. This thesis project aims at addressing this challenge, by implementing a summarization pipeline, which is capable of offering an overview of the main topics contained in the analyzed text, by creating the corresponding summary and headline. To pursue this objective a procedure articulated in two main steps is implemented: *extractive summarization* and *abstractive summarization*. Namely, different approaches are analyzed to set up the first phase, instead the last step is always implemented with a *Transformer* architecture. The general framework is firstly tested on a general-purpose news dataset, to compare it with state-of-the-art models, then it is finetuned on a financial news dataset, to generate a model tailored to the financial news summarization problem. Future works should focus on enhancing the reliability of the procedure, by creating more specific financial datasets, that would be exploited to finetune the model. In fact, to the best of our knowledge, there is a lack of public available financial datasets, and this sets a limit to the achievable performances.

Summary

In 2017, "*The Economist*" posted an article titled "*The world's most valuable resource is no longer oil, but data*", in which data are described as the new, lucrative emerging commodity of the digital era: not by chance, most of the powerful companies in the world rely on data advantage. The main critic to this analysis was based on the fact that the competitive advantage is not uniquely related to the quantity of data, but it relies mostly on the *quality*, the *reliability* and the *usability* of data: companies must be able to exploit this information, because data, generally speaking, on their own, do not guarantee success. A key point that allows companies to get a *sustainable* competitive advantage from data, is the capacity to get valuable insights from these sources, and this is strictly related to the speed with which data relevance depreciates over time. In light of this, the thesis' work tries to address this issue, by analyzing some possible pipelines, to deploy an instrument that may be used to get an at-a-glance overview of textual data. Namely, the final scope is to obtain a model capable of working with financial information, especially since they depreciate fast and so their potential value should be exploited as soon as possible.

The branch of AI, *Artificial Intelligence*, that handles language information, is NLP, *Natural Language Processing*. Which is in turn subdivided into NLU, *Natural Language Understanding*, and NLG, *Natural Language Generation*. The former looks at understanding the meaning of human language, the latter addresses the challenge of text generation. In fact, NLP carries out a lot of different tasks on human language as: *Document Summarization*, summarization of a document in a shorter version preserving the general meaning, *Machine Translation*, translation of a document from one language to another, *Sentiment Analysis*, understanding the general opinion/mood characterizing a document.

A core component of NLP, is the LM, *Language Model*. This architecture is in charge of ensuring the *validity* of the generated text. Namely, the concept of *validity* is not simply related to the grammatical correctness, but it refers to the consistency between the machine-generated text and human-writing style. Different *Language*

Models have been proposed in the literature: firstly they were implemented through *Statistical Machine Learning*, then *Deep Learning* techniques started to dominate benchmarks. Currently, most of the SOTA *State Of The Art* models are based on *Transformer* architecture, and also the procedure adopted in the final pipeline proposed in this thesis' work relies on it.

The focus of this project is to deploy a *summarization pipeline* to be applied to financial documents. In fact, almost all of the benchmarks refer to general news summarization or scientific papers summarization; and the same applies to the available checkpoints to initialize the models. Therefore this thesis' project aims at starting to fill this gap, especially with regards to abstractive summarization, since extractive summarization has already been further developed. In fact, to the best of our knowledge, in the literature are not yet public available tested abstractive models aimed at generating headlines for financial documents.

With regards to *summarization* (the task of interest of this work), there are two approaches that can be followed to summarize a text: *extractive* or *abstractive*. The main difference between these processes relies in the typologies of the obtained final results: *extraction* produces a summary by selecting the most salient sentences from the source text, *abstraction* composes a summary by analyzing the text and creating new sentences (using also words that were not present in the original document), in a more human-like style.

Going into procedural details of the proposed pipeline, starting from the original text, firstly it is applied an *extractive summarization* to obtain a summary, then it is exploited an *abstractive summarization* to get a headline. The overall process is articulated in two steps because of a limitation of *Transformer* architecture: the input must have a maximum length, and if its length exceeds the threshold, then the model will automatically truncate the text, with the risk of losing important information. For this reason, a first screening is done by selecting the most significant sentences and only over those will be applied the *abstractive* step.

Transformer is a *Language Model* based on contextualized embeddings: it can represent a word according to the context in which it is. *Transformer* presents an encoder-decoder architecture, where both the encoder and decoder are a stack of six identical layers. In the encoder each layer is made of a *multi-head attention* and a *feed forward* network. In the decoder instead, each layer is composed by a *masked multi-head self-attention*, a *multi-head self-attention* and a *feed forward* network. The main mechanisms in this model are: *positional encodings* and the *multi-head attention*. The former is used to fed into the model, not only the single word, but also its position inside the sentence (to preserve the order); and to achieve this, it

incorporates this information through a sinusoidal function. The latter helps the model to focus on specific parts of the text, namely it allows for attending to parts of the sequence differently.

Specifically, the suggested pipeline is implemented with *PreSumm* model for the *extractive summarization*, and *PEGASUS* model for the *abstractive summarization*.

PreSumm is based on BERT, *Bidirectional Encoder Representations from Transformers*, and it is characterized by some specific properties to make the model suitable for summarization. Namely, it works at sentence-level representation, and this is ensured by the usage of special tokens (*[CLS]*, *[SEP]*) that allows to distinguish each sentence. Moreover, despite it is based on *Transformer* architecture, it is not affected by length limitations with regards to the input text, thanks to some modifications done to the structure of position embedding mechanism.

PEGASUS is, in turn, based on *Transformer* architecture. The peculiarity of this model relies in the training modality. In fact, training is performed through a self-supervised objective, which consists in masking whole sentences and generating only these missing sentences starting from the remaining ones in the document, this is the core component of *PEGASUS*: GSG, *Gap Sentences Generation*. Namely, the masked sentences are right the ones which are considered more important, according to a specific scoring.

The aim of the experiments is to test the implemented pipeline (and some variations of it), to understand their capability of performing summarization task, and to identify the best configuration to achieve the underlined purpose.

The final step consists in the evaluation of the obtained results. The performances are measured by *ROUGE* score, in particular the metrics taken into account are *R1-Fscore*, *R2-Fscore* and *RL-Fscore*. These refers to the accuracy registered considering the overlapping between the reference summaries (or headlines) and the generated ones. This overlay is computed taking into account uni-grams for *R1-Fscore*, bi-grams for *R2-Fscore* and longest co-occurences for *RL-Fscore*.

Once defined, the pipeline has to be tested. Because of, as stated before, the benchmarks are defined on general news dataset, to compare the proposed pipeline with the *SOTA* techniques, a first assessment is done on *NEWSROOM* dataset. This dataset consists of around 1.3 million articles collected from some major publications, and it is very useful since it allows to test end-to-end the model, because for each article it provides the corresponding summary and headline. The performaces registered on *NEWSROOM*, compared with other proposed models in

the literature as *SHEG* for example, are quite satisfactory. In fact, the pipeline, in summary generation, registers about 58% improvement on *R2* compared to *SHEG*, and about 42% improvement on *R2* in headline generation.

Finally, after have ensured the reliability of the pipeline, it is applied to *REUTERS* dataset. This is a financial news dataset, containing about 800000 articles. This collection allows to test the model on financial documents, although it provides only the reference headline. To overcome the absence of the reference summary, different techniques are tested to find the best choice to implement the *extraction* step: *lead k sentences selection*, *TextRank algorithm*, *PreSumm* trained in self-supervised way, *PreSumm* trained on *NEWSROOM* dataset. The most valuable configuration turns out to be *lead 3 sentences selection* for the *extraction*, along with *PEGASUS* for the abstraction.

However, since no benchmarks for summarization of financial documents are public available, understanding the reliability of this pipeline on *REUTERS* dataset is not possible. This is why, future work should move in this direction: commit to creating a financial dataset for an end-to-end testing in such a way to obtain an official benchmark to measure the goodness of a model on financial documents summarization.

Acknowledgements

Dedico questa tesi a mia mamma e a mio papà, che hanno sempre creduto in me, sostenendomi nei momenti più difficili, e festeggiando con me i momenti più belli.

Un ringraziamento speciale va al mio relatore Luca Cagliero, e ai correlatori Moreno La Quatra e Jacopo Fior che mi hanno affiancata durante lo svolgimento di questo lavoro.

Alla mia famiglia, grazie.

Ad Aghi, Dile e Nai, le mie amiche di una vita. La prova che i legami veri sopravvivono al tempo, alla distanza e a qualsiasi ostacolo si possa incontrare.

Ad Ange, il mio mare. A quelle amicizie che fanno giri immensi e poi ritornano, più forti di prima. A tutte le risate, le uscite e le avventure di questi ultimi anni, "per fortuna ci sei tu".

A Lucia e Vale che mi hanno fatto sentire sempre la benvenuta, che mi hanno accolto a braccia aperte.

Alle pallavoliste, i miei cuori.

A Dani e Manuel, a quelle amicizie nate da poco ma che capisci subito esser importanti.

Ad Ari, alle insalate di pollo e crackers al mais, ai giri al Bennet, ai nodini agli zaini perché non mi derubassero, e alle bottiglie in aeroporto. Sei stata una delle prime amiche che ho trovato a Torino, mi hai fatto sentire a casa, e non potrò mai ringraziarti abbastanza.

A Giulia, ai balli isterici prima dell'esame di Analisi I, agli Hit, al Twinings, e ovviamente alla Romana e Cammafà. Ai telefoni che si scaricano di notte con

la neve mentre stiamo spiegando al taxi dove siamo e ai ritorni magici dal Cacao. Alla miglior coinquilina che potessi trovare, plagiata dal mio accento e dalle mie manie.

A Sveva, a tutte le avventure, soprattutto a quelle che qui non è il caso di elencare. Alle feste, allo Chalet che ha sancito la nostra amicizia, ai telefoni rubati, ai notturni rincorsi. Alle domeniche passate all'Opera sgomitando per entrare. Ad un'amicizia che né il tempo e né la distanza ha mai fatto affievolire.

A Fabri e Gabri, alle serate pizza e alla cena che ancora vi devo cucinare. Siete stati le mie luci al primo anno, al mio "Perché i è uguale a due?" avete trovato la forza e la bontà d'animo per farmi capire i cicli for; non sarei sopravvissuta al Poli senza di voi.

Ad Adri, Bianca, Marta e Marti, le mie super colleghe che mi sono state accanto durante questo percorso.

A Christian, il mio guru, che ad ogni dubbio e problema mi ha sempre aiutata.

A Irene, una persona dal cuore d'oro, che non ha esitato un momento ad aiutarmi quando ero in difficoltà.

A tutti i miei amici, a chi c'era e a chi c'è, grazie, vi voglio bene.

Sofia

Table of Contents

Acronyms

AI
Artificial Intelligence

BART
Bidirectional and Auto-Regressive Transformers

BERT
Bidirectional Encoder Representations from Transformers

CAC
Controlled Actor-Critic

CNN
Convolutional Neural Network

CRF
Conditional Random Fields

ELMo
Embeddings from Language Models

GPT
Generative Pre-trained Transformer

GSG
Gap Sentences Generation

LDA
Latent Dirichlet Allocation

LM

Language Model

LST

Long Short-Term Memory

MLM

Masked Language Model

NLG

Natural Language Generation

NLP

Natural Language Processing

NLU

Natural Language Understanding

NSP

Next Sentence Prediction

PGN

Pointer-Generator Networks

PPMI

Positive Pointwise Mutual Information

RNN

Recurrent Neural Network

ROUGE

Recall-Oriented Understudy for Gisting Evaluation

SOTA

state of the art

SVD

Singular Value Decomposition

SVM

Support Vector Machines

T5

Text-to-Text Transfer Transformer

Chapter 1

Introduction

Now more than ever, the amount of information is very huge, but the time available to spend on managing these data is declining. So it is very important to be able to exploit this information in an efficient way, since data on their own, do not guarantee success. A key point that allows companies to get a sustainable competitive advantage from data, is the capacity to get valuable insights from these sources, and this is strictly related to the speed with which data relevance depreciates over time. That is why it is very useful to get an instrument which allows to get a preview of all this information, may producing a concise summary with the most salient details. This is even more true for companies operating in financial sectors: they have to make decisions quickly, and despite they have gathered a lot of data to support their strategies, it could happen that it is not feasible for them to process all of the information on time, especially if it is encoded by textual corpora.

Focusing on the task of interest, *summarization*, two approaches can be adopted: *extractive* and *abstractive*. The former produces a summary by selecting the most salient sentences from the source text, the latter composes a summary by analyzing the text and creating new sentences in human-fashion way. The aim of this work is to analyze some strategies in support to this process, by offering an instrument capable of summarizing text, and producing also a concise headline to get an at-a-glance view of the topic. Namely, the final objective it is to construct a pipeline suitable for working with financial news. In fact, to the best of our knowledge, in the literature are not yet public available tested abstractive models aimed at generating headlines for financial documents.

The first part of this dissertation is devoted to analyzing the branch of AI which deals with summarization task, NLP (*Natural Language Processing*) and its core component, LM (*Language Model*), that is the architecture which is in charge of ensuring the validity (in the sense of consistency between the machine-generated

text and human-writing style) of the created sentences. Different LMs have been proposed in the literature, and currently, the state-of-the-art is represented by the *Transformer* architecture. This work has also explored the literature to look for datasets suitable for doing a comparison between the proposed pipelines and previous works, and making the model suitable for working with financial data. Moreover, state-of-the-art techniques are analyzed, to understand the best models to implement in the final pipeline; mainly focusing on *Transformer*, a *Language Model* based on contextualized embeddings. This architecture is articulated in an encoder-decoder structure, where both encoder and decoder are made up by a stack of 6 identical layers. In the *encoder* each layer is made of a *multi-head attention* and a *feed forward network*. In the decoder instead, each layer is composed by a *masked multi-head self-attention*, a *multi-head self-attention* and a *feed forward network*.

The analysis of state-of-the-art techniques is followed by the presentation of the proposed pipeline. In this framework, the first step is implemented through an *extractive summarization* to get the summary, and then, an *abstraction summarization* is exploited to obtain the corresponding headline. It has been decided to adopt this hybrid approach because *Transformer* architecture suffers from dealing with long text. In fact, the input must have a maximum length, and if its length exceeds the threshold, then the model will automatically truncate the document, with the risk of losing important information. Because of that, firstly the most important sentences are selected, in order to feed to the *Transformer* a shorter input.

Specifically, the *extraction* is performed by *PreSumm*, while the *abstraction* is implemented through *PEGASUS*; these two models will be further described in the dissertation.

Finally, the obtained results are tested with *ROUGE* score, a metric that takes into account the overlapping between the reference summaries (and the reference headlines) and the corresponding generated ones.

As previously mentioned, to the best of our knowledge, no public benchmarks for abstractive financial summarization are available. For this reason, the reliability of the pipeline is firstly ensured by testing it on *NEWSROOM*, a general news dataset, because in this way it is possible to compare the pipeline with state-of-the-art techniques. After that, the framework is used to generate summaries and headlines for *REUTERS*, a financial news dataset. However, understanding the reliability of the pipeline on REUTERS dataset is not possible. This is why future work should move in this direction: commit to creating a financial dataset for an end-to-end testing in such a way to obtain an official benchmark to measure the goodness of a model devoted to financial documents summarization.

Chapter 2

Natural Language Processing fundamentals

2.1 NLP framework

Natural Language Processing (NLP) is that branch of AI, Artificial Intelligence, that looks at understanding and replicating human languages.

Many challenges can be addressed with NLP, for example:

- **Document Summarization**, summarization of a document in a shorter version, preserving the general meaning
- **Machine Translation**, translation of a document from one language to another
- **Sentiment Analysis**, understanding the general opinion/mood stored in a document

Altogether, NLP is divided into two main categories: **NLU** and **NLG**.

Natural Language Understanding (NLU), looks at understanding the meaning of human language; instead Natural Language Generation (NLG), addresses the challenge of text generation.

These are very tough challenges, which must deal with many different problems, as has been shown in [1]:

- **Ambiguity** regarding the syntax, the concepts interpretation, the lexis.

  For example, the sentence "I am very happy to be here, and so is Giulia" can be interpretated as "Giulia and I are both happy that I am here" or also "I am happy to be here, and Giulia is happy to be here" (*syntactic ambiguity*).

Another example could be: the word "Will" inserted at the begging of a sentence, indicates the name of a person or the future tense helping verb?

- **Figurative language**, namely those slangs of which the meaning behind is not directly understandable from the written expression.

  An example could be the expression "ghosting" (the cutting of communication).

- **Common sense knowledge**, there are ambiguities that can be solved only by reasoning.

  In fact, it is very simple for a human to understand that the sentence "I saw a mouse in the barn" refers to a view of an animal, but this is not for an NLU model, which struggles to understand if "mouse" refers to the animal or the pointing device.

2.1.1 Language Models

LM, Language Models are the core components of *NLP*, they are in charge of constructing (understanding) a sentence which is reasonable according to the grammar rules of the language.

Originally, *LM* models were implemented through Machine Learning: *Statistical Machine Learning* techniques started to dominate benchmarks. These approaches relied on k^{th} order Markov assumptions, and the probability of the analyzed sentence was calculated according to the statistics of $k-$gram frequencies in a large text corpus. For example *SVM* (Support Vector Machines) was widely used in sentiment analysis [1].

The next revolution occurred with deep learning, where network architectures like *LSTM* (Long Short-Term Memory) and *CNN* (Convolutional Neural Network) were exploited. The neural models belonging to this category, like *RNN* and *Transformer*, in fact allow to face with the curse of dimensionality (when data become very sparse because they are represented in a high-dimensional space), and represent a step forward compared to machine learning techniques [1].

Currently, the benchmarks are mostly dominated by models based on *Transformers* architecture, which will be deepened in later sections.

2.2 Embeddings techniques

One of the most important steps in NLP is the encoding of the language information in a low-dimensional vector space. Different techniques can be adopted to implement this process:

- **Word Embeddings**
- **Graph Embeddings**
- **Sense Embeddings**
- **Contextualized Embeddings**
- **Sentence and Document Embeddings**

An overall introduction will be done for all the listed techniques, to get a general overview. Then, the approaches adopted in this dissertation will be detailed properly.

Word Embeddings

Traditional approach

The traditional approach is the so called *Count Based model*, which looks at constructing a matrix where word frequencies are stored. Different matrices can be used to store this information. For example in the *Word-context* matrix the columns and the rows correspond to the words, each cell will contain the co-occurrences of the two words in the document.

Since raw values do not provide much information, the PPMI (Positive Point-wise Mutual Information) takes into account the co-occurrences of the words by normalizing these measures by the frequency of each word. In this way it is possible to get how likely to happen are the co-occurrences [1].

It is easy to foresee how a dimensionality reduction is needed in these cases. In fact since each column, and row, corresponds to a word in the vocabulary, the dimension of the matrix could increase exponentially.

One of the most used techniques is SVD, Singular Value Decomposition. This consists in factorizing the initial matrix, M, into three smaller matrices:

$$M = U\Sigma V^*$$

Σ contains the *singular values* (the values in its diagonal), and also with only a set of this values it is possible to reconstruct the initial matrix M [1].

Word2vec

The novel approach introduced with Deep Learning constructs the *Embedding* exploiting neural networks.

One of the more famous techniques with regards to *Word Embeddings* is **Word2vec**.This is characterized by a feedforward neural architecture, trained with language modelling objective.

An example of **Word2vec** model is the *Continuous Bag-Of-Words*. This, exploiting the surrounding context, looks at predicting the current word by minimizing the loss

$$L = -log(p(\vec{w_t}|\vec{W_t}))$$

where $\vec{w_t}$ is the target word and $\vec{W_t}$ indicates the sequence of words the context [1].

Moreover, another variant of **Word2vec** is *Skip-gram*, which is very similar to the former, but the main difference is that in this case the algorithm tries to predict the words in the context given the target word [1].

Graph Embeddings

Nowadays, a lot of information is represented through graph structures. This is the reason why, a specific technique to encode this kind of knowledge is very useful.

There are two main approaches [1]:

- *Node embedding*, where the graph's nodes are embedded in a semantic space by preserving the distances
- *Relation embedding*, where the graphs's edges are the subjects of attention

Focusing on *Node embedding*, one of the most famous techniques is **Autoencoder-based model**. The main idea is to train the autoencoder to encode the representation from which it can be reconstructed the original input. This process is articulated in two main steps:

- a context vector is extracted for each node
- the context vector is encoded by the autoencoder into a lower dimensional space

The general idea, depicted in figure 2.1, consists in starting from a node (node number 3 in orange in the image) and to extract the context vector, which in this example is based on adjacency statistics. Then, the autoencoder will compress this extracted vector in a smaller embedding [1].

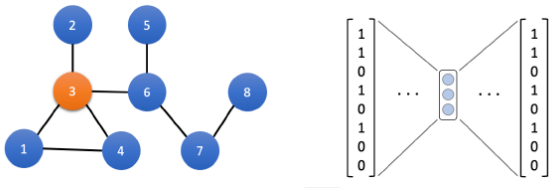

Figure 2.1: Graph embedding [1]

Sense Embeddings

The main objective of this approach is to deal with ambiguous lexical meaning. Two main strategies can be followed:

- Unsupervised
- Supervised

The former exploits only the information contained in the input text: by analysing the context, the model predicts different sense for each word. There are two sub-categories inside unsupervised sense embeddings :

- *clustering-based*, which is articulated in two steps (firstly sense induction, then representation learning). As depicted in figure 2.2, firstly the algorithm gets the occurrences of the analyzed word, then studies the contexts and infers the senses for the word, finally computes the sense representation [1]
- *joint training*, which performs both step together

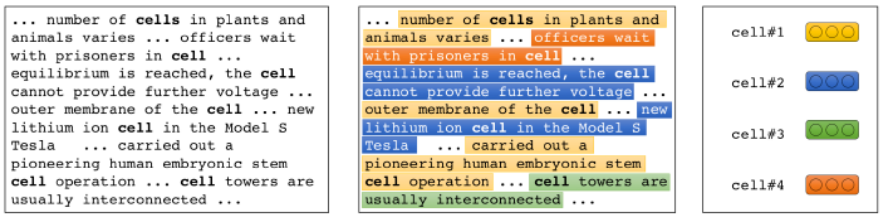

Figure 2.2: Sense embedding [1]

In supervised sense embedding, instead the model relies on external sources to enhance its overall sense knowledge. The main sources exploited are textual definitions (sense embedding model is initialized with pre-trained word embedding) and semantic networks (where nodes are concepts and edges are the relations between concepts) [1].

Sentence and Document Embeddings

This approach is very interesting since it allows to embed longer units of meaning.

With regards to *Sentence Embeddings*, there are two approaches: unsupervised and supervised. An example of the former is the *Sentence-level training of word*. This model aims at predicting surrounding sentences given an inputted one. *Skip-Thought*, through an RNN architecture, encodes the input sequence into an intermediate representation, and by decoding this intermediate representation, the generated sentence is obtained. An example is reported in figure 2.3, where it is possible to see how firstly, the algorithm encodes the given phrase in an intermediate representation, then the decoder starting from this result, produces the output sequence [1]. Instead for *Supervised sentence embeddings*, the main architecture is similar to the previous one, but in this case it is taken into account also additional text corpora.

Specifically, *Sent2Vec* and *Doc2Vec* are two models for this typology of embedding, and they derives from *Word2Vec*. In fact, the result is very similar to the one obtained with *Word2Vec*: word vectors (as in *Word2Vec*) plus sentence vectors (for *Sent2Vec*) or document vectors (for *Doc2Vec*).

Figure 2.3: Unsupervised sentence embedding [1]

Contextualized Embeddings

This typology deserves a particular attention, since *Transformers* architecture (widely used in state-of-the-art techniques) is the core component of this approach.

The power of this embedding lies on its capability of representing a word according to the context in which it is: the same word in different contexts can be represented in different ways, this is a *dynamic embedding*.

In fact, the weakness of word embedding techniques, as *Word2vec*, is the fixed representation for each word. Regardless of the context, the same word will be represented always in the same way: the context is ignored and higher order semantic phenomena can't be caught [1].

Moreover, *Contextualized Embedding* does not need of external resources: the learning process is totally unsupervised.

Contextualized Embedding is integrated in NLP model. It results to be an internal state of the neural network, and according to the typology of encoder implemented, it can be identified two main categories of embeddings: *RNN* and *Transformer*.

RNN-Based models In this case, the main architecture is a LSTM-based encoder. RNNs present some advantages as:

- capability of dealing with words order
- capability of giving more attention to those words semantically closer to the context

An example of this strategy is ELMo (Embeddings from Language Models). *ELMo*, as it is depicted in figure 2.4, is composed of 2-layer bidirectional LSTM with some residual connections in between the LSTMs.

This model is trained according to a language modelling objective, and it is then employed to get the contextualized embedding used as input in different NLP models [1].

Transformer-Based models Transformer architecture presents many advantages with respect to RNN, for example the possibility of parallelization and the capability of processing the input in a bi-directional way. Different models, based on Transformers architecture, were adopted to perform *Contextual Embedding*:

- **GPT**, Generative Pre-trained Transformer. Its architecture takes only the decoder of the Transformer, and its objective is to predict a word given a sequence of words [1]
- **BERT**, Bidirectional Encoder Representation form Transformer. Its architecture consists in the Transformer's encoder, and it will be deepened lately.

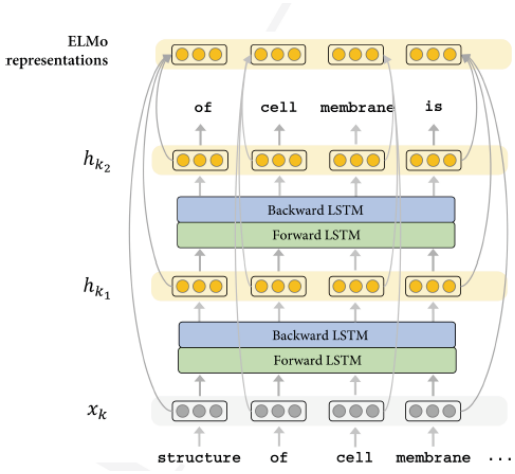

Figure 2.4: ELMo architecture [1]

Chapter 3

Datasets

This dissertation at building a summarization pipeline able to deal with financial documents.

The starting point consists in testing the analyzed methods through well-known datasets, employed to bechmark state-of-the-art algorithms, in order to identify the best performing models. In pursuit of this cause, the following datasets are exploited:

- CNN/DailyMail [2]
- GigaWord
- NEWSROOM [3]

The available checkpoints on these datasets will be then used as starting point for final models finetuning.

Furthermore, in the last step, a specific financial dataset is used, to obtain a final model suitable for financial document summarization:

- Reuters rcv1 [4]

3.1 CNN/DailyMail

This dataset consists of around 300k articles written by CNN and Daily Mail journalists. The used version is the one offered by HuggingFace Datasets library, which contains:

- 287113 training samples
- 13368 validating samples
- 11490 testing samples

Each record is made of two features:

- *article*, the body of the article
- *highlights*, the corresponding highlights of the article

For the sake of completeness, an example is reported in table 3.1.

article	**highlights**
"(CNN) – The 2013 America's Cup will be faster than ever after organizers announced that wingsail catamarans will be the vessels of choice. The race has historically been between yachts with a single hull, however the 34th edition of the contest will be between multihull vessels with wings rather than traditional sails. This means the boats will travel faster through the water, with top speeds in excess of 30 knots, almost three times as fast as in the past. The Golden Gate Yacht Club, hosts of the 2013 race and holders of the cup, have also announced a new, shorter race format for the competition. ..... "	"The 34th America's Cup will take place on faster wingsail catamarans . An annual America's Cup World Series will begin in 2011, according to organizers . A youth competition will also be introduced in 2012 to boost interest among youngsters."

Table 3.1: CNN/DailyMail example

3.2 GigaWord

This dataset consists of around 4 million articles collected from Gigaword. The used version is the one offered by HuggingFace Datasets library, which contains:

- 3803957 training samples
- 189651 validating samples
- 1951 testing samples

Each record is made of two features:

- *document*, the body of the article
- *summary*, the corresponding headline

For the sake of completeness, examples are reported in table 3.2.

document	**summary**
"us roman catholic bishops on wednesday charged that immigration raids on us workplaces break up families and disrupt communities , without addressing the country 's flawed immigration system."	"us catholic bishops condemn us immigration raids"
"facebook 's new look became mandatory wednesday in a shift to what the popular social networking website says is a faster , streamlined and more UNK format that has some devotees in a state of rebellion ."	"facebook makeover nettles some devotees"

Table 3.2: GigaWord examples

3.3 NEWSROOM

This dataset consists of around 1.3 million articles collected from 38 major publications.

It is very useful since, thanks to the availability of article, summary and title, it allows to train and test the model end-to-end.

The used version is the one downloaded from Cornell site (`https://lil.nlp.cornell.edu/newsroom/index.html`), manual download is required. It contains:

- 995041 training samples
- 108837 validating samples
- 108862 testing samples

Each record is made of twelve features:

- *url*, URL of the article
- *archive*, URL of the archive
- *title*, headline of the article
- *date*, date of the article
- *text*, body of the article
- *summary*, summary of the article
- *compression*, compression ratio
- *coverage*, extractive coverage
- *density*, extractive density
- *compression bin*, it can be: low, medium, high
- *coverage bin*, it can be: extractive, abstractive
- *density bin*, it can be: low, medium, high

However, for the scope of the work, only *text*, *summary* and *title* fields will be used, and the remaining ones are discarded.

For the sake of completeness, an example is reported in table 3.3.

text	**summary**	**title**
"By HOLLY RAMER, Associated Press CONCORD, N.H. – A sick American engineer who was successfully evacuated from the South Pole to New Zealand is awaiting the results of medical tests after having what doctors believed was a stroke in August. Renee-Nicole Douceur told The Associated Press in an email Tuesday that she had MRI and echocardiogram exams. She said results will be shared with doctors in the United States, "so everyone will be on the same page. ...."	"By HOLLY RAMER, Associated Press CONCORD, N.H. – A sick American engineer who was successfully evacuated from the South Pole to New Zealand is awaiting the results of medical tests after having what doctors believed was a stroke in August."	"Renee-Nicole Douceur Rescued: Sick South Pole Engineer Gets Tests For Possible Stroke"

Table 3.3: NEWSROOM example

3.4 Reuters rcv1

This dataset consists of more than 800000 Reuters News.

Its strength is that it contains a lot of financial news, and in this way it is possible to build a model suitable for financial-topics summarization.

Preprocessing

This dataset has to be requested at NIST (National Institute of Standards and Technology) and it requires some preprocessing, especially to ensure its compatibility with the purpose of this work.

Each *.xml* file presents some or all of the following fields:

- *title*, it is composed by a country code and the headline
- *headline*, headline of the news
- *byline*, author of the news
- *dateline*, date of the news
- *text*, body of the news
- *copyright*, copyright of the news
- *metadata*, additional information about the news, as the topics category, the industry category, the country

The first step of data cleaning consists in converting all the news in a new format where only *headline*, *text* and *metadata* fields are considered, and those elements which do not present one or more of these fields are discharged.

Then, in the second step, all that elements that have a *None* headline or do not belong to any of the financial topics categories identified (listed in table 3.4) are eliminated.

Finally, the resulting dataset is split in train (*70%*, 562110 records), valid (*15%*, 120452 records) and test (*15%*, 120453 records) sub-datasets.

For the sake of completeness, an example is reported in table 3.5.

Financial Code list	
"CURRENT NEWS - ECONOMICS	CURRENT NEWS - INSURANCE
CURRENT NEWS - BUSINESS NEWS	STRATEGY/PLANS
LEGAL/JUDICIAL	REGULATION/POLICY
SHARE LISTINGS	PERFORMANCE
ACCOUNTS/EARNINGS	ANNUAL RESULTS
COMMENT/FORECASTS	INSOLVENCY/LIQUIDITY
FUNDING/CAPITAL	SHARE CAPITAL
BONDS/DEBT ISSUES	LOANS/CREDITS
CREDIT RATINGS	OWNERSHIP CHANGES
MERGERS/ACQUISITIONS	ASSET TRANSFERS
PRIVATISATIONS	PRODUCTION/SERVICES
NEW PRODUCTS/SERVICES	RESEARCH/DEVELOPMENT
CAPACITY/FACILITIES	MARKETS/MARKETING
DOMESTIC MARKETS	EXTERNAL MARKETS
MARKET SHARE	CONTRACTS/ORDERS
DEFENCE CONTRACTS	MONOPOLIES/COMPETITION
MANAGEMENT	MANAGEMENT MOVES
LABOUR	CORPORATE/INDUSTRIAL
ECONOMIC PERFORMANCE	MONETARY/ECONOMIC
MONEY SUPPLY	INFLATION/PRICES
CONSUMER PRICES	WHOLESALE PRICES
CONSUMER FINANCE	PERSONAL INCOME
CONSUMER CREDIT	RETAIL SALES
GOVERNMENT FINANCE	EXPENDITURE/REVENUE
GOVERNMENT BORROWING	OUTPUT/CAPACITY
INDUSTRIAL PRODUCTION	CAPACITY UTILIZATION
INVENTORIES	EMPLOYMENT/LABOUR
UNEMPLOYMENT	TRADE/RESERVES
BALANCE OF PAYMENTS	MERCHANDISE TRADE
RESERVES	HOUSING STARTS
LEADING INDICATORS	ECONOMICS
SOCIAL AFFAIRS	EUROPEAN COMMUNITY
EC INTERNAL MARKET	EC CORPORATE POLICY
EC MONETARY/ECONOMIC	EC INSTITUTIONS
EC COMPETITION/SUBSIDY	GOVERNMENT/SOCIAL
LABOUR ISSUES	WELFARE, SOCIAL SERVICES
EQUITY MARKETS	BOND MARKETS
MONEY MARKETS	INTERBANK MARKETS
FOREX MARKETS	COMMODITY MARKETS
SOFT COMMODITIES	METALS TRADING
ENERGY MARKETS	MARKETS
EURO CURRENCY	"

Table 3.4: Table with financial topics list

text	**headline**
"The higher minimum wage signed into law Tuesday will be welcome relief for millions of workers, but it may also translate into higher prices for hamburgers, pizzas and other fast-food items, some restaurant chains said. The 90-cent-an-hour increase will have little short-term impact on many fast-food chains that already pay workers rates above the federally mandated minimum. But in the long run, industry officials fear that workers already earning above the new minimum wage, which will rise to ...."	"Chains may raise prices after minimum wage hike."

Table 3.5: Reuters example

Chapter 4

Related works

4.1 Overview

Many works have tried to face with text summarization task, adopting different methodologies:

- extractive approach
- abstractive approach
- hybrid approach

Extractive approach

Extractive text summarization directly identifies the most important sentences in the input text, and the final summary consists exactly in these selected phrases.

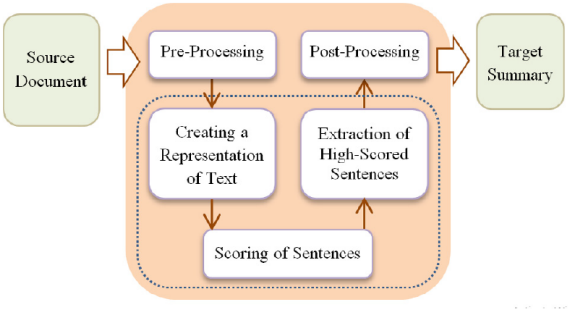

Figure 4.1: Extraction process [5]

Abstractive approach

Abstractive text summarization is a quite new proposed method which aims at generating summaries by an intermediate representation: it paraphrases the sentences with new words which may not be present in the original input.

The result is a summary more similar to the ones produced by humans.

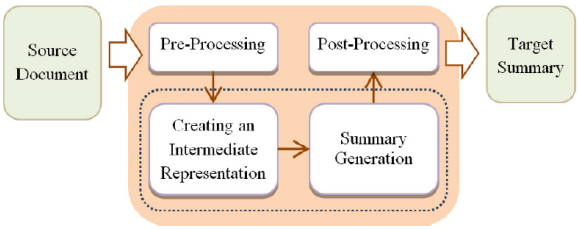

Figure 4.2: Abstraction process [5]

Hybrid approach

This method is a combination of extractive and abstractive techniques, and it is useful to make up for some weakness of the abstractive approach.

In fact, most of the well-known (and state of the art) abstractive techniques are not capable to work with documents having a size bigger than a specific upper bound.

This is the reason why, when a long document has to be addressed, it could be very useful to firstly apply an extractive phase to reduce its length, and feed this result to the abstractive algorithm, that it is now able to deal with the new smaller version.

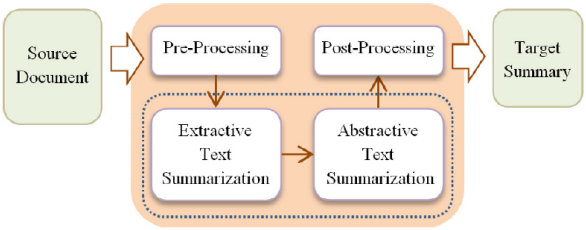

Figure 4.3: Hybrid process, defined in [5]

4.2 Fundamental models

4.2.1 TRANSFORMER

The basic structure on which most of the state of the art techniques rely, is the so called Transformer [6], so it is worth to mention this architecture.

It represents an alternative to RNN encoder-decoder architecture, and its strength relies on the possibility of parallelization (that allows to speed up the process), and on the adoption of a attention mechanism (to attend relevant textual units for context representation).

Architecture

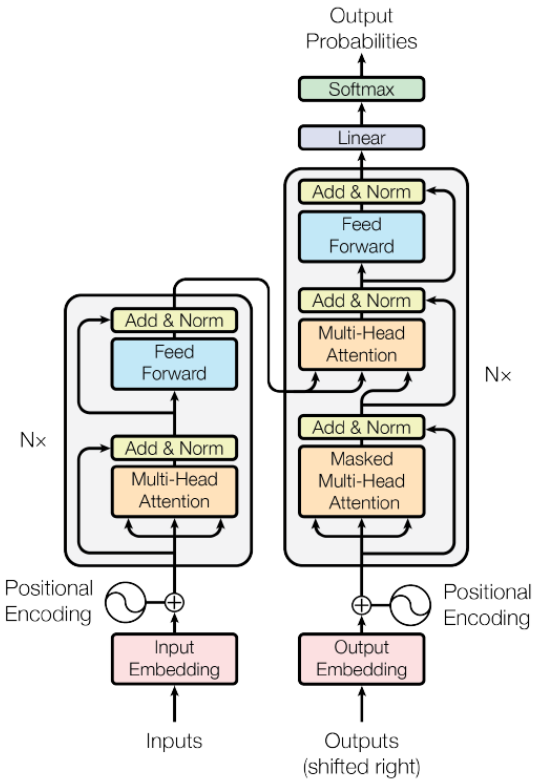

Figure 4.4: Transformers architecture, defined in [6]

Figure 4.4 presents the analyzed architecture. The encoder is made of a stack of 6 (N=6) identical layers, each of them composed by:

- a multi-head self-attention mechanism [6]

- a position-wise fully connected feed-forward network [6]

Meanwhile, the decoder is also made of a stack of 6 (*N=6*) identical layers, each of them composed by:

- a masked multi-head self-attention [6] (useful to guarantee that predictions for a certain position k are created using only the known outputs at positions less than k)
- a multi-head self-attention mechanism working on the encoder's output [6]
- a position-wise fully connected feed-forward network [6]

Each of the sub-layers, of the encoder and decoder, is supported by a residual connection plus a layer normalization.

Positional Encodings

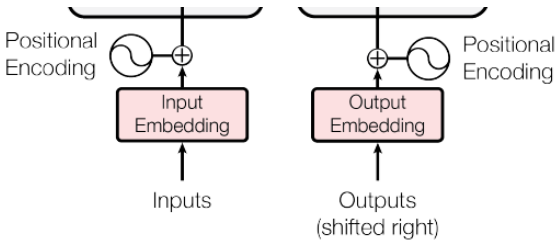

Figure 4.5: Positional Encodings [6]

The first problem that Transformer has to address is the capability of preserving the order of words in the sentences.

To face with this issue the initial step consists in adding **Positional Encodings** (figure 4.5) to the standard input. Specifically, these Positional Encodings are constructed through two sinusoidal functions which make aware the model with the information about the position of the token:

$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$ [6]

$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$ [6]

where pos is the position, and i the dimension.

Multi-Head Attention

The attention mechanism is a key component of Transformer architecture because it allows the model to focus on different part of the text. Specifically, when the encoder, or the decoder, is processing a token, the attention mechanism enables to consider also other words in the input sentence, in such a way to get an overall perspective of the entire text.

Namely, the attention improves the understanding capacity of the model: this mechanism allows to relate the analyzed word to the other words in the text.

The main component is the **Scaled Dot-Product Attention** (depicted in figure 4.6), which is articulated in some calculations:

- **MatMul**, a matrix dot-product between Query and Key. Namely,

$$MatMul(Q, K) = QK^T$$

- **Scale**, *MatMul* results are scaled by a factor

$$\sqrt{d_k}$$

  to avoid that large values obtained in the previous step mess up with softmax computation

- **Mask**, optional padding mask (Transformers assume that all the input vectors have the same fixed length, in case in which the sequence is longer, it is truncated, instead if it is shorter, it could be padded with zeros)

- **Softmax**, the results are finally passed to fit the interval *[0,1]*, as a probability distribution

The entire process performed in the **Scaled Dot-Product Attention** can be summarized as follows:

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

Going back to the overall multi-head attention mechanism 4.7, the input is split into h parts, each composed of Queries (Q), Keys (K) and Values (V) having a depth calculated according to:

$$d = d_{model}//h$$

In [6] the authors adopted *h=8*.

The three vectors are passed to the Scale dot product and then concatenated, resulting in 1 vector.

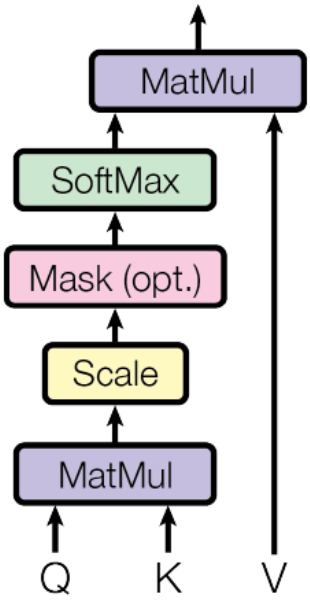

Figure 4.6: Scaled Dot-Product Attention [6]

Feed Forward

The last step consists in passing the resulting vector to a Feed-Forward neural network layer (depicted in figure 4.8), which is a two-layers linear transformation with a ReLU activation. This phase is important because it allows to convert the output from an attention layer, into a suitable form for the input of the following attention layer.

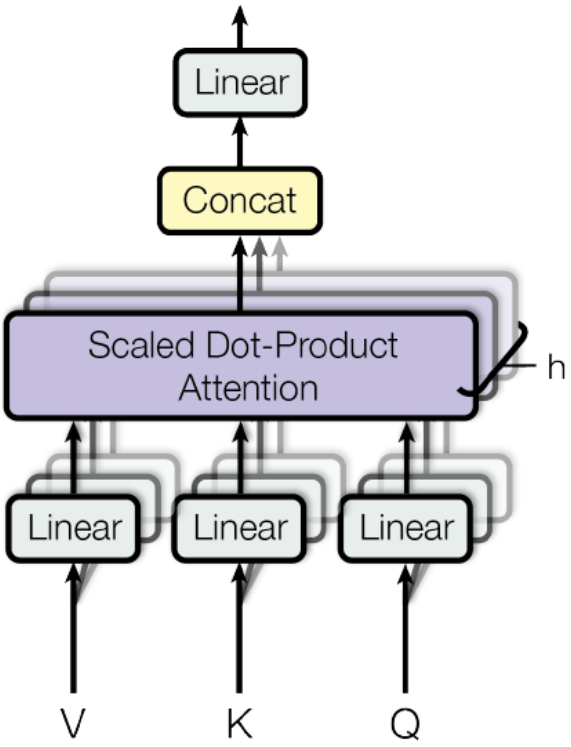

Figure 4.7: Multi-Head Attention [6]

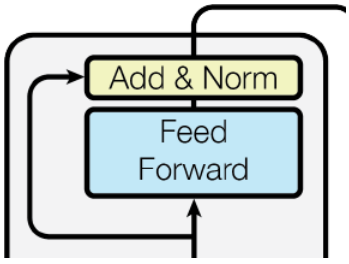

Figure 4.8: Feed Forward layer [6]

4.2.2 BERT

BERT, presented in the paper [7], is the most famous transformer-based sentence encoder.

BERT stands for Bidirectional Encoder Representations from Transformers, so as the name anticipates it is a multi-layer bidirectional encoder of Transformers architecture.

The training process is composed of two main steps: pre-training and fine-tuning. The former is a task-agnostic step, the latter instead is performed using specific data according to the downstream task.

Pre-training

Specifically, the pre-training is performed with two unsupervised tasks:

- **MLM**, Masked LM, is implemented according to the following procedure: 15% of the tokens are selected, 80% of them are replaced with *[MASK]* tokens, 10% remain unchanged, 10% are substituted with random tokens. The training consists in predicting only the masked words
- **NSP**, Next Sentence Prediction, is important to train the model at understanding the relations between sentences. This is performed thanks to a binary classifier, which is in charge of define if, given two selected sentences A and B (where B follows A in the 50% of the selected pairs), B is the actual sentence that follows A

Finetuning

Finally, there is finetuning. This step, starting from the parameters obtained in the pre-training, makes the model suitable for the specific task. This part is much less expensive with respect to the former step.

4.3 Extractive Techniques

4.3.1 MATCHSUM

This framework, presented in [8], proposes a new approach: instead of working at sentence-level, it looks at summary-level, in this way the task is treated as a semantic text matching problem.

General framework

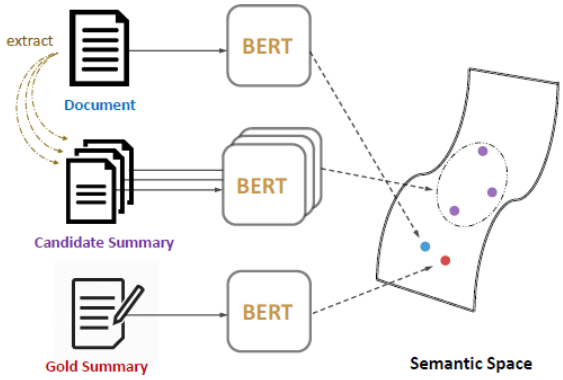

Figure 4.9: MatchSum framework [8]

Given D, the starting document, and C^*, its golden summary, C is a candidate summary.

The main idea is to get the semantic embedding of the document and each summary, using a BERT encoder.

Then, intuitively, the summaries that are more semantically similar to the document should be closer to its representation in the semantic space (figure 4.9).

The ROUGE of a candidate summary is calculated with respect to the golden summary through 2 levels:

- sentence-level score (average overlaps between each sentence of the candidate and the golden summary)
- summary-level score (similar to sentence-level, but here the sentences are considered as a whole)

After ROUGE calculation, some particular candidates are identified:

- **Pearl-Summary**, it is the summary with a higher summary-level score and a lower sentence-level score

- **Best-Summary**, it is the summary with the highest summary-level score

Researches found out that the probability that the best summary is a pearl-summary is directly related to the specific dataset, this suggests that the extractor should be decided according to the dataset under analysis.

Architecture

The adopted architecture is a Siamese-BERT based on pre-trained BERT: it is made of 2 BERTs with a cosine-similarity layer during the inference phase. Since it works at summary-level, BERT is used to create semantically meaningful embeddings of the document and the candidate summaries, to match the document D and the candidate summary C.

The finetuning is performed by two loss functions:

- *margin-based triplet loss*, it is based on the semantic similarity between the gold summary and the source document.

  $$L_1 = max(0, f(D, C) - f(D, C^*) + \lambda_1)$$

  where

  - $f(D, C) = cosine(r_D, r_C)$ is the function used to measure the similarity score between D and C
  - r_D is the embedding of D
  - λ_1 is a margin value

- *pairwise margin loss*, it takes into account all the candidate summaries.

  $$L_2 = max(0, F(D, C_j) - f(D, C_i) + (j - i)\lambda_2)$$

  where

  - $i < j$
  - $f(D, C_i) = cosine(r_D, r_{C_i})$ is the function used to measure the similarity score between D and C_i
  - r_D is the embedding of D
  - λ_2 is a parameter to highlight good and bad candidate summaries

And finally, the final loss results in:

$$L = L_1 + L_2$$

The inference step consists in looking for the best summary across the ones created from the starting document.

To face with the curse of combinations, a strategy is adopted to prune some candidates. Specifically, *PreSumm* algorithm is used to prune meaningless sentences.

Then, with the remaining sentences, all the possible combinations are analyzed.

4.4 Abstractive Techniques

4.4.1 BART

BART, proposed in [9], Bidirectional Auto-Regressive Transformers, is an architecture based on BERT and GPT.

With respect to BERT, the pretraining step consists of two phases: corruption of the text and reconstruction of it.

Architecture

Specifically, the architecture is made of a bidirectional encoder and a left-to-right autoregressive decoder, as shown in figure 4.10. The corrupted document is encoded by a bidirectional encoder, and then an autoregressive decoder is used to compute the likelihood of the original text.

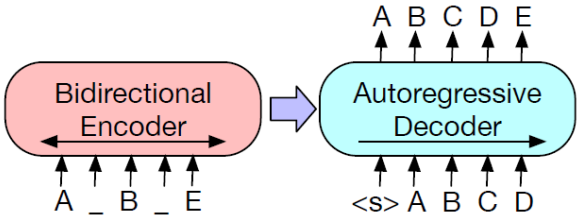

Figure 4.10: BART architecture [9]

This model is very close to BERT, but BART does not present the feed-forward network before word prediction, instead every single layer of the decoder computes cross-attention over the final hidden layer of the encoder.

As anticipated before, BART is pre-trained by a corruption function and then the entire document is reconstructed. Many typologies of noising functions are admitted, but it turns out that the best performances are reached when text infilling and sentence permutation are exploited.

Text infilling consists of sampling text spans (with a length according to a Poisson distribution, with parameter $\lambda = 3$), and replacing each of them with [MASK].

Sentence permutation instead involves the random shuffling of the sentences.

Fine-tuning phase depends on the downstream task analyzed; in case of summarization objective, the model is treated as a sequence-to-sequence model from the input to the output text.

4.4.2 PROPHETNET

This model, proposed in [10], presents an architecture based on Transformers, despite some modifications are implemented:

- different self-supervised objective: **future n-gram prediction**
- **n-stream self-attention** mechanism
- **mask based auto-encoder denoising task** for pre-training

Architecture

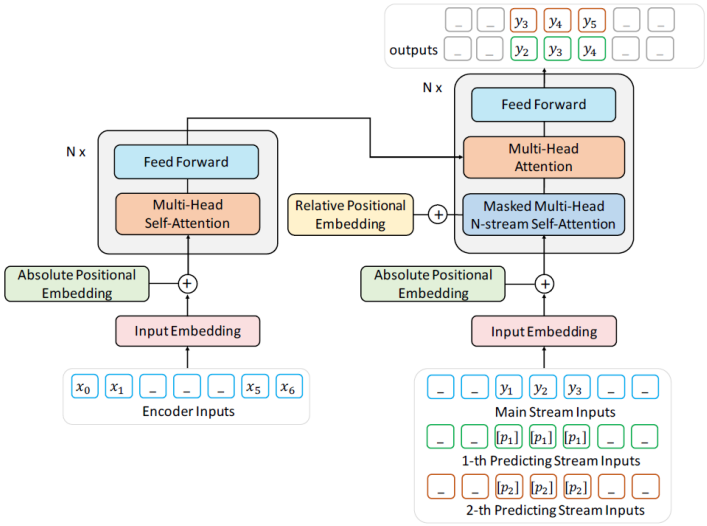

Figure 4.11: ProphetNet Architecture [10]

With reference to figure 4.11, on the left is possible to see the encoder and on the right the decoder. The encoder is unmodified with respect to the one proposed in the original Transformers architecture, instead the decoder is enriched with the *n-stream self-attention*.

The overall pre-training process consists in predicting the masked tokens, indicated by "_" in the figure, given the remaining tokens x_i.

Future N-gram Prediction *ProphetNet* looks at predicting at each time step t, the next continuous n future tokens.

So, given the input sequence, $x = x_1, ..., x_N$:

- the encoder encodes it: $H_{enc} = Encoder(x_1, ..., x_N)$

- the decoder predicts the future n-grams: $p(y_t|y_{<t}, x), ..., p(y_{t+n-1}|y_{<t}, x) = Decoder(y_t, H_{enc})$

N-Stream Self-Attention This mechanism is implemented, in addition to the standard *multi-head self-attention*, to allow the prediction of future tokens, previously described.

Namely, the k^{th} stream is in charge of predicting the probability $p(y_{t+k-1}|y_{<t}, x)$.

4.4.3 T5

T5 (Text-to-Text Transfer Transformer), described in [11], is a model based on *Transformer* architecture, and few changes have been made:

- the layer Norm bias is removed
- the layer normalization is outside the residual path
- it is used a relative positional embedding, instead of the fixed sinusoidal embedding

The main goal of this model is to set up a unified framework to allow that the same model and the same training procedure (maximum likelihood objective) can be applied to every task.

To do so, every text problem must be converted into a text-to-text format, as depicted in figure 4.12

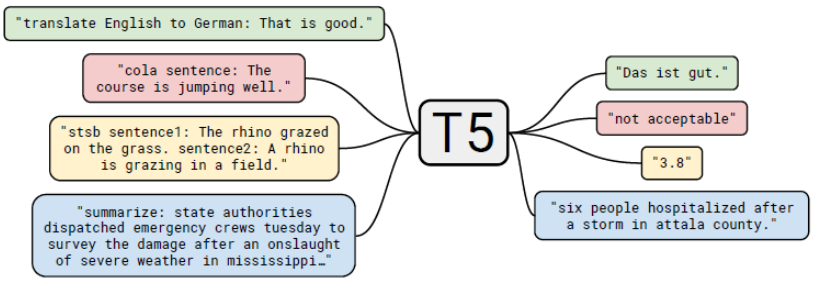

Figure 4.12: T5 [11]

This approach is justified by the effort to pre-train the model on a, as large as possible, dataset, to obtain general knowledge that can be transferred to specific-downstream tasks.

Furthermore, the pre-training is implemented throught a span corruption objective with a mean span length of 3 and corruption ratio of 15%.

Then, the model is finetuned according to the downstream task.

4.5 Hybrid Techniques

4.5.1 PGN

PGN, Pointer-Generator Network, was proposed in 2017 in the paper [12].

Architecture

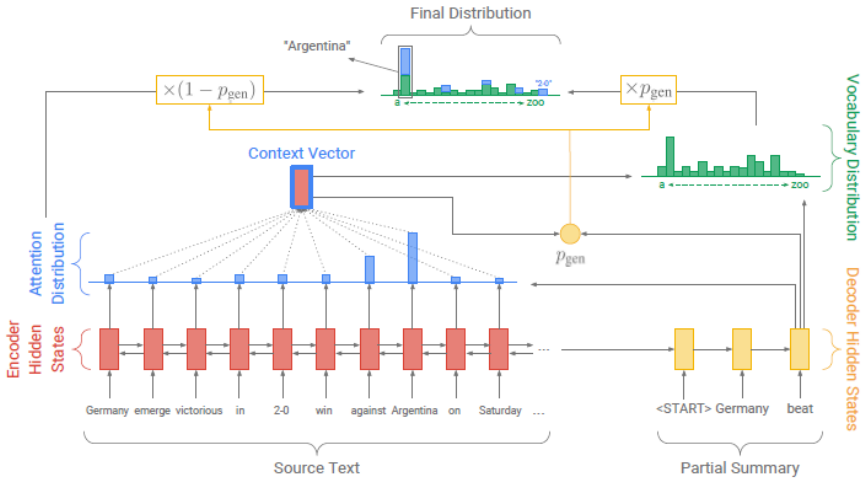

Figure 4.13: Model architecture [12]

The encoder is a single-layer bidirectional LSTM; the decoder is a single-layer unidirectional LSTM.

The process is depicted in figure 4.13. Once the input tokens are fed into the encoder, the *attention distribution* is computed:

$$a^t = softmax(e^t)$$

where:

- $e_i^t = v^T tanh(W_h h_i + W_s s_t + w_c c_i^t + b_{attn})$
- $v, W_h, W_s, x_c, b_{attn}$ are parameters to learn
- c^t indicates the coverage vector, which is in charge of overcoming the repetition problem. Namely, the coverage vector is the sum of the attention distributions over all previous decoder timesteps ($c^t = \sum_{t'=0}^{t-1} a^{t'}$)

which is used to calculate the *context vector*:

$$h_t^* = \sum_i a_i^t h_i$$

that is a representation of what has been seen from the input in the current step.

Furthermore, it is computed the *generation probability* p_{gen}:

$$p_{gen} = \sigma(w_{h^*}^T h_t^* + w_s^T s_t + w_x^T x_t + b_{ptr})$$

where:

- $w_{h^*}, w_s, w_x, b_{ptr}$ are learnable parameters

This probability is important because it allows to decide if the word will be:

- **generated** from the vocabulary (sampling from P_{vocab})
- **copied** from the input document (sampling from a^t)

Namely:

$$P(w) = p_{gen} P_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i=w} a_i^t$$

The attention distribution, calculated using the encoder and decoder hidden states, it is very useful since it indicates to the decoder where it has to look to produce the next word. Furthermore, it is used to calculate the Context Vector (representation of what has been read from the beginning until the current step).

4.5.2 SHEG

This framework, described in the paper [13], is articulated in three main steps: extractive phase, abstractive phase and headline generation phase.

Extractive step

It looks at the most salient sentences, by classifying them as *belonging* or *not belonging* to the final summary.

Specifically, an embedding matrix is constructed through *Word2vec* algorithm, and through a CNN the dependencies of close words are analyzed (sentence representation).

Then, there are two-layers bidirectional GRU. The former works at word level, the latter at sentence level.

Finally, it is performed the sentence-level classification: according to the quality of the sentence, it is included or not in the summary.

Abstractive step

The second step consists in generating a concise summary. It is introduced a novelty: the **CAC** (controlled actor-critic) which is needed to train the pointer generator network.

This architecture is implemented with a bidirectional LSTM encoder and a unidirectional LSTM decoder and a pointer network (to face OOV words).

Moreover, the CAC acts like an RL-agent, where the action is to choose the next token to insert in the summary, and the reward is based on ROUGE score

Headline generation

This phase is based on a **CRF** (Conditional Random Fields) model. This step is in charge of generating the headline according to a sequence prediction task.

It works by creating a feature function to map the outcome in a Euclidean space, and this feature vector takes into account many parameters like named entities and dependency features.

On Extractive and Abstractive Neural Document Summarization with Transformer Language Models

This work, presented in [14], faces with the challenge of long documents summarization and the overall process is articulated in two steps: extractive and abstractive one.

The former is implemented with two hierarchical models: on based on a sentence classifier and the other one on pointer network.

The latter is made of a transformer language model, conditioned by the results of the previous step.

Extractive phase

Sentence Classifier An encoder hierarchical LSTM is used to produce the document representation.

Then the classification is performed by computing the probability that each sequence belongs or not to the summary.

Pointer Nerwork Hierarchical Seq2Seq Sentence Pointer has an encoder-decoder architecture.

The sentence-encoder, consisting in a bi-directional LSTM and working at token-level, aims at encoding every sentence.

The document-encoder, consisting in a bi-directional LSTM and working at sentence-level, looks at encoding the generated embedded sentences.

The decoder, made of an autoregressive LSTM, predicts the next extracted sentence relying on the hidden state of the document-encoder related to the last extracted sentence, through an attention mechanism.

This produces the so-called attention aware hidden state which is added to the input of the following time step.

Abstractive phase

To solve the challenge of dealing with long documents, that a Transformer cannot support, the summary is generating starting from the introduction of the document, and the extracted sentences are used as conditioning text.

Combination of abstractive and extractive approaches for summarization of long scientific texts

Also [15] deals with the summarization of long documents, and solves this problem adopting an hybrid technique.

First of all, the most important sentences are selected by extractive summarization (implemented as a classification problem), then the obtained result is concatenated to other article parts, and it is all used as starting point for the final summary generation.

Specifically, the best performances are obtained when the abstraction is performed over the extracted sentences concatenated with the introduction and the conclusion of the scientific paper.

Looking further into the overall structure of the model, there are two architectures: the classifier for the first step and the abstractive model for the second one, both implemented with pre-trained transformer-based LMs.

Extractive phase

The authors tested different architecture for the extractive step: BERT, ELECTRA and RoBERTa. The best results were achieved using BERT.

An interesting intuition made by the authors was to use the back-translation technique to paraphrase the sentences: exploiting a pre-trained transformer, the extracted sentences are translated from English to German, and then the procedure is repeated by translating from German to English.

Abstractive phase

The abstractive phase was implemented with GPT-2 and BART transformers, the best performance was obtained using BART.

In GPT-2 the abstractive summary is generated starting from the extracted sentences and the paraphrased sentences. Specifically, the input to the model is the result of the concatenation of the conditioning text t_c, and the target summary t_s:

$$input_{abs} = concat(t_c, t_s) * mask_{segment}$$

Where the mask is needed to allow to distinguish which part is the conditioning text and which the target summary.

In BART instead the conditioning sentences are used to feed the encoder, and the target sentences to fed the decoder.

Chapter 5

Proposed model

5.1 Overview

The approach proposed in this thesis work is an hybrid technique. Namely, it is articulated in two steps:

- *Extractive summarization* implemented with **PreSumm** [16]
- *Abstractive Summarization* implemented with **PEGASUS** [17]

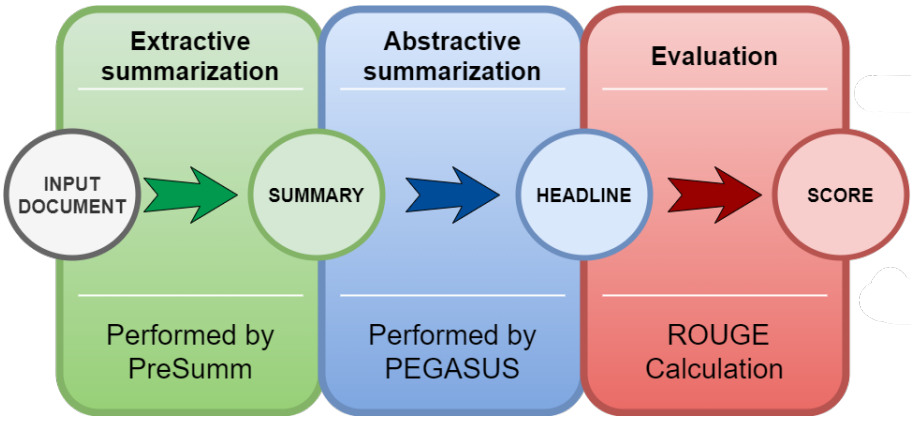

Figure 5.1: Overview of the pipeline implemented in this work

5.2 Extractive Summarization - PreSumm

The former step consists in selecting the most salient sentences of the input document to obtain an intermediate summary.

Namely, this process can be interpreted as a binary classifier, where the assigned label indicates if a sentence should be incorporated in the summary or not.

The main reason to perform this intermediate step is to filter the "useless" information contained in the document, in such a way to feed the abstractive phase with a shorter input, to overcome the limitations related to the length of the input, which affects *Transformers* performances.

This task is performed through *PreSumm* model, and the exploited version is the one offered by [18].

5.2.1 Overview

PreSumm, proposed in [16], is based on BERT, Bidirectional Encoder Representations from Transformers, presented in section 4. However, some modifications are done in order to make the model suitable for the summarization task. In fact, *PreSumm* is a document-level encoder, deployed for document summarization.

5.2.2 Main variations to BERT model

Sentence-level representation

To get a model which is capable of manipulating sentence-level representations (instead of token-level ones as in *BERT*), each sentence is delimited by *[CLS]* and *[SEP]* tokens.

Interval segment embeddings

Moreover, it is keeping track of multiple sentences through *interval segment* embeddings: given a document D containing sentences

$$[sent_1, sent_2, sent_3, sent_4, sent_5, sent_6, sent_7]$$

each sentences will be embedded as E_A or E_B depending on whether it is a odd or even sentence. So the document D will be embedded as

$$[E_A, E_B, E_A, E_B, E_A, E_B, E_A]$$

. This will allow a hierarchical learning of the document representation:

- lower layers in the architecture will focus on closer sentences
- higher layers in the architecture will focus on multi-sentences (supported by self-attention)

Position embeddings

In *BERT*, the model is bounded by a maximum length of 512 for positional embedding; *PreSumm* overcomes this restriction by adding some position embeddings randomly initialized and finetuned subsequently.

5.2.3 Architecture

The architecture is composed by a *BERT* layer on top of which are stacked several Transformer layers.

The *BERT* layer outputs a sentence vector T, as can be seen in figure 5.2, which is then fed to the inter-sentence Transformer layers, which are in charge of capturing document-level features.

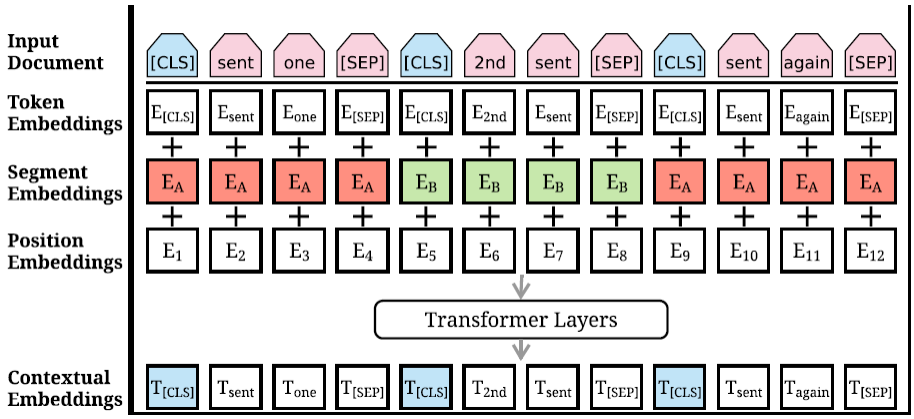

Figure 5.2: PreSumm Architecture, defined in [16]

Specifically, the computations performed are:

$$\tilde{h}^l = LN(h^{l-1} + MHAtt(h^{l-1}))$$

$$h^l = LN(\tilde{h}^l + FFN(\tilde{h}^l))$$

where:

- $h^0 = PosEmb(T)$
- T is the output of the *BERT* layer, previously mentioned
- *PosEmb* is used to adjust T with a sinusoidal positional embedding (useful to highlight the position of each sentence)
- *LN* is the layer normalization operation

- *MHAtt* is the multi-head attention operation
- *FFN* is the feed forward layer operation

Finally, the last layer is a sigmoid classifier:

$$\hat{y}_i = \sigma(W_o h_i^L + b_o)$$

where:

- h_i^L is the output of the top Transformer layer for sentence i. (The best performances were reached for $L = 2$)

The loss exploited to train the model is a binary classification entropy.

Binary Classification Entropy Loss

This loss typology allows to measure the entropy between the prediction ($\hat{y}_i$) and the gold label (y_i^*). Namely, it is calculated:

$$l(\hat{y}_i, y_i^*) = -w_i[y_i^* log\hat{y}_i + (1 - y_i^*)log(1 - \hat{y}_i)]$$

where w_i is the rescaling weight assigned to each element.

5.3 Abstractive Summarization - PEGASUS

The abstractive summarization phase consists in elaborating the summary obtained in the former step, creating a concise sentence to enclose its main topic: the headline.

The peculiarity of abstractive methods is the capability of paraphrasing: the obtained result is not an extracted sentence from the input document, but a complete new text which contains the main idea of the input document (with also novel words).

5.3.1 Overview

PEGASUS, proposed in [17], differs from the original Transformer model because it is trained with a self-supervised objective tailored for abstractive text summarization, which consists in masking whole sentences and generate these phrases starting from the reaming sentences in the document. This approach is called **GSG**, that stands for Gap Sentences Generation.

So, in contrast to other main methods based on Transformer architecture, *PEGASUS*:

- masks entire sentences rather than single tokens
- reconstructs only the masked sentences rather than the entire document

5.3.2 Architecture

As anticipated, it is a Transformer-based encoder-decoder model, as depicted in figure 5.3.

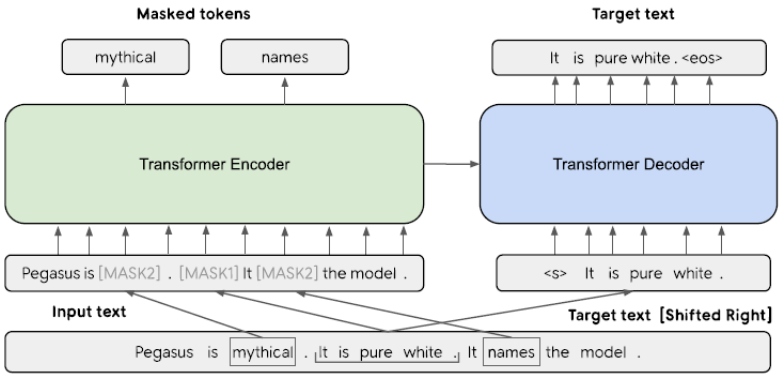

Figure 5.3: PEGASUS Architecture, defined in [17]

GSG

It is supposed that **GSG**, Gap Sentences Generation, works well as pre-training objective because it recalls the abstractive summarization task.

An important parameter of *GSG* is **GSR**, Gap Sentences Ratio, that indicates the percentage of selected sentences (the masked ones) with respect to the total number of sentences in the input text.

This mechanism consists in replacing each masked sentence with *[MASK1]* token, taking into account that those selected sentences are the ones which seem to be the more important to the document.

To identify the principal sentences, *PEGASUS* was tested on selecting them according to two modalities:

- **Independently**, score is calculated between the sentence and the rest of the document:

$$score_{sentence_i} = rouge(sentence_i, D \setminus \{sentence_i\})$$

  where "$D \setminus \{sentence_i\}$" indicates the whole document with except of the analyzed sentence. The k sentences with the highest scores are selected.

- **Sequentially**, where k sentences are iteratively selected according to the following approach:

  $S = \emptyset$
  for i $\leftarrow$ 1 to k **do**
  $s_i = rouge(S \cup \{x_i\}, D \setminus (S \cup \{x_i\}))$ $\quad \forall i, i \notin S$
  $j = argmax_i\{s_i\}_n$
  $S = S \cup \{x_k\}$
  end for

Moreover, for the rouge calculation, the n-grams could be considered as a set (*Uniq* variant) or not (*Orig* variant).

The best performances are reached with **Indipendetly-Orig** configuration, that is the one implemented in the final model.

5.4 Performance Evaluation

5.4.1 Rouge Score

To validate the goodness of the implemented pipeline, the obtained results are evaluated through *rouge score* calculation, exploiting the version made available by [19]. Thanks to this implementation, it is possible to easily compute precision, recall and accuracy related to the rouge metrics.

In fact, it is unfeasible to manually evaluate all the generated summaries and headlines, and so an automatic approach must be adopted.

ROUGE, *Recall-Oriented Understudy for Gisting Evaluation*, was proposed by [20] and it is an instrument that allows to understand the goodness of a generated summary, by comparing it with a reference summary, generally created by humans.

ROUGE offers different metrics, and the ones considered in this evaluation are:

- ROUGE-1
- ROUGE-2
- ROUGE-L

ROUGE-1 and ROUGE-2

These two measures belong to ROUGE-N category, which takes into account the overlapping of N-grams (1-gram for ROUGE-1, 2-grams for ROUGE-2) between the generated summary and the reference one.

Formally:

$$ROUGE - N_{recall} = \frac{\sum_{S \in ReferenceSummaries} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S \in ReferenceSummaries} \sum_{gram_n \in S} Count(gram_n)}$$

where:

- n is the length of the n-gram
- $Count_{match}(gram_n)$ is the maximum number of n-grams which occur both in the generated summary and in the reference one

The precision instead can be computed considering at the denominator the total number of n-grams in the generated summary (instead of the total number of n-grams in the reference summary).

Finally, the f-score is calculated as:

$$F = \frac{2 \times Recall \times Precision}{Recall + Precision}$$

ROUGE-L

This measure instead relies on the LCS, longest common sub-sequence. The benefit of this metric is that no grams length has to be defined in advance, and it relies on in-sequence matches instead of consecutive matches.

Given a generated summary Y and a reference summary X, both made of one sentence:

$$R_{lcs} = \frac{LCS(X,Y)}{m}$$

$$P_{lcs} = \frac{LCS(X,Y)}{n}$$

$$F_{lcs} = \frac{2 \times R_{lcs} \times P_{lcs}}{R_{lcs} P_{lcs}}$$

where:

- m is the length of the sentence in the reference summary
- n is the length of the sentence in the generated summary

When the summaries present more than one sentence, the formulas previously described must be adapted to:

$$R_{lcs} = \frac{\sum_{i=1}^{u} LCS_{\cup}(r_i, C)}{m}$$

$$P_{lcs} = \frac{\sum_{i=1}^{u} LCS_{\cup}(r_i, C)}{n}$$

where:

- u is the number of sentences in the reference summary
- r_i is a sentence in the reference summary
- C is the whole set of sentences in the generated summary

5.4.2 BERTScore

For the sake of completeness, the results obtained from the implemented pipelines are evaluated also using *BERTScore*, proposed by [21].

BERTScore is based on BERT, it computes the cosine similarity between the embeddings of the reference summary and the embedding of the candidate one. The metrics which can be obtained are: precision, recall and F1 measure.

Score computation

In [21] the authors propose a specific approach to calculate *BERTScore*:

- BERT model is used to compute the contextual embeddings for the tokens in candidate and reference summaries (or headlines). In this way a vector representation is obtained: $\hat{x}$ represents the vector referred to the candidate summary embedding, x to the reference summary embedding
- metrics are computed, based on cosine similarity:
  - $R_{BERT} = \frac{1}{|x|} \sum_{x_i \in x} max_{\hat{x_j} \in \hat{x}} x_i^T \hat{x}_j$
    Namely, it corresponds to the cardinality of the set of the tokens in x that match with tokens in $\hat{x}$
  - $P_{BERT} = \frac{1}{|\hat{x}|} \sum_{\hat{x_j} \in \hat{x}} max_{x_i \in x} x_i^T \hat{x}_j$
    Namely, it corresponds to the cardinality of the set of the tokens in $\hat{x}$ that match with tokens in x
  - $F1_{BERT} = \frac{2 \times P_{BERT} \times R_{BERT}}{P_{BERT} \times R_{BERT}}$

  Because vectors are pre-normalized, the cosine similarity between a token x_i and a candidate token $\hat{x}_j$ can be directly computed through the inner product: $x_i^T \hat{x}_j$
- precision and recall are further improved by considering also the IDF scores: to adjust the metrics according to the importance of each word
- each score is scaled in order to belong to the interval [0,1]

Chapter 6

Experimental Design

6.1 Introduction

Let now us deepen the analysis of the conducted experiments.

Firstly, the state-of-the-art models are tested, exploiting the public available checkpoints. This is done to get the reliability of each model, and how much it is suitable to be incorporated in the final pipeline. This research phase has involved:

- for the extraction step:
  - PreSumm
  - MatchSum
- for the abstraction step:
  - BART
  - PEGASUS
  - PGN
  - PROPHETNET
  - T5

Then, once the best models are identified, the framework is set up, trained and test, before on *Newsroom* dataset, then on *Reuters*.

6.2 Candidate models evaluation

In this phase no training is performed, simply the previously listed models are tested with regards to the summary generation task exploiting the public available checkpoints.

6.2.1 Screening for the Extraction step

MatchSum It is exploited the code released in [22], with the provided checkpoints for the pre-trained model on *CNN/DM* dataset for summary generation task.

PreSumm It is used the publication in [18], specifically the pre-trained model on *CNN/DM* dataset for summary generation task.

Results

The summaries produced are evaluated with *ROUGE* score, especially focusing on *F-score* metric, and the registered performances are depicted in figure 6.1.

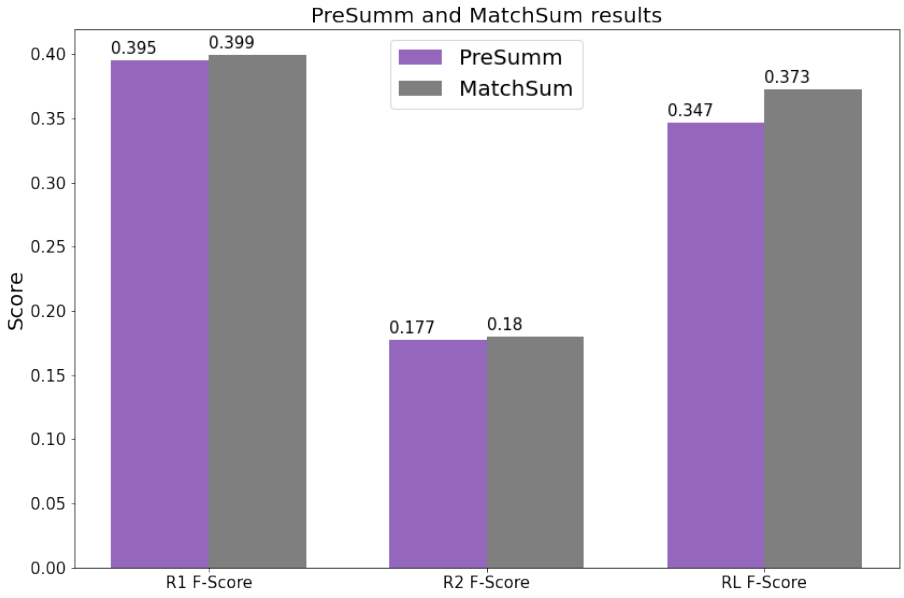

Figure 6.1: Results on *PreSumm* and *MatchSum*, ROUGE Score

As shown in 6.1, the two models present very similar results. Despite *MatchSum* has slightly better performances, in the final pipeline it has been chosen to use *PreSumm* since the public code released by the author could be finetuned on specific dataset, and this was not possible on *MatchSum*.

6.2.2 Screening for the Abstraction step

BART, PEGASUS, PROPHETNET, T5 For all of them, it is adopted the *Huggingface* version, and the corresponding available checkpoints on *CNN/DM* and also *NEWSROOM* for *BART* and *PEGASUS* [23], [24], [25], [26].

PGN It is used the publication in [27], specifically the pre-trained model on *CNN/DM* dataset for summary generation task.

Results

The summaries produced are evaluated with *ROUGE* score, especially focusing on *F-score* metric, and the registered performances are depicted in figures 6.2 and 6.3.

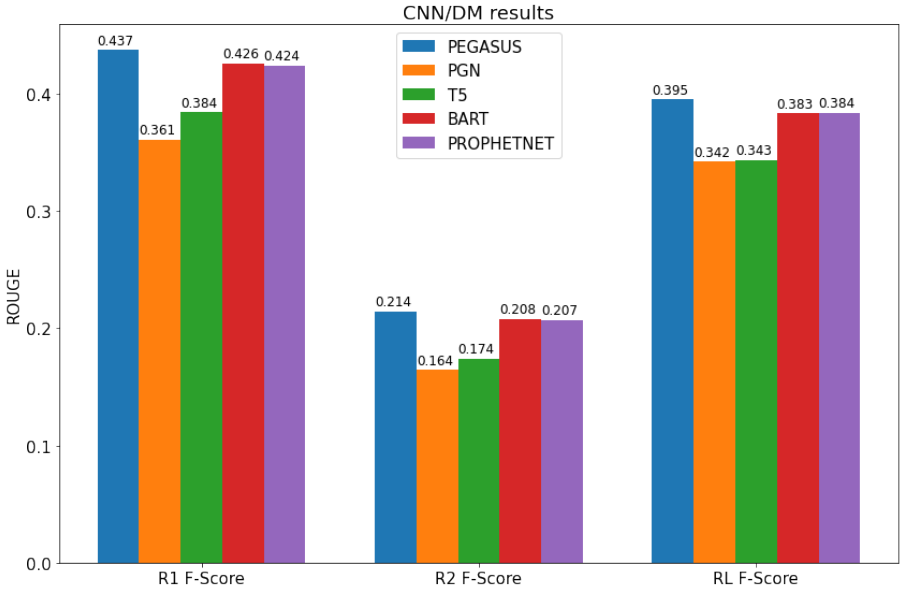

Figure 6.2: Results on *CNN/DM*, ROUGE Score

As shown in figures 6.2 and 6.3, *PEGASUS* outperforms the other models, and because of this, it will be incorporated in the final pipeline to implement the abstraction phase.

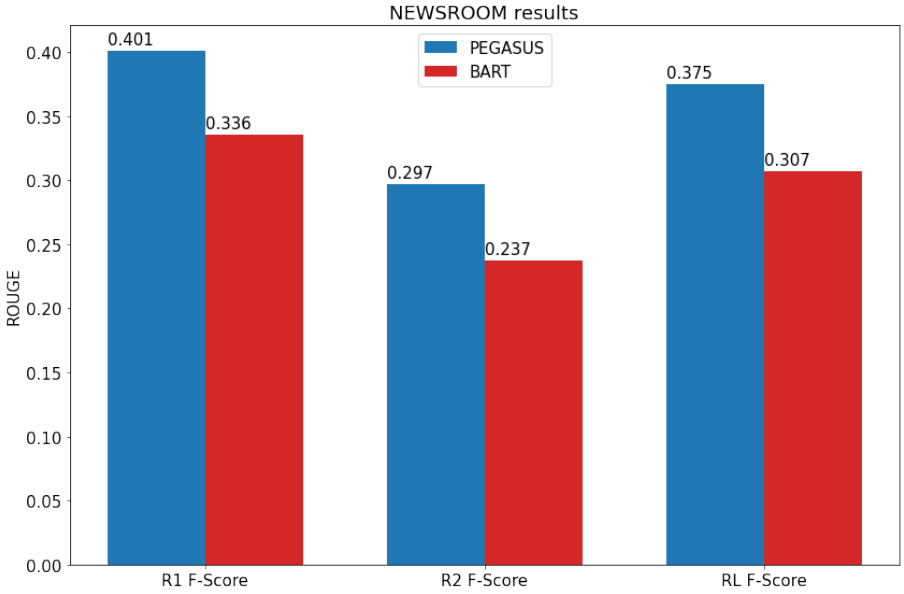

Figure 6.3: Results on *NEWSROOM*, ROUGE Score

6.3 Extractive step

Different configurations were tested, especially for the first phase, the extractive one.

This because the *Reuters* dataset, does not provide *reference summaries*, and so it was necessary to examine different alternatives to overcome this issue. Namely, it was required to identify a procedure capable of generating *reference summaries* to train (or finetune) *PreSumm* algorithm (self-supervised approach), or a protocol that could be robust enough to replace *PreSumm* in this first step.

For this reason, the *extraction* was performed in different ways:

- selection of random k sentences
- selection of lead k sentences
- selection done by *Text Rank* algorithm
- *PreSumm*

6.3.1 Random k sentences

This approach is implemented to have a baseline, and to understand how much the performances are improved if a more accurate procedure is adopted.

6.3.2 Lead k sentences

This method comes up from what suggested in the paper [15], where it was demonstrated that relevant information was already incorporated in the initial part of the document. So the attempt is made also in this project, to understand if it could be a good compromise for the creation of the starting text from which the headlines will be generated.

6.3.3 Text Rank algorithm

The last approach analyzed consists in the *TextRank* algorithm, and the exploited version is the one offered by [28].

This model, based on *Google's PageRank* algorithm and presented in the paper [29], is a weighted graph-based model.

Namely, each sentence is represented as a vertex in a graph, and the oriented edges are the connections between the sentences. The importance of a vertex is directly related to the number of vertexes that vote for it, weighted by the importance of the vertex that has voted. Specifically, a vertex A is voting for vertex B if in the graph there is an edge that starts from A and end up into B.

So, the score for a vertex V_i can be computed according to the following formula:

$$WS(V_i) = (1-d) + d \sum_{V_j \in In(V_i)} \frac{w_{j,i}}{\sum_{V_k \in Out(Vj) w_{j,k}}} WS(V_j)$$

where:

- d is a damping factor, $d \subset [0,1]$, which is used to incorporate the probability of random jumps in the vertexes graph
- $In(V_i)$ is the subset of vertexes that are voting for vertex i
- $Out(V_j)$ is the subset of vertexes that vertex j is voting for
- $w_{j,k}$ is the weight of the edge between j and k, it indicates the power of the connection between the two vertexes

The ranking process is articulated as follows:

- all the vertex weights are randomly initialized (and this won't influence the reliability of the final result, it may only change the number of iterations needed to reach the convergence)
- the calculations are iteratively performed until a given threshold related to the error rate of the vertexes is satisfied

To apply this algorithm to extract sentences, the first things to do it is to build a graph where vertexes represent sentences, and edges connect sentences with a common concept. Namely, an edge is present only between two similar sentences, where the similarity is calculated according to:

$$Similarity(S_i, S_j) = \frac{|\{w_k | w_k \in S_i \& w_k \in S_j\}|}{log(|S_i|) + log/(|S_j|)}$$

where:

- S_i indicates the sentence i
- $S_i = w_1^i, w_2^i, ..., w_N^i$

Therefore, once the graph is designed, it will be sufficient to run the algorithm until the convergence is reached, and then selecting the top k detected salient sentences.

6.4 Experiments on Newsroom

In this section, it will be deepened the approach adopted to conduct the experiments, in such a way to guarantee the reproducibility.

Computational resources provided by *hpc@polito*, which is a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino (`http://hpc.polito.it`)

6.4.1 Tested pipelines

Random *k* sentences and *Pegasus* The extractive summary is obtained by selecting k sentences randomly, the seed used to fix the randomness and guarantee the replicability is *42*. The abstractive headline is computed by *Pegasus*. Different configurations were tested:

- *Random **3** sentences*, *Pegasus* parameters: ***lr*** $= 2 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*.

  Configuration name: **Random-3A**

- *Random **3** sentences*, *Pegasus* parameters: ***lr*** $= 2 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 80*, ***max target length*** *= 25*, *average target length = 10*

  Configuration name: **Random-3B**

- *Random **3** sentences*, *Pegasus* parameters: ***lr*** $= 5 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*

  Configuration name: **Random-3C**

- *Random **5** sentences*, *Pegasus* parameters: ***lr*** $= 2 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*

  Configuration name: **Random-5A**

Lead *k* sentences and *Pegasus* The extractive summary is obtained by selecting the first k sentences. The abstractive headline is computed by *Pegasus*. Different configurations were tested:

- *Lead **3** sentences*, *Pegasus* parameters: ***lr*** $= 2 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*

Configuration name: **Lead-3A**

- *Lead **3** sentences*, *Pegasus* parameters: ***lr*** $= 2 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 80*, ***max target length*** *= 25*, *average target length = 10*

  Configuration name: **Lead-3B**

- *Lead **3** sentences*, *Pegasus* parameters: ***lr*** $= 5 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*

  Configuration name: **Lead-3C**

- *Lead **5** sentences*, *Pegasus* parameters: ***lr*** $= 2 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*

  Configuration name: **Lead-5A**

- *Lead **5** sentences*, *Pegasus* parameters: ***lr*** $= 5 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*

  Configuration name: **Lead-5B**

TextRank* and *Pegasus The extractive summary is obtained thanks to *TextRank* algorithm, which looks at selecting the *3* most salient sentences in the text. The abstractive headline is computed by *Pegasus*, trained with the following parameters: ***lr*** $= 2 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*.

PreSumm* and *Pegasus The extractive summary is obtained thanks to *PreSumm* algorithm, which looks at selecting the *3* most salient sentences in the text. The abstractive headline is computed by *Pegasus*, trained with the following parameters: ***lr*** $= 2 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*.

6.4.2 Results

Delving into the performance of the different implemented pipelines, the major considerations that can be made are:

- the extraction phase, used to generate the summaries, highlights that *Lead k-sentences selection* and *PreSumm* achieve good results, especially if compared

with the performance declared in [13] (*SHEG* framework, previously discussed), where the authors registered the following scores on summaries generation process:

  - *R1-Fscore = 0.282* (**Lead-3C**, *R1-Fscore = 0.33639*)
  - *R2-Fscore = 0.147* (**Lead-3C**, *R2-Fscore = 0.24367*)
  - *RL-Fscore = 0.259* (**Lead-3C**, *RL-Fscore = 0.30972*)

  From this consideration, it can be derived that the salient information is already present in the initial part of the text (otherwise *firs-k selection* wouldn't work so well), and this could be directly related to the nature of the dataset: news articles aim at reporting as soon as possible the more interesting information, to capture the reader attention. This let us consider that, despite also *PreSumm* achieved very good results, in case of news dataset, the best approach is the one of selecting the first k sentences. In fact, in this way, no training is needed, the process is speeded up and good results are obtained without consuming a lot of computational resources

- the abstraction phase, points out that *PEGASUS* algorithm works better with specific typologies of summaries generation process. In fact, as anticipated, *Lead k-sentences selection* and *PreSumm* present similar performances about summaries generation, but the headlines generated by *PEGASUS* are different in terms of computed scoring metrics. In fact, the headlines generated from the summaries obtained by *Lead k-sentences selection* present higher rouge scores with respect to the headlines generated from *PreSumm* results. Furthermore, the rouge scores registered from the headlines generated by the configuration **Lead-3C** are much more satisfactory than the ones reported in *SHEG* paper, [13], which are:

  - *R1-Fscore = 0.1381* (**Lead-3C**, *R1-Fscore = 0.32042*)
  - *R2-Fscore = 0.0794* (**Lead-3C**, *R2-Fscore = 0.13209*)
  - *RL-Fscore = 0.1142* (**Lead-3C**, *RL-Fscore = 0.29426*)

To examine the specific performance metrics, please refer to the following sections where are reported the values calculated through *ROUGE Score* and *BERTScore*.

ROUGE Score

The obtained results are summarized in the following tables: 6.1, 6.2.

A comparison can be done also through these charts: 6.4,6.5,6.6,6.7,6.8,6.9.

Summaries generation			
Configuration	R1-Fscore	R2-Fscore	RL-Fscore
Random-3A	0.1902	0.09652	0.16618
Random-3B	0.1902	0.09652	0.16618
Random-3C	0.1902	0.09652	0.16618
Random-5A	0.22858	0.13363	0.20311
Lead-3A	0.33639	0.24367	0.30972
Lead-3B	0.33639	0.24367	0.30972
Lead-3C	**0.33639**	**0.24367**	**0.30972**
Lead-5A	0.33688	0.24409	0.30986
Lead-5B	0.33688	0.24409	0.30986
TextRank and Pegasus	0.21134	0.11524	0.18499
PreSumm and Pegasus	0.33423	0.23356	0.30103

Table 6.1: Results on Newsroom, summaries generation, ROUGE Score

Headlines generation			
Configuration	R1-Fscore	R2-Fscore	RL-Fscore
Random-3A	0.24461	0.08725	0.22572
Random-3B	0.22652	0.07603	0.20516
Random-3C	0.24635	0.08812	0.22713
Random-5A	0.26807	0.1011	0.24701
Lead-3A	0.31362	0.12882	0.28814
Lead-3B	0.29149	0.11294	0.26166
Lead-3C	**0.32042**	**0.13209**	**0.29426**
Lead-5A	0.31926	0.13164	0.29328
Lead-5B	0.32258	0.13351	0.29628
TextRank and Pegasus	0.26737	0.10015	0.24591
PreSumm and Pegasus	0.28868	0.11343	0.26444

Table 6.2: Results on Newsroom, headlines generation, ROUGE Score

BERTScore

The computed BERTScores on the obtained results are summarized in the following tables: 6.3, 6.4.

In light of the obtained results, especially for Precision BERTScore, a consideration has to be done. As anticipated, in the section where BERTScore was presented

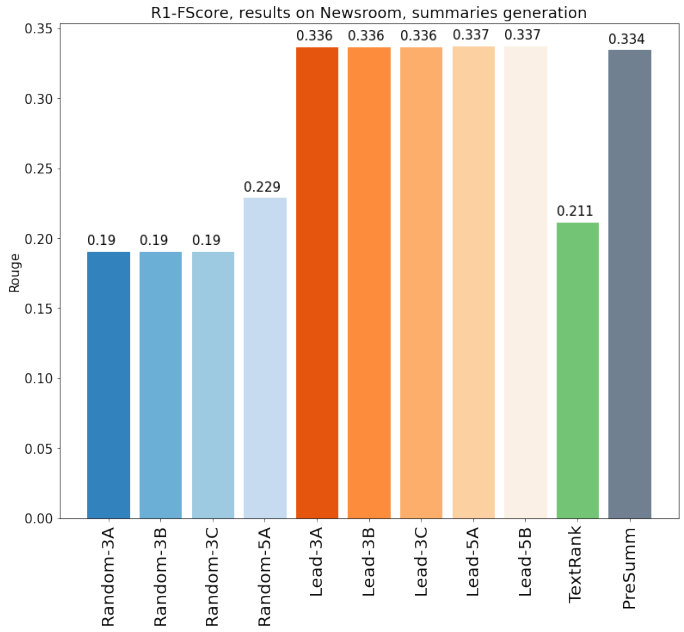

Figure 6.4: R1-Fscore for summary generation task performed on *Newsroom*, ROUGE Score

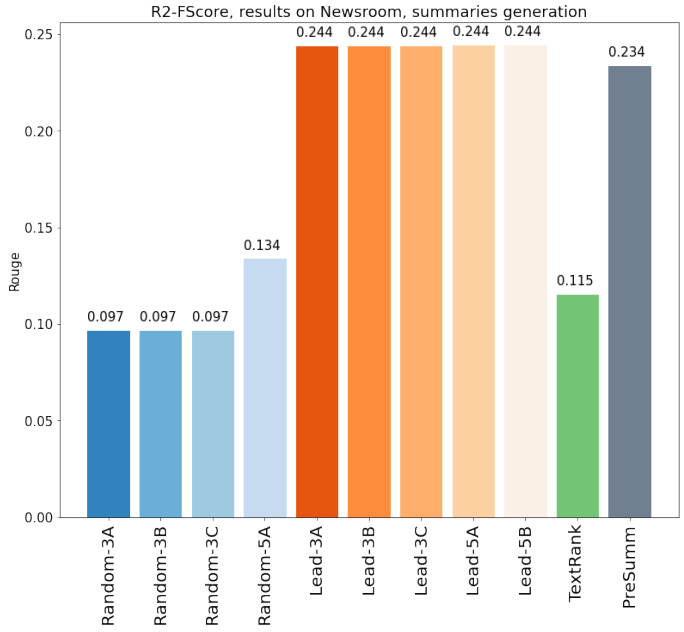

Figure 6.5: R2-Fscore for summary generation task performed on *Newsroom*, ROUGE Score

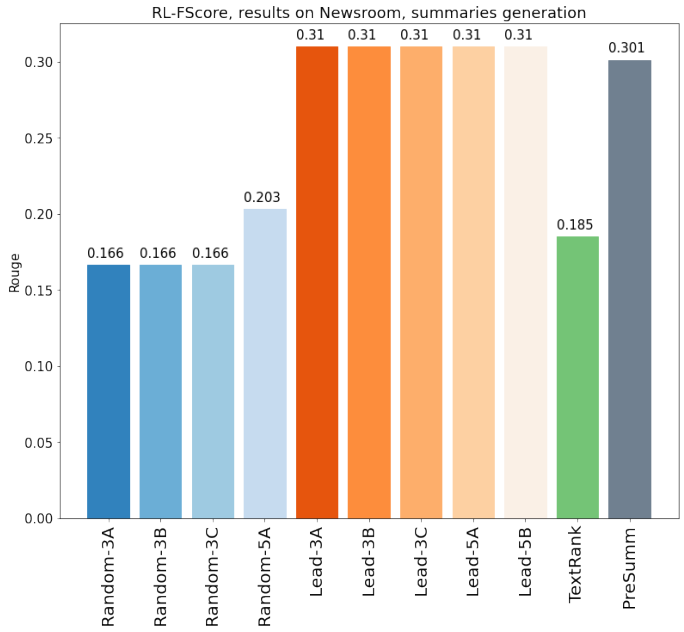

Figure 6.6: RL-Fscore for summary generation task performed on *Newsroom*, ROUGE Score

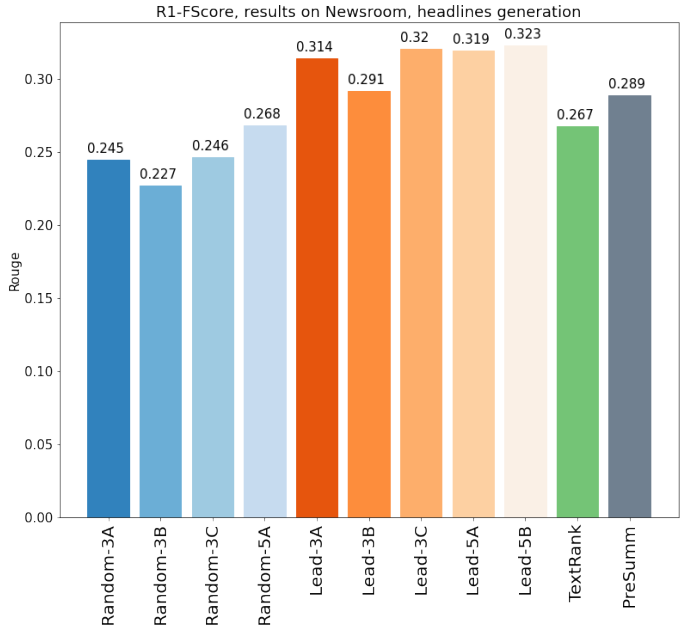

Figure 6.7: R1-Fscore for headline generation task performed on *Newsroom*, ROUGE Score

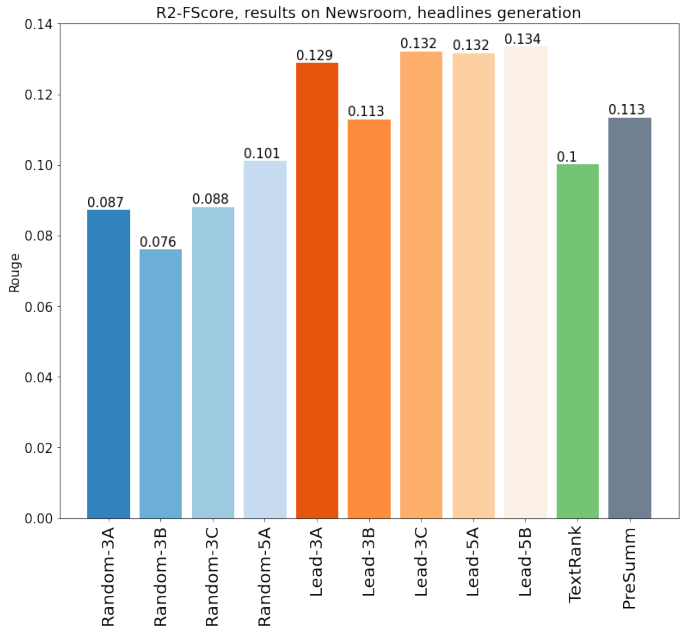

Figure 6.8: R2-Fscore for headline generation task performed on *Newsroom*, ROUGE Score

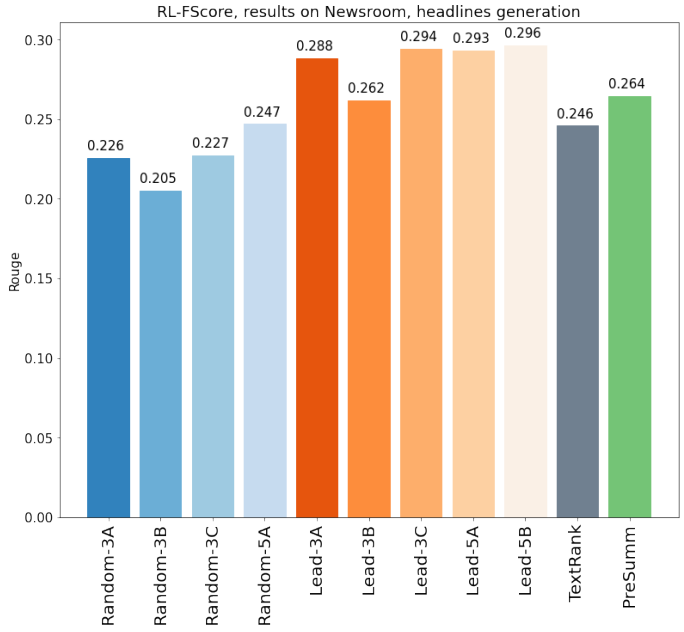

Figure 6.9: RL-Fscore for headline generation task performed on *Newsroom*, ROUGE Score

(5.4.2), the scores are mapped in the interval *[0,1]*. The reason why the precision scores computed for the generated summaries present negative values is that the implementation used to compute BERTScore, [30], applies a re-scaling on the calculations. From a practical perspective, negative values indicate that the pair generated_summary-reference_summary is worse than a random pair. This is not unexpected for the precision scores. In fact, precision is related to the cardinality of tokens in the generated summary that match with the token in the reference summary. Since the reference summary is much shorter than the generated one, it follows accordingly that the performance measured in terms of precision are very poor.

Summaries generation			
Configuration	Precision	Recall	F1
Random-3A	-0.08094	0.22046	0.06483
Random-3B	-0.08094	0.22046	0.06483
Random-3C	-0.08094	0.22046	0.06483
Random-5A	-0.08094	0.22046	0.06483
Lead-3A	0.00159	0.39901	0.19170
Lead-3B	0.00159	0.39901	0.19170
Lead-3C	**0.00159**	**0.39901**	**0.19170**
Lead-5A	-0.10348	0.40466	0.13727
Lead-5B	-0.10348	0.40466	0.13727
TextRank and Pegasus	-0.06021	0.25966	0.09426
PreSumm and Pegasus	-0.07186	0.29491	0.10419

Table 6.3: Results on Newsroom, summaries generation, BERTScore

A comparison can be done also through these charts: 6.10,6.11,6.12,6.13,6.14,6.15.

6.4.3 Best performing models

Lead-5B results to be the best performing configuration. Despite this, *Lead-3C* has been preferred because the average length of the reference summaries is much smaller than 5 sentences.

6.5 Experiments on Reuters

In this section, it will be deepened the approach adopted to conduct the experiments, in such a way to guarantee the reproducibility. Please note that the performances

Headlines generation			
Configuration	Precision	Recall	F1
Random-3A	0.14858	0.17244	0.16048
Random-3B	0.06458	0.20392	0.13326
Random-3C	0.14356	0.17285	0.15814
Random-5A	0.15480	0.18457	0.16960
Lead-3A	0.19849	0.22897	0.21357
Lead-3B	0.10833	0.27187	0.18862
Lead-3C	**0.20174**	**0.23645**	**0.21892**
Lead-5A	0.20171	0.23425	0.21780
Lead-5B	0.20446	0.23824	0.22116
TextRank and Pegasus	0.15317	0.18901	0.17099
PreSumm and Pegasus	0.17362	0.19751	0.18549

Table 6.4: Results on Newsroom, headlines generation, BERT Score

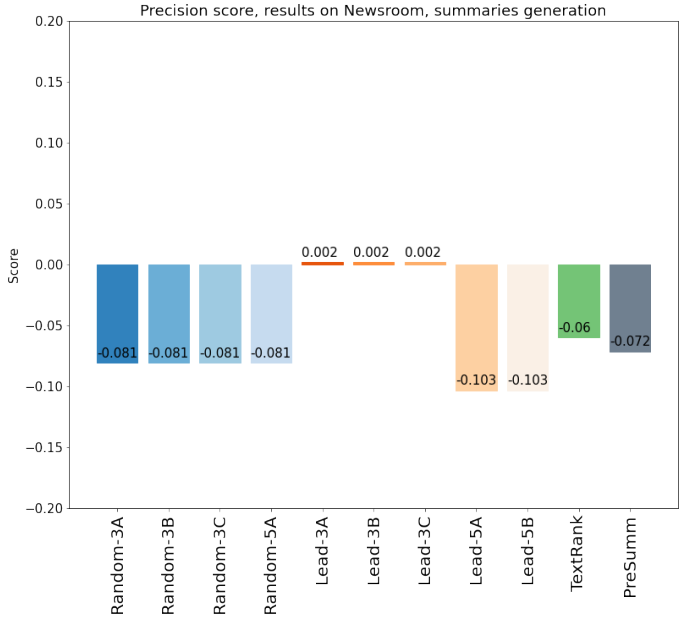

Figure 6.10: Precision score for summary generation task performed on *Newsroom*, BERT Score

could be measured only with regards to headlines generation tasks, since *Reuters* does not provide reference summaries, but only reference headlines.

Computational resources provided by *hpc@polito*, which is a project of Academic

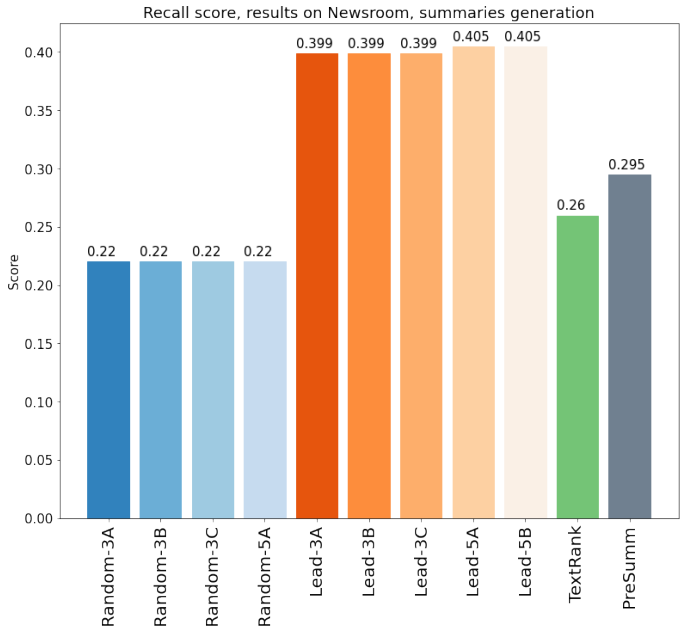

Figure 6.11: Recall score for summary generation task performed on *Newsroom*, BERT Score

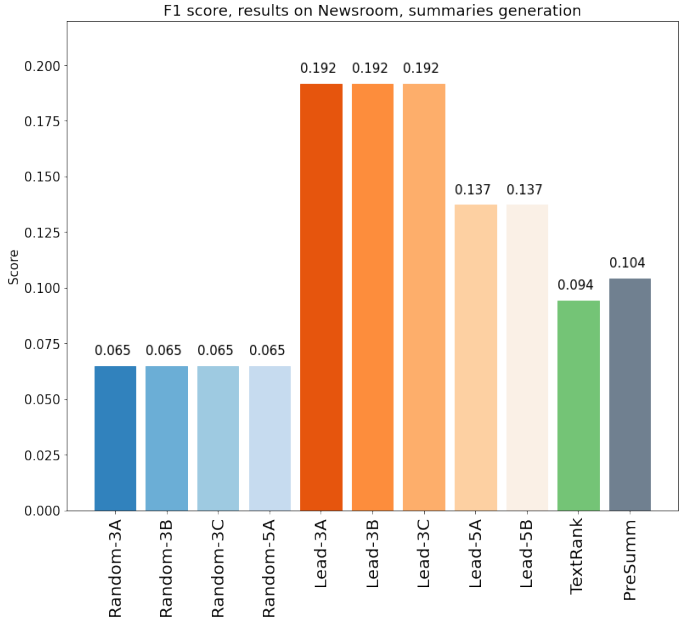

Figure 6.12: F1 score for summary generation task performed on *Newsroom*, BERT Score

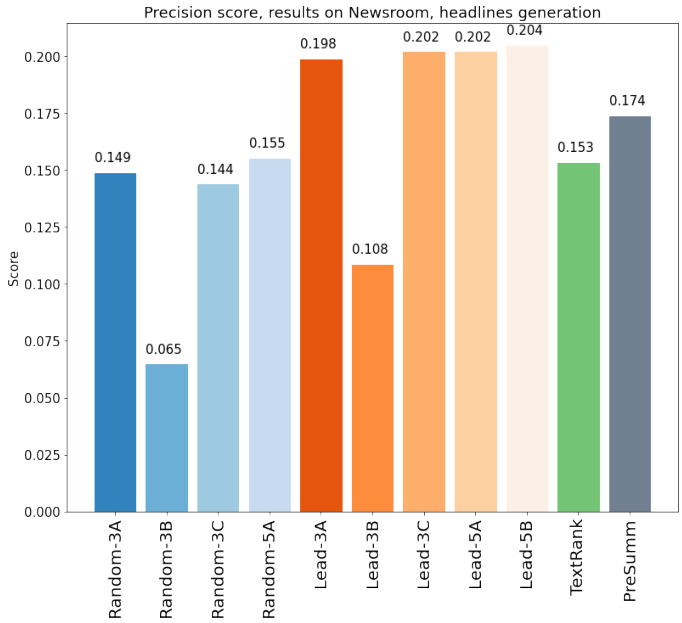

Figure 6.13: Precision score for headline generation task performed on *Newsroom*, BERT Score

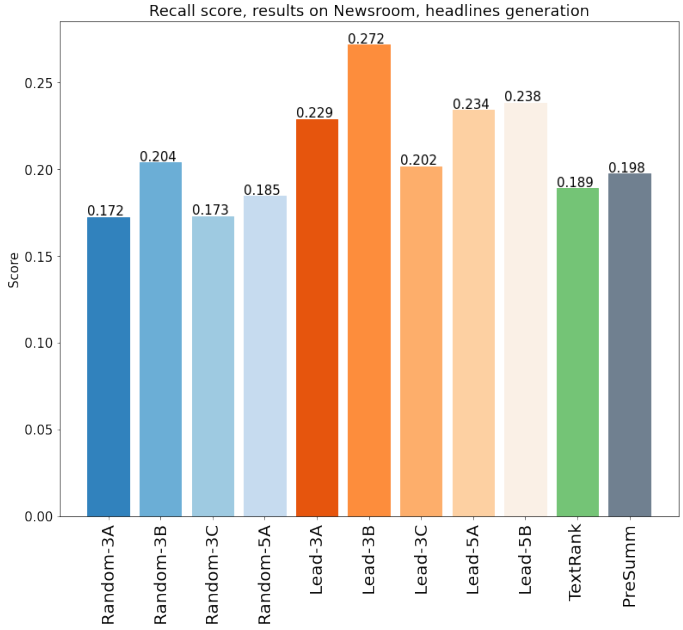

Figure 6.14: Recall score for headline generation task performed on *Newsroom*, BERT Score

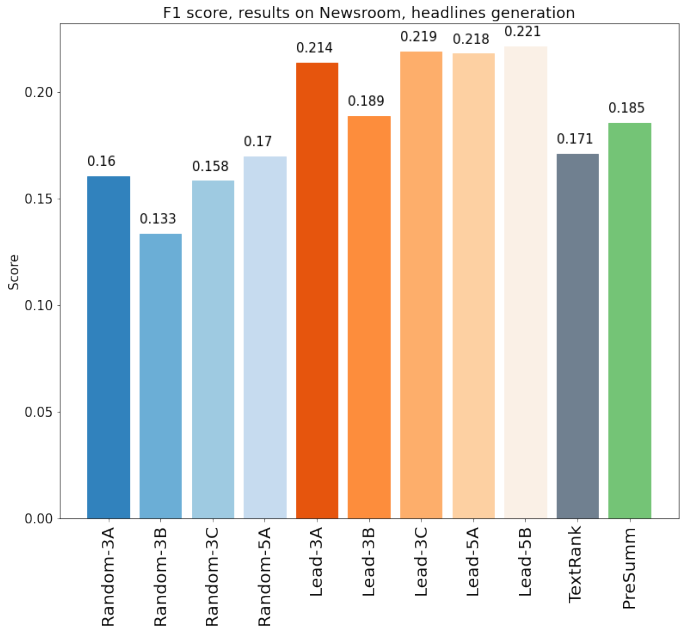

Figure 6.15: F1 score for headline generation task performed on *Newsroom*, BERT Score

Computing within the Department of Control and Computer Engineering at the Politecnico di Torino (`http://hpc.polito.it`)

6.5.1 Tested pipelines

Random *k* sentences and *Pegasus* The extractive summary is obtained by selecting k sentences randomly, the seed used to fix the randomness and guarantee the replicability is *42*. The abstractive headline is computed by *Pegasus*. Different configurations weere tested:

- *Random **2** sentences*, *Pegasus* parameters: $\boldsymbol{lr} = 2 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length** = 70*, ***max target length** = 15*, *average target length = 10*.

  Configuration name: **Random-2A**

- *Random **3** sentences*, *Pegasus* parameters: $\boldsymbol{lr} = 2 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length** = 70*, ***max target length** = 15*, *average target length = 10*.

  Configuration name: **Random-3A**

- *Random **5** sentences*, *Pegasus* parameters: ***lr*** $= 2 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*.

  Configuration name: **Random-5A**

Lead *k* sentences and *Pegasus* The extractive summary is obtained by selecting the first k sentences. The abstractive headline is computed by *Pegasus*. Different configurations were tested:

- *Lead **2** sentences*, *Pegasus* parameters: ***lr*** $= 2 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*

  Configuration name: **Lead-2A**

- *Lead **3** sentences*, *Pegasus* parameters: ***lr*** $= 2 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*

  Configuration name: **Lead-3A**

- *Lead **3** sentences*, *Pegasus* parameters: ***lr*** $= 5 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*

  Configuration name: **Lead-3B**

- *Lead **5** sentences*, *Pegasus* parameters: ***lr*** $= 2 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*

  Configuration name: **Lead-5A**

- *Lead **5** sentences*, *Pegasus* parameters: ***lr*** $= 5 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*

  Configuration name: **Lead-5B**

TextRank* and *Pegasus The extractive summary is obtained thanks to *TextRank* algorithm, which looks at selecting the *3* most salient sentences in the text. The abstractive headline is computed by *Pegasus*, trained with the following parameters: ***lr*** $= 2 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*

PreSumm* and *Pegasus The extractive summary is obtained thanks to *PreSumm* algorithm, which looks at selecting the *3* most salient sentences in the text. The abstractive headline is computed by *Pegasus*, trained with the following parameters: $\boldsymbol{lr} = 2 * 10^{-5}$, *weight decay = 0.01*, *epochs = 4*, ***max input length*** *= 70*, ***max target length*** *= 15*, *average target length = 10*. For the training of *PreSumm*, two approaches were tested:

- *PreSumm* trained on *NewsRoom*, since *Reuters* does not provide reference summaries Configuration name: **PreSumm-A**
- *PreSumm* trained on *Reuters*, the ground truth used in the training phase was artificially created: the first 3 sentences, for each article in the training set, were selected to be used as reference summaries to train the model Configuration name: **PreSumm-B**

6.5.2 Results

Regrettably, as previously mentioned, *REUTERS* does not allow an end-to-end model testing. In fact, this dataset does not provide the reference summaries, but only the reference headlines. For this reason, the main considerations can be made only on the final results of the pipeline: the headlines. Furthermore, in contrast to *NEWSROOM*, where the performance achieved by other algorithms (as *SHEG*), are public available, at the best of our knowledge, it is not the case for *REUTERS*. Indeed, although *ROUGE score* and *BERTScore* are computed, they won't be sufficient to understand the reliability of the implemented pipelines on *REUTERS* dataset, since no comparisons are possible. The major considerations pointed out are:

- *PreSumm* does not report good results. The primary cause could be attributed to the fact that *REUTERS* does not provide reference summaries and for this reason it is not possible to perform an adequate training. In fact, the configuration *PreSumm-B*, where *PreSumm* is trained considering as reference summaries the first 3 sentences in the article, performs very poorly, almost like a *Random k-sentences selection*. Better results are achieved by *PreSumm-A*, which exploits the *PreSumm* model trained on *NEWSROOM* dataset. So, it can be inferred that *PreSumm* trained exploiting ad hoc reference summaries may achieve interesting performance
- *Lead k-sentences selection* configurations, as in *NEWSROOM* dataset, report the best performance. The reason could be the same analyzed for the other dataset: selecting the first sentences works so well because we are dealing with news articles, where the most important information are generally contained in the first passages of the text. Furthermore, by adopting this approach, the

generated summary is not an aggregation of unrelated sentences taken from different parts of the document, but it is an extract where all the sentences are syntactically and contextually related; and this should simplify *PEGASUS* abstraction

To examine the specific performance metrics, please refer to the following sections where are reported the values calculated through *ROUGE Score* and *BERTScore*.

ROUGE Score

The performances measured with ROUGE score on the obtained results are summarized in the following table: 6.5

Headlines generation			
Configuration	R1-Fscore	R2-Fscore	RL-Fscore
Random-2A	0.34527	0.15023	0.33073
Random-3A	0.39158	0.18215	0.37484
Random-5A	0.44153	0.22167	0.42257
Lead-2A	0.50308	0.26758	0.48098
Lead-3A	0.50979	0.27409	0.48756
Lead-3B	0.52086	0.28882	0.49926
Lead-5A	0.51398	0.27936	0.49174
Lead-5B	**0.52641**	**0.29566**	**0.50478**
TextRank and Pegasus	0.45993	0.23877	0.44052
PreSumm-A	0.47778	0.24528	0.45651
PreSumm-B	0.41961	0.20685	0.40201

Table 6.5: Results on Reuters, headlines generation, ROUGE Score

A comparison can be done also through these charts: 6.16, 6.17, 6.18.

BERTScore

The performances measured with BERT score on the obtained results are summarized in the following table: 6.6.

A comparison can be done also through these charts: 6.19, 6.20, 6.21.

6.5.3 Best performing models

Lead-5B results to be the best performing configuration, and because of no comparison with reference summaries is possible, since they are not provided, this model

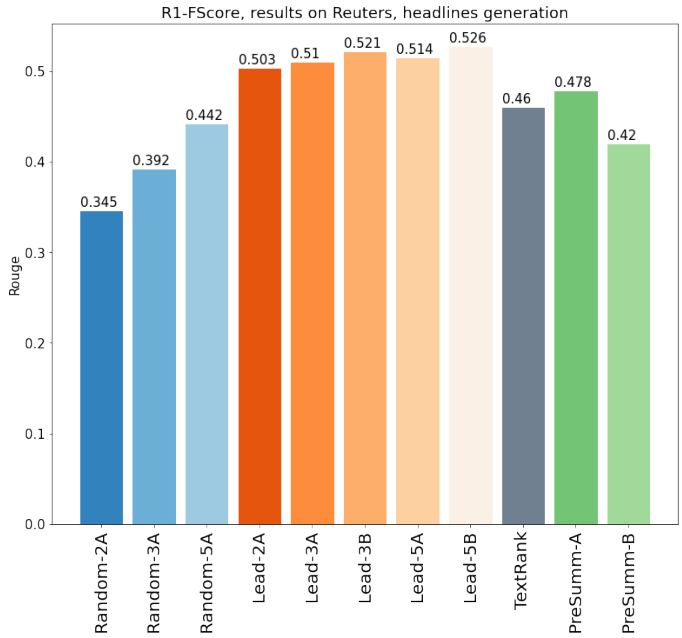

Figure 6.16: R1-Fscore for headline generation task performed on *Reuters*, ROUGE Score

Headlines generation			
Configuration	Precision	Recall	F1
Random-2A	0.20000	0.34670	0.27130
Random-3A	0.23619	0.39159	0.31161
Random-5A	0.27424	0.43918	0.35416
Lead-2A	0.31533	0.49497	0.40217
Lead-3A	0.31979	0.49863	0.40626
Lead-3B	0.32054	0.50810	0.41110
Lead-5A	0.32226	0.50217	0.40926
Lead-5B	**0.32481**	**0.51209**	**0.41526**
TextRank and Pegasus	0.28306	0.45456	0.36608
PreSumm-A	0.27005	0.43054	0.34788
PreSumm-B	0.22669	0.38207	0.30221

Table 6.6: Results on Reuters, headlines generation, BERT Score

will be used to implement the extraction step.

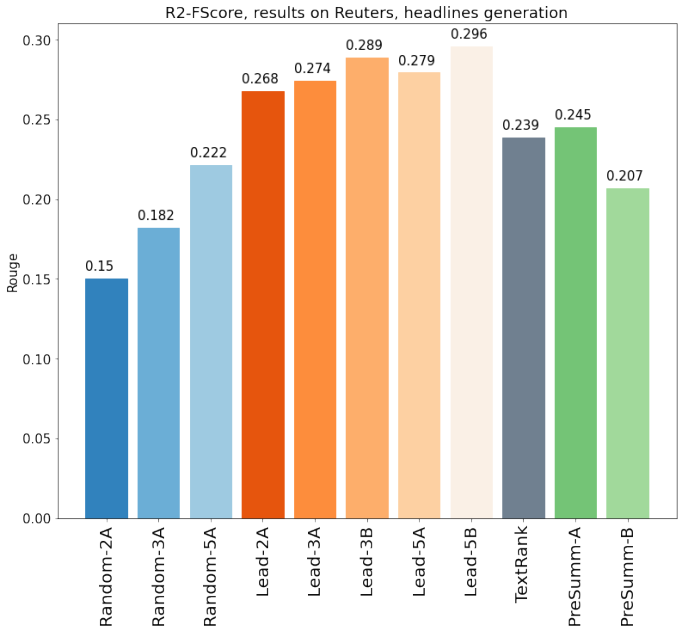

Figure 6.17: R2-Fscore for headline generation task performed on *Reuters*, ROUGE Score

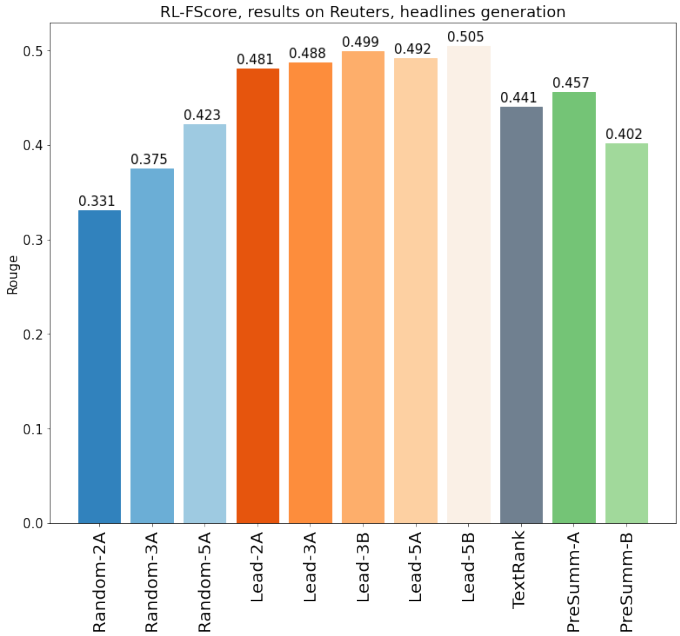

Figure 6.18: RL-Fscore for headline generation task performed on *Reuters*, ROUGE Score

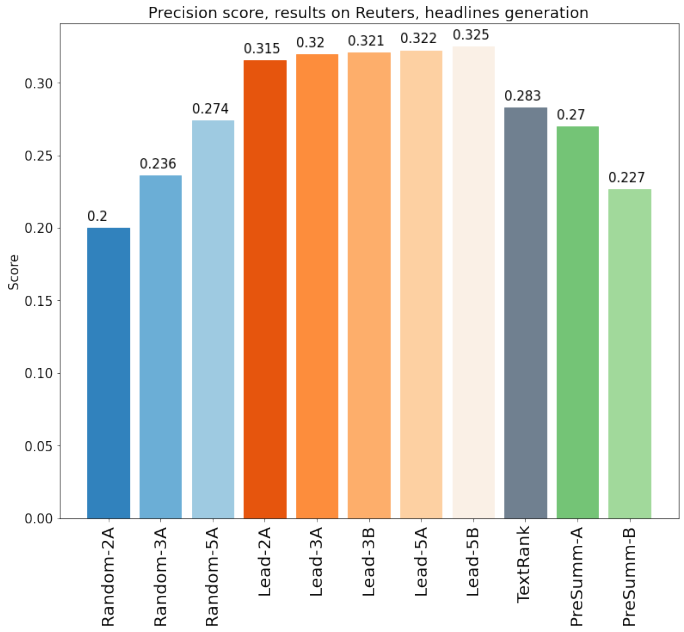

Figure 6.19: Precision score for headline generation task performed on *Reuters*, BERT Score

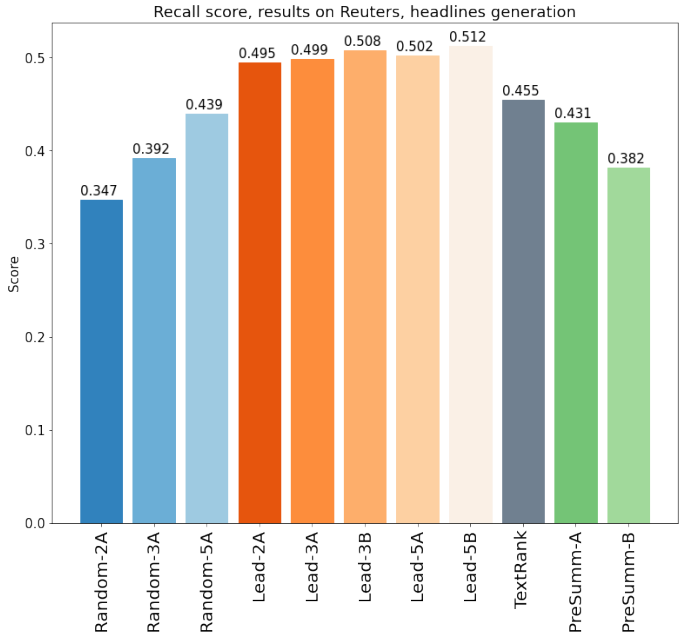

Figure 6.20: Recall score for headline generation task performed on *Reuters*, BERT Score

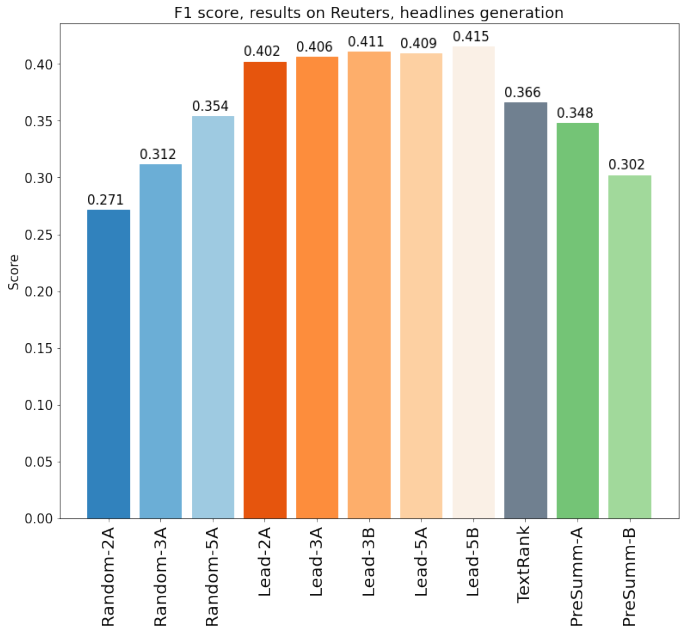

Figure 6.21: F1 score for headline generation task performed on *Reuters*, BERT Score

Chapter 7

Conclusions and final remarks

7.1 Conclusion

This thesis project allows me to deepen the NLP domain, especially the document summarization task. This is very popular today; in fact, now more than ever, companies are struggling to incorporate *Natural Language Processing* in their internal processes, in order to derive insights from textual data, which are not so easy to deal with.

Starting from the analysis of the available datasets and *state of the art* models, specific algorithms were chosen to be incorporated in the final pipeline.

The framework has been tested firstly on *Newsroom* and then on *Reuters*, attempting different configurations. Surprisingly, the best configuration turns out to be the extraction phase performed by selecting the first five sentences, and the abstraction step implemented by *PEGASUS* model. It is supposed that the reason why lead sentences selection works better than *PreSumm* is that we are dealing with news articles not very long, and the salient information is already contained in the first part of the text. Moreover, since *Reuters* does not provide reference summaries, it is not possible to properly perform the training required by *PreSumm*.

This limitation should be addressed by future works. A good strategy could be the one to employ the original pipeline (*PreSumm* and *PEGASUS*) on a more complete dataset, where documents are longer (to ensure that salient information will be in different part of the text, and not mainly in the first passages) and reference summaries are available to allow to finetune the model on the specific

dataset.

7.2 Future works

Future works should focus on creating or identifying ad-hoc financial documents datasets to allow an appropriate training and an end-to-end model testing. In fact, a limitation of *Reuters* dataset is the lack of reference summaries which avoids a complete measurement of the framework performances.

Although the consideration mentioned above, the main issue that future works should address is the establishment of a public financial documents dataset that can be used to benchmark different models. Indeed, without a common dataset, it is impossible to understand in depth the goodness of a proposed new model, since it can not be compared with other frameworks.

7.3 Final remarks

To summarize the analysis done at each step:

- *Scouting dataset phase*

  NEWSROOM and *REUTERS* were identified as the most suitable dataset to pursue the objective of the project. In fact, the former allows to test a model end-to-end, deepening the reliability both on summaries generation and headlines generation. The latter was employed because it is a financial documents dataset, and so it matches the final objective of this work

- *Scouting models phase*

  Different models were tested, and the most interesting ones were found to be *PreSumm* for extraction and *PEGASUS* for abstraction

- *Pipelines implementation*

  Many pipelines were analyzed and the better performance were achieved with *Lead k-sentences selection* combined with *PEGASUS*

- *Future works identification*

  Future works should focus on training and testing the original identified pipeline (*PreSumm* and *PEGASUS*) on a dataset characterized by:

  – longer documents
  – presence of reference summaries

Bibliography

[1] Mohammed Taher Pilehvar and Jose Camacho-Collados. *Embeddings in Natural Language Processing, Theory and Advances in Vector Representation of Meaning*. 2020 (cit. on pp. 3–10).

[2] Ramesh Nallapati, Bowen Zhou, Cicero Nogueira dos santos, Caglar Gulcehre, and Bing Xiang. *Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond*. 2016. URL: https://arxiv.org/abs/1602.06023 (cit. on p. 11).

[3] Max Grusky, Mor Naaman, and Yoav Artzi. *Newsroom: A Dataset of 1.3 Million Summaries with Diverse Extractive Strategies*. 2020. URL: https://arxiv.org/abs/1804.11283 (cit. on p. 11).

[4] URL: https://trec.nist.gov/data/reuters/reuters.html (cit. on p. 11).

[5] Wafaa S. El-Kassas, Cherif R. Salama, Ahmed A. Rafea, and Hoda K. Mohamed. *Automatic text summarization: A comprehensive survey*. 2020 (cit. on pp. 19, 20).

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2017. URL: https://arxiv.org/abs/1706.03762 (cit. on pp. 21–25).

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. URL: https://arxiv.org/abs/1810.04805 (cit. on p. 26).

[8] Ming Zhong, Pengfei Liu, Yiran Chen, Danqing Wang, Xipeng Qiu, and Xuanjing Huang. *Extractive Summarization as Text Matching*. 2020. URL: https://arxiv.org/abs/2004.08795 (cit. on p. 27).

[9] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. 2019. URL: https://arxiv.org/abs/2004.08795 (cit. on p. 30).

[10] Weizhen Qi, Yu Yan, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. *ProphetNet: Predicting Future N-gram for Sequence-to-Sequence Pre-training*. 2020. URL: https://arxiv.org/abs/2001.04063 (cit. on p. 31).

[11] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2020. URL: https://arxiv.org/abs/1910.10683 (cit. on p. 33).

[12] Abigail See, Peter J. Liu, and Christopher D. Manning. *Get To The Point: Summarization with Pointer-Generator Networks*. 2017. URL: https://arxiv.org/abs/1704.04368 (cit. on p. 34).

[13] Rajeev Kumar Singh, Sonia Khetarpaul, Rohan Gorantla, and Sai Giridhar Allada. *SHEG: summarization and headline generation of news articles using deep learning*. 2019 (cit. on pp. 36, 56).

[14] Sandeep Subramanian, Raymond Li, Jonathan Pilault, and Christopher Pal. *On Extractive and Abstractive Neural Document Summarization with Transformer Language Models*. 2020. URL: https://arxiv.org/abs/1909.03186 (cit. on p. 37).

[15] Vladislav Tretyak and Denis Stepanov. *Combination of abstractive and extractive approaches for summarization of long scientific texts*. 2020. URL: https://arxiv.org/abs/2006.05354 (cit. on pp. 38, 52).

[16] Yang Liu and Mirella Lapata. *Text Summarization with Pretrained Encoders*. 2019. URL: https://arxiv.org/abs/1908.08345 (cit. on pp. 39–41).

[17] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. *PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization*. 2020. URL: https://arxiv.org/abs/1912.08777 (cit. on pp. 39, 43).

[18] Yang Liu. *PreSumm*. URL: https://github.com/nlpyang/PreSumm (cit. on pp. 40, 49).

[19] Tagucci. *pythonrouge*. URL: https://github.com/tagucci/pythonrouge (cit. on p. 45).

[20] Chin-Yew Lin. «ROUGE: A Package for Automatic Evaluation of Summaries». In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013 (cit. on p. 45).

[21] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. *BERTScore: Evaluating Text Generation with BERT*. 2020. URL: https://arxiv.org/abs/1904.09675 (cit. on p. 47).

[22] maszhongming. *MatchSum*. URL: https://github.com/maszhongming/MatchSum (cit. on p. 49).

[23] huggingface. *Bart*. URL: https://huggingface.co/transformers/model_doc/bart.html (cit. on p. 50).

[24] huggingface. *Pegasus*. URL: https://huggingface.co/transformers/model_doc/pegasus.html (cit. on p. 50).

[25] huggingface. *ProphetNet*. URL: https://huggingface.co/transformers/model_doc/prophetnet.html (cit. on p. 50).

[26] huggingface. *T5*. URL: https://huggingface.co/transformers/model_doc/t5.html (cit. on p. 50).

[27] abisee. *PGN*. URL: https://github.com/abisee/pointer-generator (cit. on p. 50).

[28] Paco Nathan. *PyTextRank, a Python implementation of TextRank for phrase extraction and summarization of text documents*. 2016. DOI: 10.5281/zenodo.4637885. URL: https://github.com/DerwenAI/pytextrank (cit. on p. 52).

[29] Rada Mihalcea and Paul Tarau. *TextRank: Bringing Order into Texts*. 2004. URL: https://web.eecs.umich.edu/~mihalcea/papers/mihalcea.emnlp04.pdf (cit. on p. 52).

[30] Tianyi Zhang*, Varsha Kishore*, Felix Wu*, Kilian Q. Weinberger, and Yoav Artzi. «BERTScore: Evaluating Text Generation with BERT». In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=SkeHuCVFDr (cit. on p. 61).