People's Democratic Republic of Algeria Ministry of Higher Education and Scientific Research

Saad Dahleb Blida University – Blida 1

Computer Science department

Generative models for automatic multi-document summarization

In order to obtain the Master's degree

Domain: Mathematics and computer science

Branch: Software Engineering

Realized by:

Bensidiaissa Walid

Bouchetara Rym

Supervised by:

Mr. Abdallah Hicham Kamech

Date : 26/10/2020

In front of the jury :

Ouahrani Leila

Zahra Fatima Zahra

Year: 2019 - 2020

Abstract

In recent years, there has been an explosion in the amount of text data from a variety of sources. This data needs to be effectively summarized to be useful.
Text summarization in natural language processing has widely been approached with extractive methods that stick to selecting parts of the original document to capture the main topic ideas. What has been less attempted is abstractive summarization.
In our work, we focus on the latter type of automatic summarization. We performed a series of experiments to judge the effectiveness of abstractive summarization systems, whether or not they are applicable in a real context. Our choice went towards the use of the machine learning approach with models inspired by the architecture of transformers. At first, we focused on the extractive multi-document summarization, then we finetuned DistilBart, a recent model proposed by the Huggingface team, for abstractive summarization on different datasets and compared each of the obtained models with the basic model, and then between them.
We also created an algorithm to be used during preprocessing. The objective of this algorithm is to replace similar sentences that are grouped in clusters by a single sentence belonging to that cluster. This algorithm also uses a model based on transformers. Evaluation is done automatically using the ROUGE scores. Our method, as simple as it is, has shown promising results since the scores were higher when using that preprocessing.

Keywords: Automatic Summary, Abstract, Multi-Document, Deep Learning, Semantic similarity, Fine-tuning, Transformers, BERT, GPT-2, BART.

ملخص

في السنوات الأخيرة، شهدنا انفجارًا في كمية البيانات النصية الواردة من مصادر مختلفة. تم التعامل مع ملخص النص في معالجة اللغة الطبيعية على نطاق واسع باستخدام طرق الاستخراج التي تلتزم باختيار أجزاء من المستند الأصلي لالتقاط الأفكار الرئيسية للموضوع. لكن ما تم التعامل به نادرا هو الملّخص النصي المبني على إعادة الصياغة.

قمنا بالتركيز في عملنا على النوع المبني على إعادة الصياغة من الملخص التلقائي. فقد أجرينا سلسلة من التجارِب للحكم على فعالية أنظمة الملخصات المجردة، سواء كانت قابلة للتطبيق في سياق حقيقي أم لا. كان اختيارنا موجهًا نحو استخدام نهج التعلم الآلي مع نماذج مستوحاة من هندسة Transformers. ركزنا أولاً على الملخص الاستخراجي متعدد المستندات، ثم قمنا بتحسين DistilBart، وهو نموذج حديث اقترحه فريق Huggingface، من أجل الملخص المجرد لمجموعات البيانات المختلفة وقارنا النماذج التي تم الحصول عليها مع النموذج الأساسي، ثم بينهم.

وقد أنشأنا أيضًا خوارزمية تُستخدم خلال المعالجة المسبقة لملخص المستندات المتعددة. الهدف من هذه الخوارزمية هو استبدال الجمل المماثلة التي تم تجميعها في مجموعات بواسطة جملة واحدة تنتمي إلى الأخيرة. تستخدم هذه الخوارزمية أيضًا نموذجًا يعتمد على Transformers. يتم التقييم تلقائيًا باستخدام ROUGE. طريقتنا، بسيطة بقدر ما هي، أعطت نتائج واعدة لأنها أعلى عند استخدام هذه المعالجة المسبقة.

الكلمات المفتاحية : الملخص التلقائي، الملخص، المستندات المتعددة، التعلم العميق، التشابه الدلالي BART ، Fine-tuning Transformers، BERT، GPT-2،.

Résumé

Nous avons connu ces dernières années une explosion de la quantité de données textuelles provenant de diverses sources. Le résumé de texte dans le traitement du langage naturel a été largement abordé avec des méthodes d'extraction qui s'en tiennent à la sélection de parties du document original pour capturer les idées principales du sujet. Ce qui a été moins tenté cependant, c'est le résumé abstractif.
Dans notre travail, nous nous concentrons sur ce dernier type de résumé automatique. Nous avons réalisé une série d'expériences pour juger de l'efficacité des systèmes de résumé abstractif,s'ils sont applicables ou non dans un contexte réel. Notre choix s'est orienté vers l'utilisation de l'approche de l'apprentissage automatique avec des modèles inspirés par l'architecture des Transformers. Nous nous sommes d'abord concentrés sur le résumé extractif multi-documents, puis nous avons affiné DistilBart, un modèle récent proposé par l'équipe Huggingface, pour le résumé abstractif sur différents corpus de données et nous avons comparé chacun des modèles obtenus avec le modèle de base, puis entre eux.
Nous avons également créé un algorithme qu'on utilise lors des prétraitements pour le résumé multi-documents. L'objectif de cet algorithme est de remplacer les phrases similaires qui sont regroupées dans des clusters par une seule phrase appartenant à ce dernier. Cet algorithme utilise également un modèle basé sur les Transformers. L'évaluation est faite automatiquement en utilisant les scores ROUGE. Notre méthode, aussi simple soit-elle, a donné des résultats prometteurs puisque les scores étaient plus élevés en utilisant ce prétraitement.

Mots-clés: Résumé automatique, Abstractif, Multi-documents, Apprentissage profond, Similarité sémantique, Raffinement, Transformers, BERT, GPT-2, BART.

Acknowledgements

Above all, it is thanks to Allah that we were able to get to where we are. It is above all thanks to Him that our path has been lit to finish our modest work.
We would also like to thank all those who helped us from near and far to finish this work. Our promoter, who allowed us to work on this subject. Misters Sam Shleifer and Suraj Patir for their patience, their availability, and especially their advice and answers to our questions. Our parents who have been there for us all along, you have sacrificed everything for your children sparing neither health nor efforts. You have given us a wonderful model of perseverance. We are forever in your debt.
We would like to express our gratitude to our brothers and sisters, family, and friends for their encouragement, especially to Messabhia Mohamed Amine who generously provided us with his own Google Cloud Platform storage that we needed. We also extend our sincere thanks to the teachers who made these years more beautiful, especially Dr. Ouziri, we have only memories to cherish by her side. Thank you Dr. Ouziri for being who you are, passionate about your work, thank you for offering your time to help us even outside of your working hours. We will never forget that your sessions were both productive and fun. You are our example. Thank you also to Dr. Habani and Dr. Chikhi for inspiring us to give our best. Lastly, we thank professor Benblidia, and Dr. Guesmia for their big help and support.
To all of them, we extend our thanks, respect, and gratitude.

Contents

List of Figures

List of Tables

General Introduction

Context

There are many people whose daily work consists of largely if not exclusively in the production of summaries: judges who sum up the evidence presented in court, the company secretary who records the minutes of a board meeting, the scientist who reviews recent publications in his research field, and so we could go on. However, we cannot possibly create summaries in every situation manually.
We live in an age of near-instant gratification where people are constantly searching for the next hit, and they wouldn't hesitate to switch to another product or service if a company does not respond to their needs. Hence, companies have to think quickly; and with the speed at which the world moves, keeping up is a constant task that never gets easier.
Fortunately, massive online information is available. Information can come from any source: media, blogs, books, journals and articles, web pages, etc. Most available data however come as texts, and it is unthinkable to read every document a search engine returns to find what we are searching for. Because generally, to get trusted information we need to read from multiple trusted sources, and by doing so we waste a lot of time. This volume of text is indeed an invaluable source of information and knowledge, but it needs to be effectively summarized to be useful.
Those needs led to exhaustive research in the natural language processing area for automatic text summarization.

Problematic

Automatic text summarization is the task of producing a concise and fluent summary without any human help while preserving the meaning of the original text document [1]. Producing a summary can be a difficult task for humans and it is even more challenging for machines. We usually read the integrity of the text to develop our understanding before writing a summary highlighting its important ideas. Since computers lack human knowledge and language understanding, it makes automatic text summarization a very complex task. Most of the work in the field of automatic summarization is based on the extractive approach which consists of extracting parts of the text, generally sentences, and concatenating them to produce a summary. The second approach, abstractive summarization, consists of producing the summary by interpreting the text and generating a new shorter text, parts of which may not appear as part of the original document and it is not enough to produce words and phrases that capture the gist of the source, as the summary should be accurate and understandable. This is a more challenging approach but it is the approach ultimately used by humans.

Objectives

In our project, we conduct a series of experiments on both extractive and abstractive summarization while focusing on abstractive summaries of multiple documents. All models we perform our experiments on are based on a very recent architecture called Transformers, our objectives are:

- Finetune an existing model for abstractive multi-document summarization with different datasets.
- Try to improve the quality of our generated summaries with a method independent from the model used.
- Finally, evaluate the performance of these models, whether or not they are usable in a real context.

Organization of the memoir

In order to facilitate the reading of this memoir, we will briefly present the chapters that make it up. Our memoir is organized as follows:

- **Chapter 1 - Concepts and definitions of automatic text summarization:** In this chapter, we will start by defining what automatic summarization is and present the types to which an automatic summarization system can belong. Then we will briefly present the general steps that systems follow to generate a summary. Lastly we will mention some evaluation methods.
- **Chapter 2 - Deep neural networks:** This chapter covers the necessary prerequisites for our work. Our choice is oriented towards a model that uses the machine learning approach, so we will define this approach, the types of machine learning, deep neural networks, and the most commonly used architectures.
- **Chapter 3 - Methods of automatic text summarization:** Over the past half-century, the problem has been addressed from different perspectives, hence we will present the different works that have been proposed and discuss our choices.
- **Chapter 4 - Design of our system Quiksum:** In this chapter, we will define our problem, present the systems we have used, and describe the steps of summary generation for each of these systems. We will also detail the features of our application and close with the method we proposed to improve the summaries.

- **Chapter 5 - Realization of Quiksum:** This chapter is dedicated to the implementation and evaluation of our systems as well as the implementation of our application. First, we will describe the specifications of the environment we have been working on, then we will present our choice of programming languages, the libraries used, and the training and evaluation corpora. Then we will present our application, the metrics used, and finally present and discuss the results obtained for our experiments.

Chapter One

Concepts and definitions of automatic text summarization

1.1 Introduction

Data explosion or the century of big data is what defines this era the most; what we're witnessing is only natural given the number of users and the variety of their concerns. As a consequence of this extraordinary growth, people are getting overwhelmed, first because of the expanding availability of information and secondly because of the little amount of time available to process it and make the best out of it; that's where the summaries come. Summaries are important when it comes to processing tremendous amounts of information. Among the different types of data, we are particularly interested in text data, in this case, a summary is defined as a text that is created from one or more writings, that conveys important information within the unique text(s), and that is no longer than half of the original text(s), and usually, significantly less than that [2].

This chapter covers the general concepts of automatic text summarization. We will first give a general definition of automatic text summarization. Next, we will present the different types of automatic summaries, we will also explain the steps for generating a summary. Besides different approaches exist for this problem, hence we will describe each one of them. Finally, we will conclude with the evaluation criteria of automatic summaries.

1.2 Definition

Chettri and Chakraborty [3] define summarization as a way of abstracting pertinent data from one or more sources. It increases the probability of finding the focuses of writings, so the user will spend less time reading the entire documents. According to Moiyadi et al. [4], automatic text summarization is the creation of a shortened version of a text by a computer program. The product of this procedure still contains the

most important points of the original text and is generally referred to as an abstract or a summary. From this definition, we can notice three significant aspects concerning automatic summaries characteristics: first, the summary has to contain fewer words otherwise there is no point in it. Second, it has to preserve the key information of the source. Lastly, it is not created manually, rather by a computer.

1.3 Types of automatic text summarization

Although all summarizers have the same objective in common - producing a summary, each summarizer focuses on a particular aspect. A summarization system can be classified using many aspects [5]. We can classify the summarization according to the input, the output, and the audience.

1.3.1 According to the input

This classification is based on the characteristics of the input document. Does the text change, or is it constant? Does the summarizer take one document or multiple documents at a time? The most common types of summarization when focusing on the input aspect are static, dynamic, single-document, and multi-document summarization. This section explains each one of them.

Static summarization

This type of summarization provides the same summary when summarizing the same text meaning that the input is fed without performing any changes on it [6].

Dynamic summarization

This summarization does not have a fixed input which will lead to different summaries each time [6]. We can give as an example the summarization of the content of a blog where the summarizer dynamically generates the summary according to the current content of the blog, hence each time the content changes, the summary changes too.

Single-document summarization

In single document summarization, the system takes one document at a time and produces a summary for each input it takes [6]. A single document can be composed of some subdocuments with multiple paragraphs. The described content of each of these subdocuments emphasis different aspects all surrounding the same topic.

Multi-document summarization

In multi-document summarization, we summarize multiple documents that have the same topic [6]. For example, summarizing documents talking about a crash accident, or coronavirus recent discoveries, etc. This means that different articles can have different sources, which is one of the advantages of this type of summarizers. Using multi-document summarization on documents obtained after a google search is another common use case. However, it is a big challenge to avoid redundancy since all of them are more likely to include a certain degree of similar information.

1.3.2 According to the output

Based on the output, there are two commonly used types, which are extractive summarization, and abstractive summarization.

Extractive summarization

Extractive summarization consists of pulling out the most relevant parts of the documents, generally, sentences, that yields the idea of the subject to form the summary [6]. Extractive summarization is easier to perform because it just selects a subset of the input text based on its relevance. This ensures that the summary is readable and grammatically correct. However, consistency is not guaranteed. For example, if the summary system selects sentences containing references (acronyms, personal pronouns, etc.) and does not select sentences containing their antecedents, it's very possible that the summary produced will be unclear. In addition, the extractive methods may produce summaries containing a lot of redundancy.

Abstractive summarization

It is inspired by the method used by humans. As we generally proceed by understanding the documents then creating a summary using our own words [6], abstractive summarization consists of creating new sentences without necessarily using the same words to represent the global most pertinent content of the original documents; which improves the focus of a summary, reduce its redundancy and keeps a good compression rate [7]. Abstractive summarizers require a more profound analysis of the content, one major issue is that they are difficult to design due to their heavy dependence on linguistic techniques [8].

1.3.3 According to the audience

Usually, selecting information depends not only on the document(s) to be summarized but also on contextual factors such as the audience [9]. For what category of readers

are we summarizing the document, and what are the needs of those readers ? In this section we will present some of the types that are based on the audience.

Generic summarization

A generic summary provides an overall global idea of the original input by including all the relevant information without focusing on a particular subject in the input [10].

Query-oriented summarization

A summarization system that considers the user's preferences is called query-oriented. A query-oriented summary is a summary that includes the parts of the original document that are closely related to the query [10]. For example, if a user wants a summary focusing on events of a particular date, it must contain events that turn around this date without losing the main interest of the document.

Informative summarization

A summary can allow the person concerned with it to have a general idea of the document as a substitute for it [11]. This way, he or she will not have to read the whole document to have access to the relevant information. After reading an informative summary, it's easy to tell what are the main ideas presented. It includes informative content and generally contains these three components: the purpose, the methodology, results, and conclusion. This type is usually used in journals, thesis, research articles, surveys, experiments, etc. where it's a must to present all the essential information.

Indicative summarization

Unlike informative summaries, indicative summaries are summaries that do not include details (informative content) [11]. They inform about the scope and give a global description of the document to help one to decide whether he/she would be interested in the document. The abstracts found in books and reports are an example of indicative summaries.

Background summarization

A system may assume that the reader does not have extensive prior knowledge of the subject, so it includes explanatory material [12]. Summaries of historical texts, for example, pay attention to certain details such as actors, place, and time. This type of summarization is called background summarization.

Just-the-news summarization

A system may assume that the reader has sufficient background to interpret it in the context, providing just novel or principal themes. Such summaries are extremely useful in tracking news stories, tracing new product reviews, etc. This type of summarization is called just-the-news summarization [13].

Neutral summarization

When the input material contains opinion or pre-judgment content, we can decide whether we want our summary to include those opinions or not. A neutral system provides a summary of the relevant content of the input document(s) without including criticisms or evaluation [5].

Evaluative summarization

Summarizing opinions from customer reviews or comments on social networks is very useful, especially for decision making [14]. An evaluative summary is a summary that includes some opinions either explicitly by including statements of judgment or implicitly by including some materials and omitting others [5]. In the next few years, it's expected that the evaluative summarization will deal with the domain specifics and also with user satisfaction [15].

1.4 Automatic summarization process

All summarizers generally follow these steps for generating a summary [6]:

- Pre-processing.
- Summary generation.
- Post-processing.

In this section, we will describe each one of these steps.

1.4.1 Pre-processing

Pre-processing of a text is a set of steps to make a document in a format that is predictable and analyzable. In any input text, some words and symbols have no significant meaning related to the topic discussed and generally used to link words with each other. The repeated occurrence of these words can mess with the score of the important words. To solve this problem several methods have been used like Tokenization, Normalization and others.

Normalization

Normalization is a set of steps to render the text in a standard form [16]. We often find in many types of text like blog comments, chat conversations and customer reviews in online shop abbreviations, misspelling words for example "wym" as "what do you mean", "cy@" as "see you", even there are symbols to describe a certain meaning "fresh food > canned food" means "fresh food is better than canned food". therefore the use of Normalization is highly recommended to have a standardized meaning of the texts. According to [17], normalization improved the accuracy by 4% in their work.

We can achieve the standard form of a text by applying certain operations such as:

- The removal of special characters and tags like HTML tags and other symbols.
- The Replacement of abbreviations and symbolized meanings with the correct words.
- Correction of the common misspelling mistakes.

Table 1.1 shows some examples of normalization of words generally founds in social media.

Raw	Normalized version
FYI, B4	For your information, before.
Tomz, 2morrow, tmrw	Tomorrow
:'(	Sad face

Table 1.1: Examples of normalization

Tokenization

Tokenization is the process of breaking a stream of text into words, phrases, symbols, or other meaningful elements called tokens [18]. The goal of Tokenization is to split a large text into a set of sentences and then split those sentences to explore the words in it, therefore the list of tokens becomes an input for many statistical approach algorithms in extractive summarization that will be mentionned in chapter two.

- **Segmentation into sentences**
  Generally, punctuation is key when it comes to separate sentences in a large text. The period (.) is used to indicate the end of declarative sentences, the exclamation point (!) for exclamatory sentences, the interrogation point (?) for interrogative sentences. According to [19], a sentence starts with an uppercase letter and ends with one of the mentioned punctuations. However, this definition cannot be applied in some cases like:

- A sentence can have more sub-sentences splitted by punctuations like the colon (:), the semicolon (;) and the comma (,) in order to describe a saying or sequence of actions.
- Some languages don't even have punctuation, like Thai and original Chinese.
- We can find the period (.) used in other cases like mentioning a website "www.name.com", in numbers like "85.97", etc.

It is clear that we cannot just rely on the uppercase letters and the period to determine sentence boundaries. In the following example we have meaningful sentences if we split on the double points ":" , and the comma: "The moon does not have a light of its own: it reflects the sun light, just like the moon humans also does not shine on their own as they are the reflection of the people they are surrounded by."
David D.Palmer [19] grouped many factors that can assist in sentence segmentation:

- **Part of speech**: Palmer and Hearst [20] declared that the use of parts of speech can assist in sentence segmenting.
- **Word length**: Riley [21] used the length of the words before and after a period as one contextual feature.
- **Prefixes and suffixes**: Reynar and Ratnaparkhi [22] used both prefixes and suffixes of the words surrounding the punctuation mark as one contextual feature.

1.4.2 Summary generation

The process of generating summaries relies on many factors such as the format of the input, the purpose of the summarization, the type of summary, etc. Hence there is no fixed method for generating a summary. Multiple solutions exist and follow a certain approach. The most popular approaches that exist are: statistical approach, graph-based approach, linguistic approach, sentence compression approach, and machine learning approach [23], [24]. In this section we will briefly explain each one of them.

Statistical approach

Statistical approaches depend on statistical features to extract relevant units from the input document(s) [25]. To do that, it uses one or several techniques that will be combined to assign a score to the text units (words, sentences, or other). Eventually, the highest score refers usually to the highest importance. A statistical approach does

not require many language-dependent tools. It is easy to implement and does not need a lot of processing power. However, for the feature combination, it is critical to know how to fuse them, and how much weight each one should have. The features weights are generally set manually, by using optimization or using machine learning techniques.

Graph-based approach

These methods create graph structures to represent the documents. Vertices illustrate sentences and edges in between indicate how similar two sentences. Documents can also represent vertices and the edges represent the similarity between the documents. The graph-based approach depends on transforming document or documents units into a graph using a similarity measure [2]. Figure 1.1 illustrates a set of documents represented as a graph.

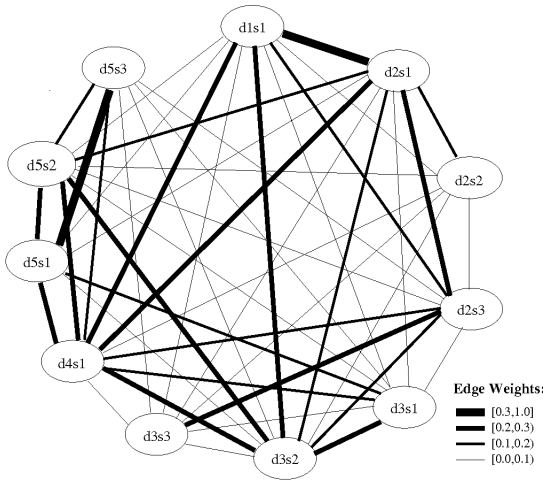

Figure 1.1: Weighted cosine similarity graph for a cluster of documents [26].

We can use a graph-based approach for single as well as multi-document summarization. It can be language-independent when using only lexical similarities such as cosine similarity and statistical features [13]. One of its major problems is when we use it to process a great volume of text. The greater the size the more it needs huge processing power.

Linguistic approach

The linguistic approach uses sophisticated natural language processing (NLP) techniques to generate summaries [24]. Some of these techniques are part-of-speech, rhetoric relations, semantic, etc. It is more powerful than the statistic approach since it integrates more elaborate processing of the input text. Nenkova and McKeown [27]

suggested using it as a post-processing task to improve the linguistic quality of the generated summary rather than a processing one.

Sentence compression

Sentence compression aims to retain the most salient information of a sentence, and delete the least critical ones, rewritten in a short form [23]. In general, since sentence compression is language dependant, it can't be used alone to have a summary but in conjunction with other approaches. Also, it can be used as a post-processing task after generating a summary, which will eliminate more redundancy and allows more space for other sentences to be included [13].

Machine learning approach

Machine learning is to make a machine perform the tasks that humans can do in a much better and faster way. Similarly to humans, this is achieved by making the machine learn using data. It can be applied to a wide range of fields such as image recognition, speech recognition, automatic text summarization, etc. The process of using data refers to training, while task execution refers to the prediction or inference. Machine learning techniques use complex algorithms that improve by themselves based on observing the data they are trained on, then they are used to predict based on unseen data [28]. In the case of automatic text summarization, we can feed the machine with pairs of documents and their summary, then it will generate summaries after it has sufficiently trained.

1.4.3 Post-processing

Post-processing is a step that is generally applied to improve the quality and readability of the generated summary. It can remove the redundancy, resolve any inconsistencies concerning the pronouns, etc. Anaphora resolution is a method used to replace the pronouns in a sentence by the subject they refer to in the previous sentence, it is considered as a pre-processing as well as a post-processing technique [29]. Redundancy elimination concerns either the removal of repetitive sentences in terms of meaning or the removal of consecutive repetitive words in the same sentence, resulting in a more compressed and clear summary. Post-processing techniques are not always used compared to the prepossessing ones which are usually crucial for this task.

1.5 Automatic text summarization evaluation

Evaluating a summary is a difficult task because there is no ideal summary for a given document or set of documents. The goal of any summarization system is to optimize topic coverage and readability. Topic coverage refers to the ability of the summary to incorporate the main topics from the document, and readability means that the sentences flow logically, hence it's human-understandable [30]. For this, there are many evaluation criteria [30]:

- Salience: Are we capturing the most valuable information of the document?
- Length: Is the summary of a proper length?
- Structure and coherence: Summaries must be well structured and organized. A summary should not be just a block of information but should be assembled to form a coherent body of information. If a person or entity is mentioned, the relationship to the rest of the summary should be clear. Also, it should be simple to identify to whom or to what pronouns and other non-self-referential elements refer.
- Balance: Is it covering all the topics from the initial document?
- Grammar: Is the text grammatically correct?
- Non-redundancy: A redundant summary is a summary which, if we exclude a particular sentence or a set of sentences, we can still get the overall pertinent information.

There are two methods to evaluate these models: intrinsic techniques and extrinsic ones [12].

1.5.1 Intrinsic evaluation

Intrinsic evaluation is meant to evaluate the system in of itself [30]. This one determines the summary quality based on a comparison between the automatically generated summary and the original document, or to a human-made manual summary (reference summary).

Cosine similarity

Similarity measures are a set of functions that can be used to compare how similar are two entities such as words, sentences, documents, etc. Cosine similarity [31] is one of these measures, it can also be used to rank documents based on a query. The value

represents the cosine of the angle between two vectors and determines whether those two are pointing in roughly the same direction.

Let X and Y be the representations of a system summary and its reference document based on the vector space model. Using the cosine measure as a similarity function, we have the equation 1.1:

$$Cos(X,Y) = \frac{\sum_i x_i * y_i}{||x|| * ||y||} \tag{1.1}$$

Unit overlap

Another similarity measure is Unit Overlap [32]. Equation 1.2 describes how to calculate it.

$$overlap(X,Y) = \frac{||X \bigcap Y||}{||X|| + ||Y|| - ||X \bigcap Y||} \tag{1.2}$$

Where X and Y are representations based on sets of words or lemmas. $||X||$ is the size of set $X$and $||Y||$ is the size of the set Y.

Longest common subsequence

We calculate the longest common subsequence [33] as shown in equation 1.3.

$$lcs(X,Y) = \frac{length(X) + length(Y) - edit_{di}(X,Y)}{2} \tag{1.3}$$

where X and Y are representations based on sequences of words or lemmas, $lcs(X,Y)$ is the length of the longest common subsequence between X and Y, $length(X)$ is the length of the string X, and $edit_{di}(X,Y)$ is the edit distance of X and Y.

PYRAMID

The Pyramids method [34] is a semi-automatic evaluation method which its basic idea is to identify summarization content units (SCUs) used for comparison of information in summaries. A pyramid is a representation of the reference summary. It also represents the opinions of several writers, each of which has written a model abstract. The key characteristic of a pyramid is that it quantifies the agreement between the human abstracts. Since we use it to evaluate the content of the abstract, the units of comparison in a pyramid correspond to the units of meaning. Hence, an SCU is a set of textual units of reference abstracts expressing the same information. It has a weight equal to the number of reference summaries that instantiate it. These SCUs are organized in a pyramid where each tier group together SCUs of the same weight. The number of annotated model summaries represents the maximum number of tiers in a pyramid. Based on that, we can compute the informativeness of a new summary as the

ratio of the sum of the weights of its SCUs to the weight of an optimal summary with the same number of SCUs. Pyramid has a set of formulas that we can use to calculate the score. Figure 1.2 represents a pyramid with two SCUs at the top and four in the next tier, it shows two optimal summaries that this pyramid provides.

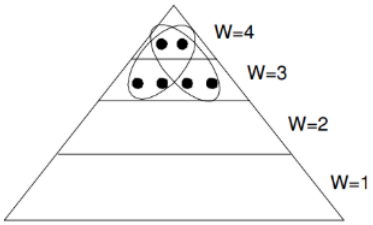

Figure 1.2: Two of six optimal summaries with 4 SCUs [34].

ROUGE

ROUGE [35] stands for recall oriented understudy for gisting evaluation. It is an intrinsic metric for evaluating summaries and is based on BLUE [36] defined for machine translation. Although ROUGE is not as good as human evaluation, it gives acceptable results and is more convenient. There are multiple variations of ROUGE, such as ROUGE-N, ROUGE-L, ROUGE-S, etc. Further details will be provided in chapter five, but essentially:

- ROUGE-N measures unigram, bigram, trigram, and higher-order n-gram overlap.
- ROUGE-L measures the longest matching sequence of words using the longest common subsequences (LCS). An advantage of using LCS is that it does not require consecutive matches but in-sequence matches that reflect sentence-level word order.
- ROUGE-S is any pair of words in a sentence in order, allowing for arbitrary gaps.

1.5.2 Extrinsic evaluation

Extrinsic evaluation is a method that determines the impact of summarization on other tasks [30]. If the task can be completed using the summary instead of the original document, this means that the summary includes all the relevant information in the original document. Some of these tasks are document categorization, informational retrieval, and question answering.

Document categorization

This evaluation task seeks to determine whether the generic summary is effective in capturing essential information to correctly categorize the document. The classifica-

tion of summaries and comparing them to a full document categorization or random sentence extracts allows determining if the generic summary permits an analyst to categorize a document as quickly as possible. For instance, in TIPSTER SUMMAC [37] evaluation, they proposed ad-hoc tasks and categorization tasks. In the ad-hoc task, the objective is to test the pertinence of indicative summaries on a topic. First, they proceeded with the evaluation as an ad-hoc task, then the human subject had to choose after reading the document one category out of five, or "none of the above"; then, they calculated the scores using recall, precision, and F1-score.

Information retrieval

Relevance correlation is an information retrieval based measure for assessing the relative decrease in retrieval performance when moving from full documents to summaries [33]. If we perform information retrieval on a summary, it should present as good results as a full document. For example, given a query Q and a corpus of documents D, a search engine ranks all documents in the corpus according to their relevance to query. If rather than the corpus D, we substitute the full document with the corresponding summaries and rank the resulting corpus of summaries S by the same retrieval engine for pertinence to the query; we expect that the ranking will be very similar.

Question answering

Question answering seeks to evaluate the summaries in terms of their informativeness. Such evaluation was carried out in [38], authors picked four Graduate Management Admission Test reading comprehension exercises. The exercises were multiple-choice, with a single correct answer. They measured how many of the questions the subjects answered correctly under different conditions. Then, they compared the results of answering in these different conditions. Also, in TIPSTER SUMMAC [37], they compared the generated automatic summaries manually to some answer keys for each input document; to decide if the answer is correct, partially correct or incorrect. Then, they defined ARS (Answer Recall Strict), and ARL (Answer Recall Lenient) metrics to measure accuracy (see Equation 1.4).

$$ARS = \frac{n1}{n3}, ARL = \frac{n1 + (0.5 * n2)}{n3} \tag{1.4}$$

Where $n1$ and $n2$ are the numbers of correct and partially correct answers in the summary, and $n3$ is the number of questions answered in the key.

1.6 Conclusion

Automatic text summarization is the task of generating a shorter version of one or multiple documents. In this chapter, we went through its general concepts by presenting the different types of automatic text summarization, the general steps that summarizers follow for generating summaries, the approaches that exist, as well as the methods for evaluating a summary. While summarization has been there for decades, it gained more popularity in the past few years. In the following chapter, we will present some of the works that have been proposed for this task.

Chapter Two

Deep neural networks

2.1 Introduction

Machine learning is a subfield of artificial intelligence that aims to give computers the ability to learn and improve its performance from experience (training) without being explicitly programmed [28]. It teaches what comes naturally to humans and is useful when we have complex tasks or problems involving a large amount of data. Machine learning covers multiple applications, such as image recognition, speech recognition, natural language processing, etc.

In this chapter, we will briefly present machine learning, its concepts, and its subsets. Besides, this chapter explains the most important points that lead to our proposed solution.

2.2 Machine learning

Machine learning is a field of study that gives computers the capability to learn without being explicitly programmed [39]. At a high level, machine learning is the process of teaching a computer system on how to make accurate predictions when fed data without human intervention or assistance. It might look similar to writing regular code, we write an algorithm, the machine executes the algorithm on given data, then it can do the same task with new data that it has never seen before. But with machine learning, rather than manually writing code with a specific set of instructions, the machine is trained using large amounts of data and learns to perform a task without being explicitly told how to do so. The computer needs to train to achieve its goal, and during training, it attempts to create a logic and improve it by having access to more data, doing that process a certain amount of time.

There are many ways in which a machine learns: it can be either supervised learning, unsupervised learning, semi-supervised learning, or reinforcement learning [40].

2.2.1 Supervised learning

In supervised learning, the data have labels to guide the machine towards the exact patterns it should look for, for example, images of handwritten figures marked to indicate which number they correspond to; a supervised-learning system would learn the characteristics of each number and eventually recognize handwritten numbers and make a distinction between them. We feed the system with example inputs and their desired outputs, and the goal is to learn a general rule that maps inputs to outputs. The training process continues until the model reaches the desired level of correctness on the training data [40]. Besides, the learning algorithm can compare its output with the correct one, find the errors, and modify itself accordingly. We can represent the data in the following form: $d_i = (x_i, y_i)$ with x the input, y the associated target, i the index of the observation, and (x_i, y_i) represents one training example. That is, what the algorithm does, it outputs a function called hypothesis function, and this function takes the input x, tries to output the estimated value of y.

Depending on the nature of the output, there are two types of supervised learning: regression and classification. When the output value is discrete, it is called a classification task. The hypothesis function predicts the class or category for a given observation. Generally, classification models predict a continuous value as the probability of a given input belonging to each output class. Inversely, when the output variable is real, it is called a regression task. The objective of the hypothesis function is to map input variables to real value output variables. That is, the input can be real-valued or discrete.

2.2.2 Unsupervised learning

Sometimes, the targets are not available, and we only have $d_i = x_i$. Besides, it is easier to get unlabeled data than labeled data and more practical to avoid manual intervention; that's where unsupervised learning comes. Unsupervised learning is to train a machine using data that is unlabeled, so the machine has to learn using that data without any guidance [40]. Unsupervised learning attempts to explore the data and find some structure within, it can be a goal in itself to discover unknown hidden patterns.

Clustering is a form of unsupervised learning. It groups data that have similar patterns or structure into groups called clusters. In a clustering problem, we attempt to find an algorithm to group unlabeled data into coherent clusters. The K-means [41] algorithm is by far the most popular and most widely used clustering algorithm. Suppose we would like to have k clusters:

The first step is to initialize randomly k points called the cluster's centroid. K-means is an iterative algorithm that repeats two steps: first is cluster assignment, and

then a move centroid step. The cluster assignment step loops through each example and calculates the distance between the centroids and the dataset then assigns each dataset to the closest centroid. The move centroid step takes the centroids and moves them to the average (mean) of the dataset. These two steps are repeated until the clusters stop changing. The K-means algorithm expects two inputs: the K value, and the dataset. A good clustering result mainly depends on the K value.
Association is another important concept. Association allows discovering pertinent relationships between elements by finding rules that describe large portions of data. For example, people that tend to buy new houses are more likely to buy furniture.

We can also use unsupervised learning to help us visualize high dimension data or use it to generate more data.

2.2.3 Semi-supervised learning

Semi-supervised learning is intermediate between supervised and unsupervised learning [40]. When there's not enough labeled data to produce an accurate model, and there are no resources to get more data, we can use this technique to increase the size of the training data. In case we want to train a model to classify data, but at the same time, to give our algorithm a hint about how to construct the categories, we can use both labeled and unlabeled data. Generally, this combination comprises a small amount of labeled data and a large amount of unlabeled data. For example, one can perform unsupervised learning on all data to extract features then perform supervised learning on the new representation, or one can first perform supervised learning and then predict targets on unlabelled data, and finally, incorporate this new data to supervised training [42].

2.2.4 Reinforcement learning

Reinforcement learning is about taking proper action to maximize a reward in a specific situation. The agent learns to perform an action in an uncertain environment. The model is provided with feedbacks as rewards or punishments as it operates its problem space [40].

There are two types of reinforcements: positive reinforcement is when the agent is rewarded to encourage it to follow a particular behavior, and negative reinforcement is when it is trained to avoid a negative behavior to increase the likelihood of the positive one.

The input is the initial state from which the model starts and the output depends on the state of the current input, at each time step, the next input uses the output of the previous step. The best solution is the one that provides the maximum reward.

2.3 Deep neural networks

Deep learning is a sub-field of machine learning inspired by the functioning of the human brain. When the features are not set manually and there are millions of parameters, with enough data for training, deep learning models are easier than classic machine learning solutions. Neural networks are two types: classic networks, and deep networks. Deep neural networks are neural network with multiple layers and thousands of nodes that are used to extract high level features. At every layer, the network learns a new, more abstract representation of the input which massively improves the ability of the network to do its task. Besides, when it comes to complex problems such as image classification, natural language processing, and speech recognition they perform very well compared to classic networks.

2.3.1 Artificial neural networks

Inspired by the biological neural network, artificial neural networks neural networks (ANNs) also called feed-forward neural networks (FFNs) mimic how the brain works. They are essentially multiple layers containing a set of neurons also called perceptrons connected by multiple values called weights. Each neuron calculates a specific value based on the inputs that precede it. They are also called feed-forward neural network because they has no cycle between the output and the input. Figure 2.1 represents the architecture of an artificial neural network with multiple hidden layers.

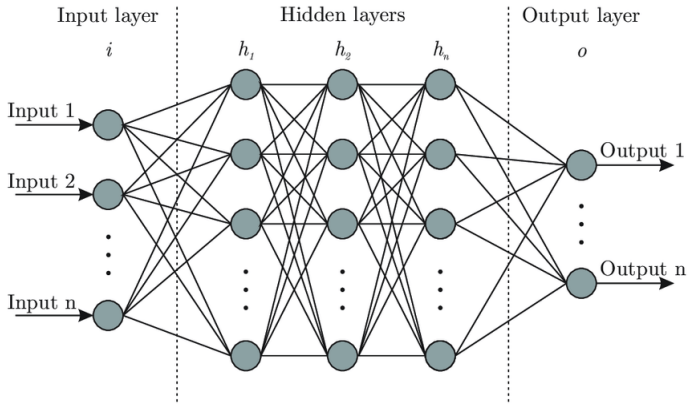

Figure 2.1: Artificial neural network architecture [43].

As we can see, each layer has perceptrons or nodes in it; there are no connections between nodes in the same layer; however, each node from a given layer is connected into all the nodes of the next layer.

- **Layer i:** The first layer (i) is the input layer; the input layer consists of a vector representing the data in its vectorized form.

- **Layers h:** Each node in the hidden layer is a result of the application of a function on the activations obtained by multiplying the inputs by the weights. Common choices for the activation function include the sigmoid and the tanh function. Another activation function that has become well used in deep learning research is the rectified linear unit (ReLU). The hidden layers are intermediate layers between the input and output layer.
- **Layer o:** It produces the neural network's results based on given inputs.

The neuron

A neuron is a single computational unit in a neural network, it has multiple inputs $(x_1, x_2, ..., x_n)$ associated with their weights $(w_1, w_2, ..., w_n)$, in addition to a bias input x_0 and its weight w_0, and only one output y is calculated via application of an activation function, for example, sigmoid (σ) on z which is the sum of multiplication of the inputs with their weights. The figure 2.2 illustrates one neuron with n inputs and one output.

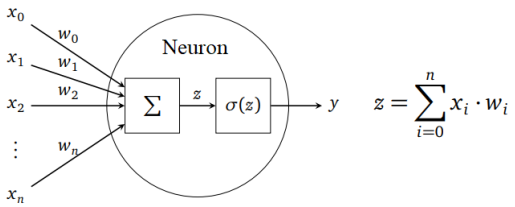

Figure 2.2: Neuron illustrated [44].

2.3.2 Forward propagation

Artificial neural networks consist of having multiple layers, the input layer represents the input data that will flow into the hidden layer; the way it does this is by having weights that connect each neuron in one layer with neurons of the next layer, these weights are initialized randomly at first. We take each input and multiply it by each associated weight of it, this value is fed to an activation function that performs other transformations, we repeat the process between the hidden layer and the output layer the resulting value is the network's prediction and the process is called Forward Propagation; however, if we want to compare these values with expected values the result won't be necessarily correct, thus we want the error value to be minimized and we can do that with backpropagation (see figure 2.3).

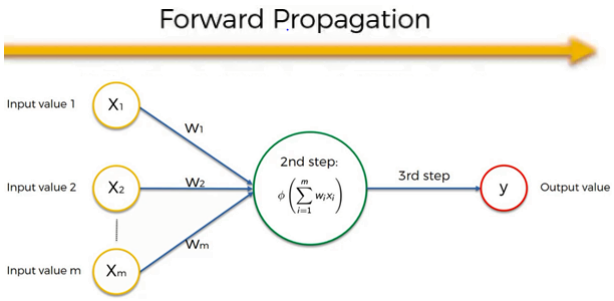

Figure 2.3: Forward propagation illustrated. [45].

Activation functions

Activation functions can mean the entire difference between a neural network that works, and a neural network that does not. Activation functions are mathematical equations that transform their input to a useful output for the neural network. They are attached to each neuron and decide whether each neuron's input is relevant to the model's prediction; they can be linear or nonlinear.

A linear function takes the form of $y = ax + b$ and can be effective only on one layer deep neural network regardless of how complex the architecture is. Real-world problems are nonlinear, and the derivative of a linear function is always equal to a which has no relation with the input. It is a constant gradient, so the gradient descent value will be constant. If there is an error in the prediction, the changes made by backpropagation will not depend on the change of the input. Therefore, nonlinear functions are used in almost all neural networks.
Nonlinear functions have to be continuous and differentiable. A neural network can take any input from $+\infty$ to $-\infty$ and should be able to map it to an output range between $[0; 1]$ or $[-1; 1]$ in some cases. Among nonlinear functions, there is ReLU (Rectified Linear Unit), LeakyReLU (Leaky Rectified Linear Unit), Tanh, GELU, etc.

- ReLU is computationally efficient as it allows the network to converge very quickly but when the input are smaller or equal to 0, the gradient of the function becomes null, hence the model stops learning as the backpropagation does not adjust the weights anymore. Equation 2.1 describes its formula and figure 2.4 represents the graph of the ReLU function.

$$\text{ReLU}(x) = \max(0,\text{x}) \tag{2.1}$$

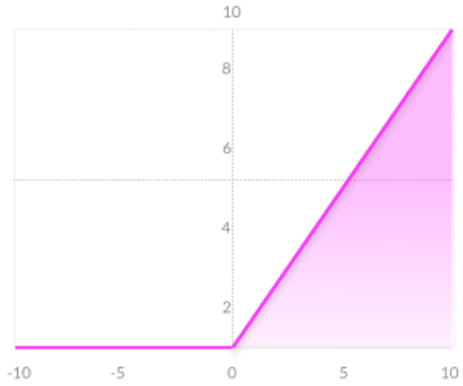

Figure 2.4: ReLU function graph [46].

- Leaky ReLU is a variation of ReLU and has a small slope in the negative area to enable backpropagation even for negative value, though those predictions can be inconsistent. Equation 2.2 describes its formula and figure 2.5 represents the graph of the ReLU function.

$$LeakyReLU(x) = \begin{cases} 0.01x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \tag{2.2}$$

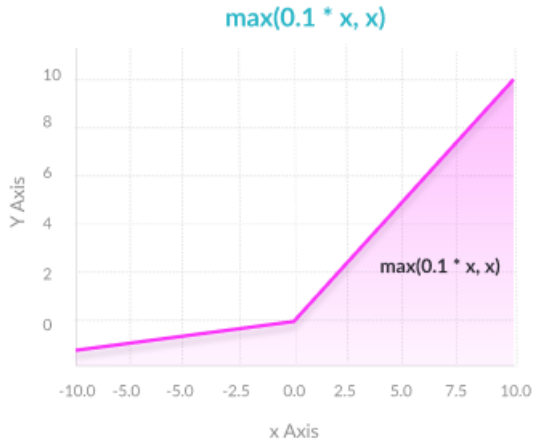

Figure 2.5: Leaky ReLU function graph [46].

- Tanh makes it easy to model inputs that have strongly negative, neutral, or strongly positive values, but it also suffers from the vanishing gradient problem. The formula to calculate it is in equation 2.3 and its graph in figure 2.6.

$$\tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \tag{2.3}$$

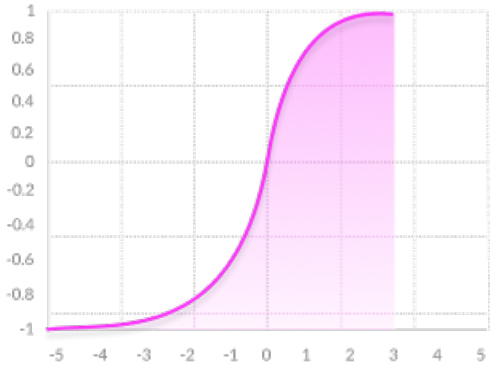

Figure 2.6: Tanh function graph [46].

- GELU function solves most of the previous issues and more importantly avoids the vanishing gradients problem. It provides a well-defined gradient in the negative area and prevents neurons from dying; besides it performs best in transformers models. The formula to calculate it is described in equation 2.3 and its graph in figure 2.6.

$$\text{GELU}(x) = 0.5x\left(1 + \tanh\left(\sqrt{2/\pi}(x + 0.044715x^3)\right)\right) \tag{2.4}$$

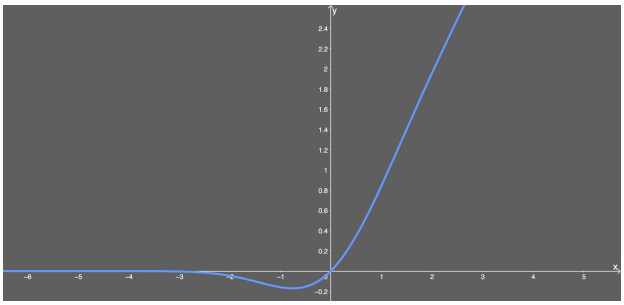

Figure 2.7: GELU function graph [47].

As seen in the equation 2.4, it is just a combination of some functions and approximated numbers.

2.3.3 Backpropagation

Backpropagation is a widely used learning algorithm for feedforward neural networks rediscovered in 1986 by David Rumelhart, Geoffrey Hinton, and Ronald Williams. The goal of backpropagation is computing the gradient (the derivative) of the loss function with respect to each weight in one layer at a time iterating backward from the output layer.

The cost function

Neural networks learn by updating the weights that connect the neurons and they do that by applying the backpropagation algorithm based on the error function. A cost function or the error function is how the global network did, a single value generally measured via the average difference between the output and the expected output of a training sample. We can conclude that cost function depends on the network weights, the biases of the network, and one training sample with the expected value of that training sample. Examples of commonly used cost functions:

- **Mean squared error** (MSE) Also known as the Quadratic cost function or maximum likelihood,it is the default choice for regression problems. Equation 2.5 explains how to calculate it.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2 \tag{2.5}$$

  Where N is the total number of inputs in the training sample, f_i is the value generated by the model for input i, and y_i is the expected value for the input i.

- **Cross-entropy cost function** (CE) When a model generates the output token by token for example, at each time step, the network produces a probability distribution over the possible next token. This distribution is penalized from being different from the true distribution. The formula to compute cross-entropy is described in the equation 2.6

$$CE = -\sum_{i}^{C} t_i \log(f(s_i)) \tag{2.6}$$

  Where t_i and s_i are the groundtruth and the output probabilities for each class i in C.

- **Binary Cross-entropy cost function** (BCE) also known as binary classification generally used for prediction problems where the expected value for each input is one of two values. Equation 2.7 explains how to calculate it.

$$BCE = -\sum_{i=1}^{C'=2} t_i \log(s_1) - (1 - t_1)\log(1 - s_1) \tag{2.7}$$

- **Multi class cost functions** It is generally used for prediction problems where the expected value for each input is one of more than two values or classes. For example there is multi class cross entropy, sparse multi class cross-entropy, and kullback leibler divergence.

Gradient decent and Adam optimization algorithm

Gradient decent is an iterative optimization algorithm for finding the minimum of a function; in backpropagation it is used on the error function.

Adam optimization algorithm is an adaptive learning rate optimization algorithm designed for training deep neural networks. The choice of an optimization algorithm can mean the difference between good results in minutes, hours, or days. Stochastic gradient descent is another algorithm that maintains a single learning rate for all weight updates, and it does not change during training. Instead, Adam is an adaptive learning rate method meaning it computes individual learning rates for different parameters. However, in some areas, Adam does not converge to an optimal solution. A solution to this problem is to use weight decay with this algorithm.

Neural network learning with backpropagation

Backpropagation algorithm looks for the minimal error value and the way it does that is explained in the following algorithm:

- (a) Initializing weights randomly.
- (b) For each training sample:
  - (1) Compute the prediction value of an input in the training sample with forward propagation.
  - (2) Calculate the error by using a cost function on the prediction value and the expected value.
  - (3) Compute new weights using the derived formula on the cost function with respect to each weight with backpropagation.
  - (4) Update network weights.
- (c) Repeat step (b) until the error is at its minimum.

Fine-tuning a neural network

Fine-tuning is to take the weights of a pre-trained neural network and use them as the initial weights for the training on another dataset of the same domain and nature. The weights are adjusted during this training such that they perform well to a certain degree on the new dataset. This practice is useful in cases when the data available is limited, and when the domains and tasks are similar, hence there is no need to train a model from scratch.

2.3.4 Convolutional neural networks

A convolutional neural network (CNN) is an artificial neural network that is most popularly used for analyzing images. Although the most common application of CNNs has been image analysis, it can also be used for other data analysis or classification problems. More generally, we can think of CNN as an artificial neural network that has some type of specialization for being able to detect patterns and make sense of them. CNN has hidden layers called convolutional layers, and these layers are precisely what makes it a CNN.

The convolutional layer

Convolutional layers are the layers where a set of filters are applied to the original input or other feature maps in a deep CNN, mainly to extract high/low-level features from the input like edges and colors for images based on how deep the CNN is. Filters or kernels are a group of weights initialized randomly and learned and adjusted later alongside the size of kernels and the number of kernels.

The pooling layer

Pooling layers are similar to convolutional layers. Pooling involves selecting a pooling operation, much like a filter to be applied to feature maps which are the result of an applied convolutional layer on the input or other feature map. The size of the pooling operation or filter is smaller than the size of the feature map; specifically, these are typically used to reduce the dimensionality of the network.

The fully connected and classification layer

The fully Connected layer is simply a feedforward neural network. The input to the fully connected layer is the output from the final pooling or convolutional layer, which is flattened (turned from a dimensional matrix into a vector) and then fed into the fully connected layer. This layer is placed before the classification layer of the CNN that outputs the final result.

2.3.5 Recurrent neural networks

The problem with FFNS is that the inputs are independent of each other: in each iteration, the network restarts fresh, it does not remember what it saw in the previous iteration when processing the current set of data. It is generally used in classification problems, such as telling whether or not the seller is good, filtering emails as spams or not spams. Besides, the majority of current data have a sequential form. For example,

if we want to predict the next word in a sentence it requires the knowledge of all previous or most pertinent words before the current one.

Recurrent Neural networks (RNNs) are FFNs that can take a sequential input and produce a model that changes over time in ways that yield accurate results dependent on the context. Human memory is context-aware, some of those memories are forged based on recent information, others are older, similarly, an RNN will remember information in a form of hidden state. The first one introduced had a very simple architecture and they computed in one way, either from left to right or from right to left.

Recurrent neural network's architecture

Since data are sequential, this means that each input will have a time step. Figure 2.8 represents the base architecture of the RNN.

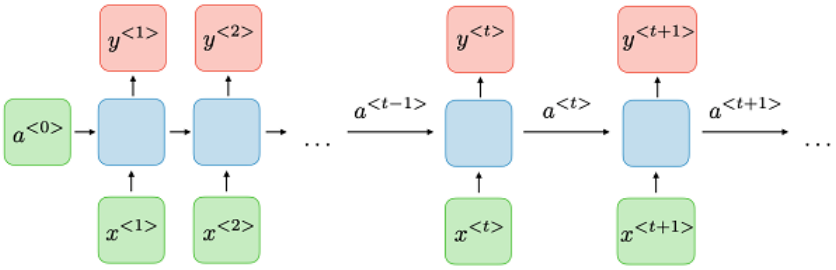

Figure 2.8: Base architecture of RNNs [48].

Where:

- $x^{<t>}$ is the input at the time step t, it is passed on always as a vector.
- $a^{<t>}$ represents the hidden state at time step t, it is the memory of the network.
- $y^{<t>}$ is the output at the time step t.

Recurrent Neural Networks suffer from short-term memory. If a sequence is too long, they won't be able to carry all the important information from past steps. When processing a text to do predictions, RNNs may leave out important information from the beginning.

Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) were created as the solution to short-term memory. They have internal mechanisms called gates that can adjust the flow of information. These gates distinguish the relevant data in a sequence which should be kept from the irrelevant that should be thrown away. By doing that, it can pass the important information down the long chain of sequences to make predictions.

2.3.6 GRU and LSTM

GRU uses gates called update gate and resets gate. The update gate decides how much past information to push forward; while the reset gate sorts the irrelevant data and tells the model to forget this data and move forward without it.

Meanwhile, LSTMs are more sophisticated but at the same time more complex as well. They have four components: the forget gate, the input gate, the cell state, and the output gate. The forget gate sorts out the relevant and irrelevant information and pushes forward only the relevant information towards the cell state. The input gate is necessary for deciding which values will be updated. The cell state transfers relative information down the sequence chain and acts as the memory of the network. Lastly, the output gate decides what the next hidden state should be. The new cell state and the new hidden are used for the next time step and so on.

Although there is no simple way to decide which one to use for a particular use case, GRUs train faster and perform better than LSTMs on less training data. Generally, the decision is made after doing a trial and error to test their performance.

Limits of recurrent neural network

First, the sequential aspect of RNNs; every hidden state depends on the previous hidden state when processing a sequence using RNNs. This makes RNNs very inefficient and slow on GPUs. The second limit is the difficulty of learning long-range dependencies in the network. Although LSTMs and GRU can have long-term memory, remembering information for long periods is still challenging, and RNNs can still have short-term memory problems. Finally, some words have multiple meanings and are context dependent which is hard to capture.

2.4 Generative models

2.4.1 Variational Autoencoders

Autoencoders are a specific type of unsupervised feed-forward neural network consisting of an encoder and a decoder.An autoencoder takes an input, converts it into a smaller representation. This representation is called context vector. The decoder then takes the context vector and reconstruct the original input using it. So at the encoder, we are taking it from high dimensional input to lower-dimensional features, then it goes the other way from low dimensional features back to a high dimensional output of the same size as the input. This limitation size in the middle enables the discovery of interesting information about the data.

The encoder and decoder are trained together as one big network. The loss func-

tion penalizes the network for creating outputs different from the input; it is usually either the mean-squared error or cross-entropy between the output and the input. Finally, autoencoders are useful to construct a compressed version of data and to learn features to initialize a supervised model by throwing the decoder part and using only the encoder. However, they are not able to generate new data. The reason is that the encoded vectors are not continuous, sampling a variation from there, the decoder will generate an unrealistic output because the decoder has no idea how to deal with that region of the latent space. During training, it never encountered encoded vectors coming from that region of latent space.

With variational autoencoders (VAEs), the output of the encoder is not a value. Instead of one value for each attribute, VAEs represent each latent attribute as a range of possible values. By having an encoder model that output a range of possible values from which it is possible to randomly sample and feed it into the decoder model, VAEs force a continuous, smooth latent space representation. The decoder can create data from any sample of of the latent distribution. Thus, nearby vectors in latent space have very similar reconstructions. For that, VAEs combine autoencoders with variational inference. Figure 2.9 describes this architecture.

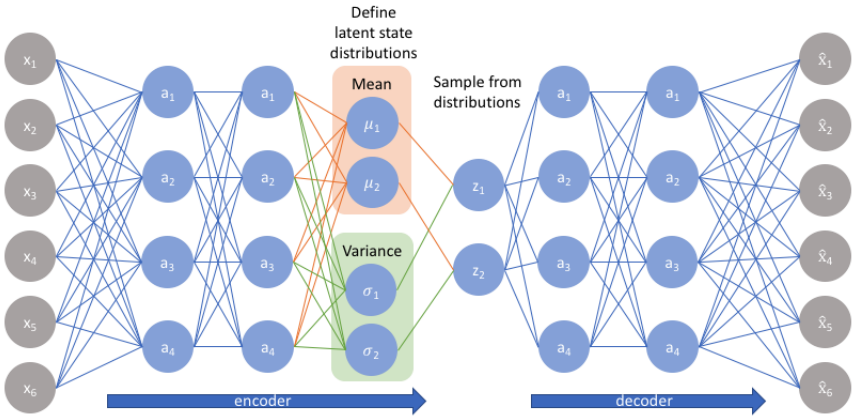

Figure 2.9: Variational autoencoders architecture.

As figure 2.9 shows, the encoder outputs two vectors: a vector of means μ, and another vector of standard deviations σ. The mean vector controls the horizontal position of the encoding, while the standard determines its area. This way, even for the same input, the encoding output will slightly vary on every single pass. So the decoder can decode specific encodings in the latent space as well as encodings that slightly vary, as the decoder is exposed to a range of variations of the encoding of the same input during training.

2.4.2 Generative adversarial networks

On the contrary to VAEs, generative adversarial networks (GANs) do not work with any explicit density function; instead, they take a game theoretic approach; they learn to generate from training distribution through a two-player game. GANs are based on a game scenario in which two neural networks pit against each other. The generator produces new data from random noise sampled using a distribution. The new data are fed to the discriminator along with the real examples from a training data set. The discriminator plays the role of a cop and the generator the role of a counterfeit, and the cop mission is to distinguish between real and fake data. Both the generator and the discriminator try to improve their precision until the discriminator is unable to make the difference between the real examples and the fakes ones. Conceptually, the discriminator in GAN guides the generator on what data to create. Figure 2.10 describes GAN's training.

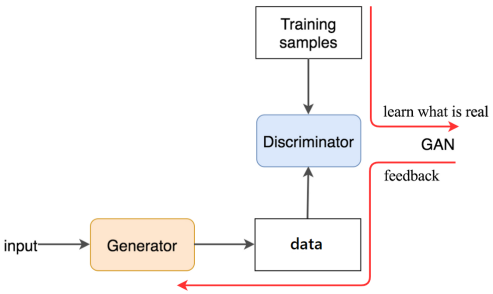

Figure 2.10: Generative adversarial network training illustrated. [49]

As figure 2.10 shows, by training with real examples and generated examples, GAN builds a discriminator to learn the features that make data real. Then the same discriminator will provide feedback to the generator to create data that look like real ones.

The discriminator is trained like a deep network classifier. The output is the probability that the input is real. Through this process, it identifies the features that contribute to real data by calculating the maximum likelihood of the observed data. Both networks are trained jointly in a minimax game where the generator wants to minimize a value V while the discriminator wants to maximize it. Equation 2.8 describes the objective funtion of GANs.

$$min_G max_D V(D,G) = E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.8}$$

$\log D(x)$ is the discriminator output for real data x, so the first part is the likelihood of real data being real, and the second one, z are the samples from the generator and the term $D(G(z))$ is the output of the discriminator for generated fake data.
GAN training uses gradient ascent on the Discriminator to maximise the objective,

while it performs gradient descent on the Generator to maximize it, and it takes only the part related with z. However in practice the gradient descent value on the generator does not help so well. An alternative is to perform gradient ascent on the generator and maximize the likelihood of the discriminator being wrong as shown in equation 2.9.

$$maxE_{z\sim p_z} \log D(G(z)) \tag{2.9}$$

Training alternates between k steps of optimizing D and optimizing G on the mini-batch. The process continues until the generator produces good quality examples.

2.5 Deep neural network with attention

2.5.1 Attention for deep neural networks

Attention

The motivation behind the attention mechanism is because a system needs different information at different time steps to capture the meaning of a sentence. The attention mechanism takes two sentences, turns them into a matrix. The words of one sentence form the columns, and the words of another sentence form the rows, then it identifies relevant context. It is also possible to put the same sentence along with the columns and the rows, to understand how some parts of that sentence relate to others. A neural network provided with an attention mechanism can understand what it is referring to. That is, it knows how to disregard the noise and focus on what is relevant and connect related words.

Transformers

There are three kinds of dependencies that we need to pay attention to in natural language processing: the dependencies between the input and output tokens, between the input tokens themselves, and between the output tokens themselves. The classic attention mechanism the problem of dependencies between the input. The Transformer [50] on the other hand, solved the three kind of dependencies. Figure 2.11 describes the Transformer architecture.

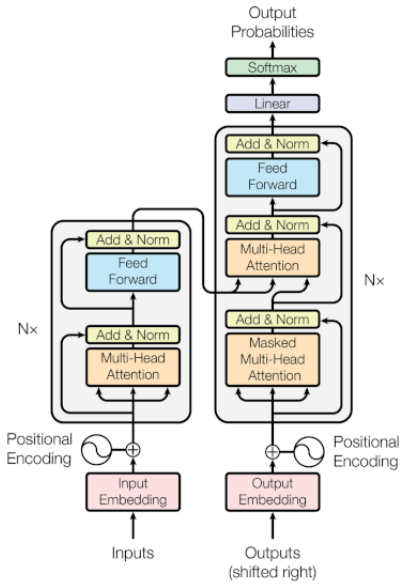

Figure 2.11: Transformer architecture [50].

The Transformer uses an encoder-decoder architecture. The left component is the encoder, and the right one is the decoder component. Each component has six blocks, which are themselves divided into smaller ones, which we will call sub-layers.

The encoder has two sub-layer: a self attention layer that helps the encoder look at other parts as it encodes a specific word; and a feed-forward network. The self-attention has a mechanism called Multi-Head Attention (MHA) which computes the self attention multiple times to learn diverse representations. A single attention head has a simple structure: it applies a linear transformation to its query, key and value vectors, calculates the relevance score between the key and the query, uses a softmax to normalize the values, then uses it to weight the values and sum them up. The key, query and value vectors are the result of the multiplication of the word embeddings by three matrices. The MHA just performs this operation in par allele, concatenates the attention of each head, then uses a linear transformation on the concatenated result. As for the type of attention, the Transformer uses the scaled dot product attention which is computed as follow (see equation 2.10):

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.10}$$

Where Q is the matrix of queries packed together, K and V are the matrices of keys and values packed together. Finally, d_k represents the dimensionality of the queries and keys. The dot product grows in magnitude with large values of d_k, it may push softmax values into regions where it the gradient is very small. To deal with this, it uses this scaling factor of $\sqrt{d_k}$ to rescale the value.

Between each sub-layer, there is a residual connection followed by a layer normal-

ization. The residual connection prevents the vanishing gradient problem, while the normalization prevents the values from changing too much, allowing faster training and better generalization. The formula is shown in the equation 2.11.

$$LayerNorm(x + sublayer(x)) \tag{2.11}$$

$sublayer(x)$ is the function that is being generated from the sublayer. For any vector x, the layer normalization is computed as shown in the equation 2.12.

$$LayerNorm(x) = \gamma\frac{x - \mu}{\sigma} + \beta \tag{2.12}$$

Where μ, σ are the mean and standard deviation of the elements in x, scale γ and bias vector β are parameters.

The feed-forward network (FFN) is a fully connected network applied to each position independently. This sub-layer is a two-layer feed-forward network with a ReLU activation function. The first layer is four times the size of the model. This seems to give the transformer enough representational capacity. The second layer will project the output of this first layer into the original size. Given a sequence of vectors $h_1, .., h_n$ the computation of a position-wise FFN sub-layer on any h_i is defined as shown in the equation 2.13.

$$FFN(h_i) = ReLU(h_iW^1 + b^1)W^2 + b^2 \tag{2.13}$$

Where W^1, W^2, b^1, and b^2 are parameters.

The decoder is very similar to the encoder but has a Multi-Head Attention layer named masked multi-head attention; the reason it is called Masked Multi-Head Attention is that we need to mask the inputs to the decoder from future time-steps to prevents future words from being part of the attention. Besides, there is an attention layer between the Masked Multi-Head attention and the feed-forward network; this helps the decoder look at the relevant parts of the input while decoding.

For the input, the embedding happens at the bottom most encoder. The MHA cannot make use of the position of the word in the sequence. The Transformer takes the embedding, adds to the embedding vectors that follow a specific pattern that the model learn. Each pattern describe a particular position.

The top most decoder outputs a vector of floats, how is it turned into words ? That's the job of the final linear layer which is followed by a softmax layer. The linear layer projects the vector in a much larger one called logits. It has the size of the vocabulary. The softmax then turns the values inside the vector (scores) into probabilities. Finally the word which has the highest probability is chosen, or by using an algorithm called beam search.

2.5.2 BERT

BERT [51] (Bidirectional Encoder Representation from Transformers) is a pretrained language model built on top of the Transformer blocks. There are many variants available, for example, BERT-Base has 12 layers and BERT-Large has 24 layers, and each layer is a Transformer encoder. BERT is unique because it reads words from both directions at once. Most language models are trained by predicting the next word in a sequence, but since it is a unidirectional strategy it limits the context integration. Hence BERT used different strategies: Masked Langage Model (MLM), and Next Sentence Prediction (NSP); these are jointly trained and the loss is the combination of the two.

Next sentence prediction

BERT is fed with two sentences as an input and is supposed to predict if the second sentence is the subsequent sentence in the document. More precisely, 50% of pairs have the second sentence as the next one, and the 50% left do not. BERT has specials tokens: the [CLS] token and [SEP] token to separate sentences, it has also special embeddings indicating if it's sentence A or sentence B, plus the positional embeddings as in the Transformer. The entire input during NSP goes through the model, the [CLS] token aggregates feature sentences, then the output of the top [CLS] is fed to a softmax that calculates the probability. This is shown in the figure in figure 2.12.

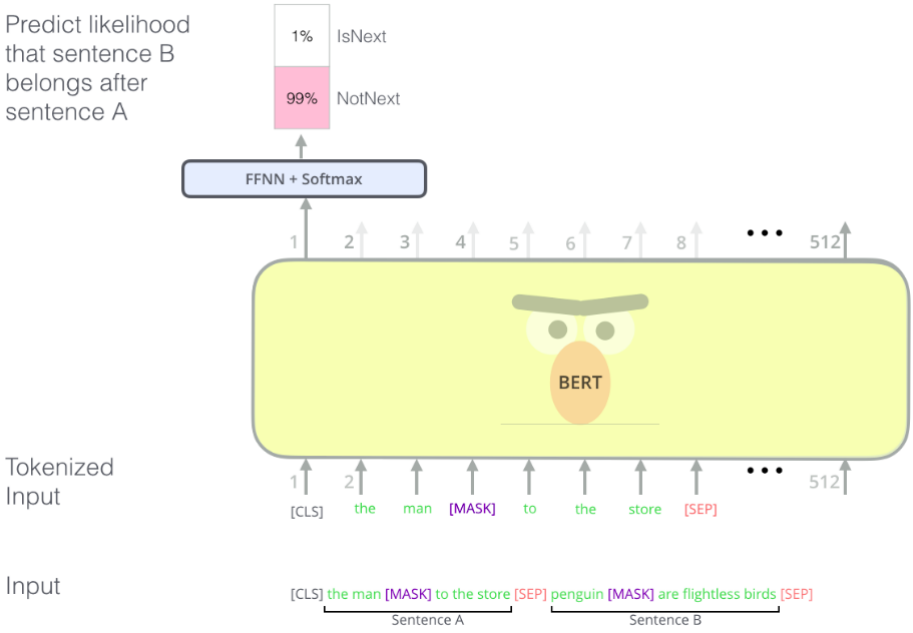

Figure 2.12: Bert NSP task for training [51].

Masked language model

Some parts of the input are masked, and the model is supposed to correctly predict the masked tokens. In fact, 15% of all tokens were masked in each sequence at random. If a token is chosen, it is replaced 80% of the time with the [MASK] token, 10% with a random token, and 10% of the time it is kept unchanged. The final hidden vectors of the [MASK] tokens are fed to a feed-forward network followed by a softmax layer to predict the original token. The advantage of this method is that BERT does not know which words it has been replaced and which words were kept the same. This way it preserves a distributional contextual representation of every input token. Figure 2.13 describes MLM in BERT.

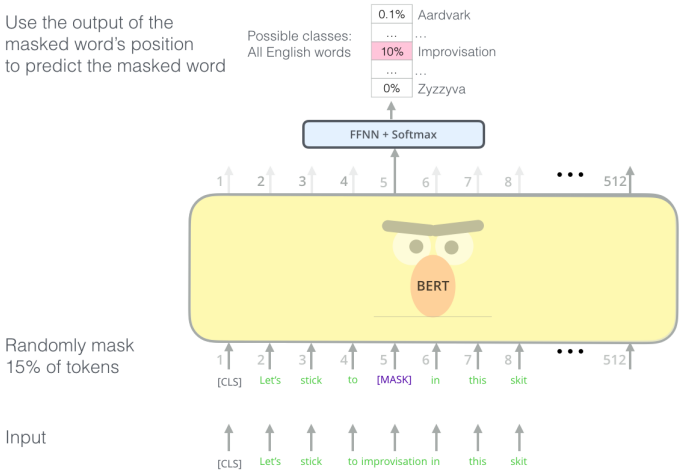

Figure 2.13: Bert MLM task for training [51].

BERT for fine-tuning

BERT can be used for multiple downstream NLP tasks. The advantage is that, since it is already pretrained, the training takes much less time compared to when trained from scratch. It has already a language understanding ability, hence it is easier to tune the weights. BERT can be used for classification, question answering, and any other task where the prediction at a certain position is allowed to look at other information from both directions.

BERT for feature extraction

The finetuning approach is not the only way to use BERT. It can also be used to extract features, then use these features as input to other models. The combination of the vectors from BERT layers depends solely on the task on which they will be employed.

2.5.3 Generative Pre-Training-2 (GPT-2)

GPT-2 [52] is a model based on GPT [53] and is a large transformer-based language model that consists of solely stacked decoder blocks from the transformer architecture. Like any traditional language model, GPT2 outputs one token at a time. After each token is generated, that token is added to the sequence of inputs, and that sequence is fed to the model as an input, and so on. There are many variations of GPT-2, each one of them has more layers and more attention heads than the original Transformer decoder. Figure 2.14 represents a high representation of some variants of GPT-2.

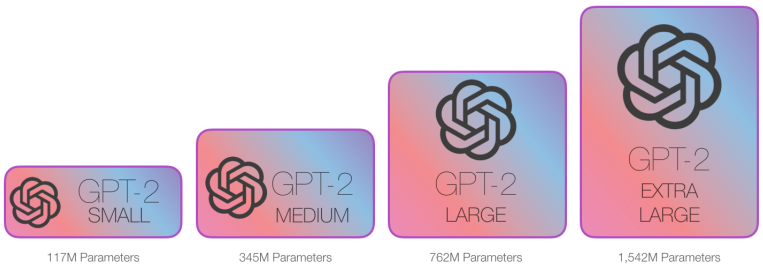

Figure 2.14: High representation of some variants GPT-2 [54].

As we can see, the smallest model has a stack of twelve decoder transformer blocks and a dimension = 768. And the biggest model has four times more layers and a dimension = 1600. Like classic language models, it was trained to predict the next word in a sequence with a massive 40GB dataset called WebText that has a vocabulary size of 50,000. During the evaluation, GPT-2 holds on to the key and value vectors of the previous tokens that were already processed, so that it will have fewer tokens to process in future time steps; besides, the way it works is exactly like the original Transformer decoder.

2.6 Conclusion

Machine learning is nothing new, but the past years have been the biggest leaps and bounds in terms of advances in automatic text summarization using this approach. Factors like growing volume and massive data available, cheaper, more powerful computational processing, and affordable data storage made it only natural to quickly and automatically produce models that can process bigger, more complex data, and deliver faster, more accurate results, even on a very large scale.
Many architectures can be used and combined for our task although recent models using Transformers seem to have greater potential. The following chapters will address our contribution. In the next chapter, we will therefore present the models we used and the design details.

Chapter Three

Methods of automatic text summarization

3.1 Introduction

After presenting the overall concepts of automatic text summarization, this chapter covers some of the works that have been proposed for this problem. In the previous chapter, we mentioned the five common approaches, namely the statistical approach that came first, the graph-based approach, the linguistic approach, the compression approach, and the machine learning approach. To classify the existing works, it is possible to use these two criteria alongside the approaches classification [13]:

- If it is single or multi-document summarization.
- Whether it is abstractive or extractive summarization.

The following table summarizes our classification (see table 3.1). Sections 2.2 and 2.3 both present single-document summmarization works and multi-document summmarization ones. Section 2.4 discuss the works of the approach we will follow in our work. Finally, section 2.5 concludes this chapter.

Input documents	Output summary	Approaches
Single-document summarization	Extractive summary	Statistical, Graph-based, Linguistic Compression, Machine learning
	Abstractive summary	Statistical, Graph-based, Linguistic Compression, Machine learning
Multi-document summarization	Extractive summary	Statistical, Graph-based, Linguistic Compression, Machine learning
	Abstractive summary	Statistical, Graph-based, Linguistic Compression, Machine learning

Table 3.1: Classification of automatic text summarization works.

3.2 Single-document summarization

3.2.1 Extractive summarization

Statistical approach

The first work of automatic summarization follows the statistical approach. Baxendale [55] used a corpus of 200 scientific paragraphs to test the impact of the position of sentences in a paragraph on its importance; and he found that in 85% of cases, the first sentence is the most important for the paragraph. The last sentence is the most important in 7% of cases. He explained the result by supposing that topic sentences are more likely occur very early or very late in a document. He took the first and last sentence in each paragraph to form the summary. Despite its simplicity it was a kind of reasonable method for the type of documents he was trying to summarize.

Luhn [56] proposed a more complex way to generate summaries and used a feature called term frequency which supposes that a term is important if it occurs many times. During the preprocessing, he removed stop words and stems so that it can objectively calculate the frequency of each word. He then grouped the sentences based on the concentration of the keywords. The group which has the most significant words is used to score the sentence, and the summary is generated by extracting the highest score sentences. Term frequency formula is described below (see equation 3.1).

$$tf(t,d) = \frac{f_d(t)}{\sum_{i=1}^{|d|} |t_i|_d} \tag{3.1}$$

Where $tf(t,d)$ is the term frequency of the term t in document d. $f_d(t)$ is the number of appearances of the term t in the document and $\sum_{i=1}^{|d|} |t_i|_d$ is the number of the distinct words in document d.

The main advantage of term frequency is that it is language-independent. However; it may consider some terms that do not thoroughly characterize the topic being addressed as being relevant. For instance, the words informatics and computer are frequent in computer science topics. Another problem is the case when some words are not so frequent but are supposed to be relevant.

To solve the problem discussed above, Salton and Yang [57] proposed another feature called $tf * idf$ where tf refers to term frequency and idf represents the inverse document frequency. It expresses that a word is more important when it is more frequent in the analyzed document and not frequent in the corpus of analyzed documents (see equation 3.2).

$$idf(t) = \log \frac{|D|}{|\{d : t \in d\}| + 1} \tag{3.2}$$

Where $|D|$ is the number of documents in the corpus and $|\{d : t \in d\}|$ is the

number of documents containing the word t. So, an inverse document frequency factor is incorporated to reduce the weight of terms that occur very frequently in the document set and increase the weight of terms that occur rarely. This way, the less frequent but important words would not be ignored.
Finally, the factor $tf * idf$ will be calculated as in the equation 3.3.

$$tf * idf(t, d) = tf(t, d) * idf(t) \tag{3.3}$$

Edmundson work [58] used some of the already mentioned features; he used the position of a sentence in a document in the same way as Baxendale [55], he also looked at term frequency, the same way as Luhn [56]; but he also added a feature called cue words. Cue words such as our "results indicate", "in this article", etc. were used to determine relevant information to the summary. These words were manually selected and saved in a dictionary that comprises three sub-dictionaries: bonus words, stigma words, and null words. Bonus words include superlatives, adverbs, etc. and add value to the sentence. Stigma words are words that negatively affect sentence importance. They include anaphoric expressions, belittling expressions, etc. Finally, null words represent neutral or irrelevant words to the sentence and are often stop words. The score of a sentence using cue words was the sum of the weights of its cue words shown in the equation 3.4.

$$Score_{cue}(si) = \sum_{w \in si} cue(w) \tag{3.4}$$

Where $cue(w)$ is the weight of the word w concerning the keyword dictionary (see equation 3.5).

$$cue(w) = \begin{cases} b > 0 & if(w \in Bonus) \\ \delta < 0 & if(w \in Stigma) \\ 0 & otherwise \end{cases} \tag{3.5}$$

Another thing he looked at was the structure of the document: is it a headline? Is it a title? Is it the first sentence after a title? As for the score of the sentence, he created a linear combination of these four features.

Besides, Fattah and Ren [59] defined positive cue words as words that have frequent occurrences, while negative words are the ones that are most unlikely to be included in a summary. To achieve that, we calculate the score of a sentence given a word w and a summary S by using the equation 3.6.

$$Score_{cue}(si) = \frac{1}{|si|} \sum_{w \in si} tf(w) * P(si \in S|w) \tag{3.6}$$

Where $P(si \in S|w)$ is the probability that the sentence si is in the summary S knowing the word w. The formula 3.7 describes how to calculate $P(si \in S|w)$, we can

get it using the training corpus.

$$P(si \in S|w) = \frac{P(w|si \in S) * P(si \in S)}{P(w)} \tag{3.7}$$

They computed the score of a sentence using negative cue words in the exact opposite way, shown in equation 3.8.

$$Score_{cue}(si) = \frac{1}{|si|} \sum_{w \in si} tf(w) * P(si \notin S|w) \tag{3.8}$$

Another interesting technique is to combine the position with other features. Ouyang et al. [60] used word position to calculate the score of a word according to its occurrences in the whole text. Not just according to other words in the sentence. They used four different functions:

- **Direct proportion** attributes a score of 1 on the first appearance and $\frac{1}{N}$ to the last one such as N is the number of words in that sentence.
- **Inverse proportion** assigns $\frac{1}{i}$ as a score where i is the position of the word. Therefore, smaller positions are punished.
- **Geometric sequence** scores the appearance of a word as the sum of the scores of all its occurrences; hence the equation is $(\frac{1}{2})^{i-1}$.
- Finally, the **binary function** gives more importance to the first appearance of a word, and the others are evenly less important. The value for the first appearance is equal to 1 and $\lambda \ll 1$ for the rest.

The final score of each sentence is calculated as follows (see equation 3.9):

$$Score(s) = \sum_{w_i \in s} \frac{\log(freq(w_i) * pos(w_i))}{|s|} \tag{3.9}$$

Where $pos(w_i)$ is one of the four functions, $freq(w_i)$ is the frequency of the word and $|s|$ is the length of the sentence s.

Rather than using the position in a sentence, Fattah and Ren [59] adopted the position in a paragraph. The first sentences are given a score of N-i where N is the number of important sentences and i is the position of the given sentence in a paragraph. For example, assuming that we consider only the first four sentences, the score of each sentence is given by the equation 3.10.

$$Score_{pos}(si) = \begin{cases} 5 - i & if(i \leq 4) \\ 0 & otherwise \end{cases} \tag{3.10}$$

Frequent itemset mining is a well-established data mining technique to discover correlations among data [61]. In automatic text summarization, itemsets are sets of terms

extracted from sentences, and those which appear in many sentences determine the frequent itemsets showing a significant correlation in the documents. MWI-Sum [61] is a multilingual summarization algorithm that replaces the traditional itemsets with weighted ones. It tries to create weighted itemsets representing word combinations with significant high-frequency and relevance. Each itemset represents a relevant concept within the news collection, itemsets are frequent ones that are not already included by earlier selected sentences. Hence, the proposed selection method picks the document sentences covering meaningful concepts yet not covered by any other sentence.

Graph approach

Salton et al. [62] constructed a graph of similarity using document paragraphs. Each paragraph represents a node that is connected to another when their similarity exceeds a given threshold. The authors defined a feature called bushiness which is the number of a node's connections. The most scored paragraphs in terms of bushiness are extracted to form a summary. Equation 3.11 describes the score based on the number of arcs, where $G = \{S, A\}$ is the graph of similarities between the sentences, S is the set of sentences and A is the set of arcs.

$$Score_{\#arc} = \{sj : a(si, sj) \in A / sj \in S, si \neq sj\} \tag{3.11}$$

This equation means that the score of any sentence si is the number of the nodes that sentence is linked to. Also, $a(si, sj) \in A$ means that the sentence si has an arc with a sentence si, and $si \neq sj$ makes sure that we do not count the arc linked to the same sentence.

Thakkar, Dharaskar, and Chandak [63] proposed a method that uses the shortest path algorithm for the summary generation. First, they built a graph model for representing the text as a way to connect text entities in the graph to form meaningful relations. Then, they applied a ranking graph-based algorithm to score each vertex of the generated graph in the previous step. Finally, they used the shortest path algorithm on the graph to generate the text summary.

Linguistic approach

Paice [64] defined indicators as "commonly occurring structures which explicitly state that the sentences containing them have something important to say about the subject matter or the message of the document", so this kind of expressions can help to determine which sentences are more likely to be relevant.

In the work of Orasan [29], the author used an anaphora resolution to enhance the informativeness of summaries. Sentences containing pronouns rather than words lead to inaccurate score calculation. Anaphora resolution helps to increase the frequencies

of words associated with these pronouns and produces more accurate frequency calculations. The author used a term frequency algorithm to score sentences alongside six anaphora resolution methods.

Also, the structure of discourse can be used through rhetorical relations between sentences to generate summaries. The rhetorical structure represents relations between various pieces of sentences in the body of each section. For example, Ono, Sumita, and Miike [65] developed a system for Japanese expository writings based on rhetorical structure extraction. The system first represents the rhetorical structure by two layers: an intra-paragraph whose units are sentences, and inter-paragraph whose units are paragraphs. Then, it extracts the rhetorical relations by relying on connective expressions, anaphoric expressions and idiomatic expressions. The figure 3.1 shows examples of rhetorical relations for representing the rhetorical structure.

Relation	Expressions
serial (<SR>)	*dakara* (thus)
summarization (<SM>)	*kekkyoku* (after all)
negative (<NG>)	*shikashi* (but)
example (<EG>)	*tatoeba* (for example)
especial(<ES>)	*tokuni* (particularly)
reason (<RS>)	*nazenara* (because)
supplement (<SP>)	*mochiron* (of course)
background (<BI>)	*juurai* (hitherto)
parallel (<PA>)	*mata* (and)
extension (<EX>)	*kore wa* (this is)
rephrase (<RF>)	*tsumari* (that is to say)
direction (<DI>)	*kokode wa . . . wo noberu* (here . . . is described)

Figure 3.1: Example of rhetorical relations [65].

Finally, the system generates a summary of each section by examining its rhetorical structure. For that, to determine the important text parts, it uses a penalty score defined over different rhetorical relations to exclude non-important sentences. The important trait of the generated summary is that since they are composed of the rhetorically consistent units, the summary does not contain incomplete sentences.

Kikuchi et al. [66] proposed a system for single document summarization based on a nested tree structure. They incorporated dependencies between words and dependencies between sentences using a rhetorical structure. They first represented a document as a nested tree formed of two types of tree structures: a document tree and a sentence tree. Figure 3.2 illustrates a nested tree structure of a document.

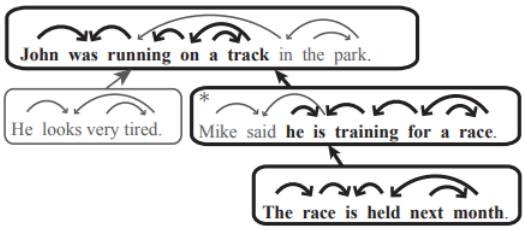

Figure 3.2: A source document represented as a nested tree [66].

The document tree has sentences as nodes and head modifiers relationship obtained by using the rhetorical structure theory as edges. The sentence tree has words as nodes and head modifiers relationship between words obtained by the dependency parser as edges. A dependency parser examines the grammatical structure of a sentence and builds relationships between head-words and words that modify those heads. Then, they expressed the problem of single document summarization as a combinatorial optimization problem, which will compact the tree and generate the summary.

The linguistic approach is harder than the statistical approach to be implemented and takes more time to generate a summary [13]. Nenkova and McKeown [27] suggested using it as a post-processing task to improve the linguistic quality of the generated summary rather than a processing one. According to the authors, it is unclear how much this approach can improve content selection compared to the methods using no linguistic relations.

Compression approach

Sentence compression aims to retain the most salient information of a sentence and delete the least critical information, rewritten in a short form [67]. In Jing and Mckeown [68], the authors claimed that it is often used by professional summarizers. They found out that 78% of the summary sentences are taken from the input document, and more than half were compressed. Sentence compression works can be classified into two approaches: a sentence deletion approach and an abstractive approach.

The deletion based approach is based on the removal of unnecessary parts of the input which is generally on a sentence level. That is, the result of the compression is a subsequence of the source sentence. Consequently, those systems produce an extractive summary [69]. For instance, Jing [23] work is one of the earliest ones in this direction. He presented a sentence compression system that uses several knowledge sources. The purpose is to compress by removing as many unnecessary parts of the sentences as possible without detracting from the main idea that the sentences convey. These sources include syntactic knowledge, context information, and statistics extracted from a corpus of professional summaries.

In Cohn and Lapata [70], instead of shortening the sentence by removing words or components, they introduced additional operations such as substitution, reordering, and insertion. Filippova [71] proposed a method based on the shortest path in word graphs to compress many sentences into a single short one to generate the summary. The sentences are clustered based on sentence similarity or relatedness.

Deep learning can also be applied to compress sentences. Filippova et al. [72] use Long Short Term Memory (LSTM) models with word embeddings to perform a deletion based sentence compression. Their method performs very well either in terms of readability or informativeness, even without using syntactic information (Part of Speech (PoS), Named Entity (NE) tags, and dependencies).

Inversely, Wang and al. [73] suggested that the incorporation of syntactic information into a compression model would be useful. Thus, they proposed an LSTM model which uses syntactic information such as POS, tags, and parsing information. They demonstrated that it influences and leverages the robustness of a model in cross-domain application. One of the weaknesses of the deletion-based approach is its incapacity to produce expressive sentences. The abstractive approach to sentence compression solves this issue by paraphrasing the sentences. This approach will be discussed later in this section.

Machine learning approach

Machine learning can solve the problem of combining features in the statistical approach for extracting the key sentences. For instance, Yatsko, Starikov, and Butakov [74] tuned the features according to the input document's genre. They used forty-five parameters of various types: statistical, positioning, and discourse to identify those that are most meaningful for a given genre of the document. Then, using K-Means, grouped the documents in a corpus according to genres: scientific, press, and artistic. Finally, they used the corpus to identify the specific parameters for each genre by assigning a weight to each one. Hence, the system can distinguish the genre of the input document and execute the adequate model of scoring.

Wong, Wu, and Li [75] proposed a learning-based approach to combine four criteria: surface features, content features, event features, and relevance features. Surface features base on the structure of documents or sentences, content features measure a sentence based on content carrying words, event features represent sentences by events they contain, and relevance features evaluate a sentence from its relatedness with other sentences. After examining feature vectors of sentences, they employed a supervised learning classifier. The system re-rank candidate sentences considering the length of the final summary that is fixed. Finally, they extracted the top sentences to generate the final summaries.

Besides, many works use machine learning in automatic text summarization as a classification problem. Kupiec et al. [76] introduced an automatic text summarization system based on Bayes classification. They developed a classification function, naive-Bayes classifier, to classify the sentences as summary sentences and non-summary sentences based on the features they have with a probability score, given a training set of documents and their extractive summaries selected manually. For each sentence s and a set of features F, equation 3.12 calculates the sentence probability.

$$P(s \in S|F_1, F_2, ..., F_k) = \frac{P(F_1, F_2, ..., F_k|s \in S) * P(s \in S)}{P(F_1, F_2, ..., F_k)} \tag{3.12}$$

The equation tells us that the probability that a sentence s belongs to the summary S given that it has a set of features is the same as the probability that these features occur if the sentence is in the summary times the probability that the sentence is in the summary, divided by the probability that these features occur together. For $P(F_i)$, they estimated it from the corpus. Assuming the independence of the features, the equation then becomes as shown in equation 3.13.

$$P(s \in S|F_1, F_2, ..., F_k) = \frac{\prod_{j=1}^{k} P(F_j|s \in S) * P(s \in S)}{\prod_{j=1}^{k} P(F_j)} \tag{3.13}$$

This equation means that the probability that the sentence ends up in the summary given a certain set of features is the product of the probability of the individual features that appear for the sentence and appear in the summary. As for the features used in their summarizer, they settled for: sentence length, fixed phrase, the position of a sentence in a paragraph, thematic word, and uppercase word. Their results showed that position was the most influential individual feature, and the best combination of features was the position, fixed phrase, and sentence length. One of the limitations of this method is that the features are rarely independent. Osborne [77] has addressed this problem using a linear logarithmic model. The use of a classifier based on the principle of maximum entropy performed lower than naive Bayes classifier but surpassed it when using a prior probability with the maximum entropy (even when we extend naive Bayes with a similar prior); and instead of ordering the sentences using their probabilities, it used a learning algorithm to classify sentences in two classes: the first one contains the summary sentences, and the second one contains the other sentences.

Reinforcement learning is to train an agent to behave correctly by rewarding it when taking the right decision [78]. In [79], they took advantage of simple embedding features in reinforcement learning to generate an extractive summary for a single document and reduce the burden of hand-crafted features. According to the authors, embedding is a useful technique to build features to represent words, sentences, or documents. In their work, they used content embeddings and position embeddings and considered sentences as the unit of summarization. The content vector described the

meaning of a sentence, and the position vector designated the position of a sentence in the document. Their method demonstrated that embedding features are useful for reinforcement learning.

In [80], authors proposed a system that improves performance for both extractive and abstractive single-document summarization following the encoder-encoder-decoder paradigm. The extractive model classifies each sentence in a document as being summary worthy or not. To enhance the sequence classification process, they encoded the input document with a Transformer. The input is the concatenation of the vector representation of the document sentences. Each sentence representation is the average of the vector representation of its constituent words. The transformer encoder is a six stacked identical layers. The extraction model has a final softmax layer that gives the probability of including a sentence in the summary. Figure 3.3 illustrates the extractive model architecture.

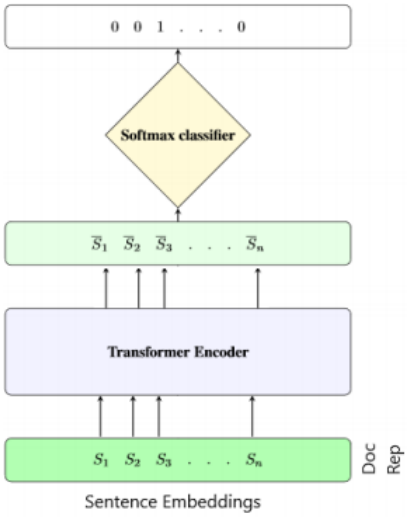

Figure 3.3: The extractive model architecture [80].

Besides, the model called BERTSUM [81] is a variant of BERT (Bidirectional Encoder Representations from Transformers) [51] for extractive summarization. The global architecture of this model comprises two components: a BERT large component, and a summarization component. BERT large consists of twenty-four transformer encoder layers. It expects an input of size 1024 and has sixteen head in the multi-head attention sublayer.

In vanilla BERT, The [CLS] is used as a symbol to aggregate features from one sentence or a pair of sentences. While in BERTSUM, they modify the model inserting external [CLS] tokens at the start of each sentence, and each [CLS] symbol collects features for the sentence preceding it. The vector T_i which is the vector of the $i-th$ [CLS] symbol from the top BERT layer is used as the representation for sentence i.

Also since BERT has only two labels for indicating sentences, those labels are hard to use for extractive summarization to distinguish between sentences. Hence, they modify the segment embeddings and assign EA or EB depending on whether the sentence is odd or even. For example, a document with five sentences [sent1, sent2, sent3, sent4, sent5], would assign embedding [EA, EB, EA, EB, EA]. The intuition behind this modification is to learn document representations hierarchically where lower Transformer layers represent adjacent sentences, while higher layers, in combination with self-attention, represent multi-sentence discourse. Figure 3.4 represents an overview of the architecture of the BERTSUM model.

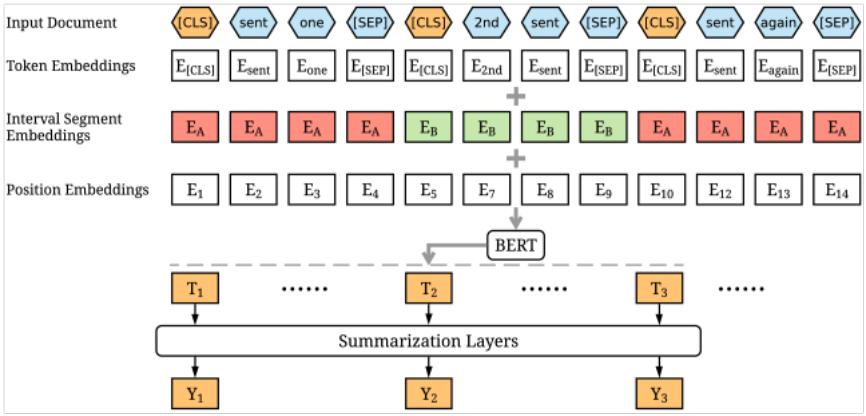

Figure 3.4: The architecture of BERTSUM [81].

They used the sentence representation from BERT as an input to other layers which are specific to summarization. Those layers will capture document-level features to generate the summaries. For each sentence, they calculate the final predicted score and compute the error using the Binary Cross Entropy of the generated label output $\hat{Y}_i$ against the gold label Y_i. These summarization layers are jointly fine-tuned with BERT. The formula of Binary Cross Entropy is described in equation

$$BCE = -\frac{1}{N} y_i * \log(p(y_i)) + (1 - y_i) * \log(1 - p(y_i)) \tag{3.14}$$

Where y_i is the gold label and $p(y)$ is the predicted probability of the sentence being into the summary and N is the number of scalar values in the model output.

The summarization layer is a built as two Transformer layers. The extraction of document-level features from the BERT outputs is described in the equations 3.15 and

3.16:

$$\tilde{h^l} = LN(h^{l-1} + MHAtt(h^{l-1})) \tag{3.15}$$

$$h^l = LN(h^l + FFN(h^l)) \tag{3.16}$$

Where $h_0 = PosEmb(T)$ and T are the sentence vectors output by BERT; $PosEmb$ is the function of adding positional embeddings to t; LN is the layer normalization operation [82]; $MHAtt$ is the multi-head attention operation [50]; the superscript l indicates the depth of the stacked layer. The final output layer is still a sigmoid classifier (see equation 3.17) :

$$\hat{Y_i} = \sigma(W_o h_i^L + b_o) \tag{3.17}$$

Where h^L is the vector for the $sent_i$ from the top layer (the $L-th$ layer) of the Transformer which in this case is layer 2.

The model is trained on the CNN/Daily Mail and NYT annotated corpus [83]. Since the dataset from NYT annotated corpus is created for abstractive summarization, they create a new ground truth using a greedy algorithm for generating an oracle summary for each document. The algorithm greedily selects sentences that can maximize the ROUGE [35] scores as the oracle sentences.

3.2.2 Abstractive summarization

Compression approach

For the compression approach in abstractive summarization there are fewer works compared to extractive summarization. We can cite the work of Choi et al. [69] who proposed an abstractive sentence compression method using event attention for compression sentences of news articles. Event attention focuses on the event words of the source sentence in generating a compressed one. The attention score represents the combination of the event attention and the global attention used to understand the global information of a sentence.

Machine learning approach

Abstractive summarization systems make use of deep learning in multiple ways. Rush, Chopra and Weston [84] implemented successfully deep learning to automatic text summarization by applying a local attention-based model to their summarizer called NAMAS. The attention-based model generates the summaries by taking into consideration all words when processing a particular word. Some of the limitations of their method are: it processes only documents with a size of around 500 words and generates very short summaries.

In the same direction, Nallapati et al. [85] used an attention model as an encoder-decoder recurrent neural network (RNN) [86]. The encoder is a bidirectional GRU-RNN [87] and the decoder is a unidirectional GRU-RNN. The decoder has the same hidden size as the encoder and a softmax layer over the target vocabulary to generate words.

To identify key concepts and entities in the document, they captured linguistic features such as part-of-speech tags, named-entity tags, and $tf * idf$ statistics of the words. Finally, for each word in the source document, they looked-up its embeddings from all of its associated tags and concatenated them into a single long vector, as shown in figure 3.5. On the target side, they used only word-based embeddings as the representation. As we can see, there is one embedding vector each for POS, NER tags, TF and IDF values, which are concatenated together with word-based embeddings as input to the encoder.

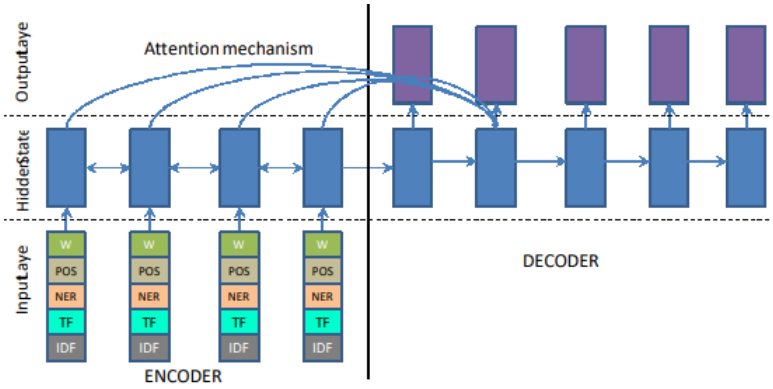

Figure 3.5: Feature-rich-encoder [85].

The attention is shared between words and between sentences. The word-level attention is further affected by sentence-level attention for capturing important sentences and important words within those sentences. In the decoding process, they only used the words appearing in the source document. Then to include new words, they added a layer of word2vec nearest neighbor in the input. They used a layer which they called Switching Generator/Pointer layer to decide if the next output is a word from the input or a new word. The decoder has a 'switch' that chooses between using the generator or a pointer at every time-step. When the switch is on, the decoder generates a word from its target vocabulary; and when it is off, the decoder instead generates a pointer to one of of the words in the input document. Figure 3.6 illustrates the Switching generator/pointer model mechanism.

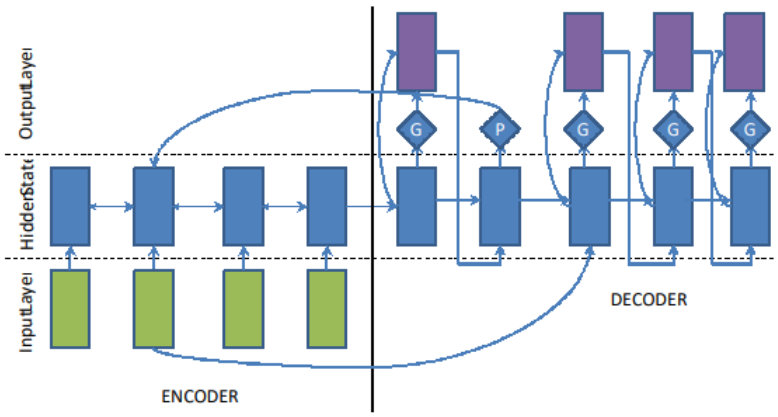

Figure 3.6: Switching generator/pointer model [85].

When the switch shows 'G', the generator consisting of the softmax layer is activated to produce a word, and when it shows 'P', the pointer copies the word from one of the source document positions. The pointer uses the embedding from the source as input for the next time-step as shown by the arrow from the encoder to the decoder at the bottom.

Sequence-to-sequence models with attention require a lot of time processing. To address this problem, Ling and Rush [88] assumed that not every word of the source is necessary for generating a summary. Hence, they tried to reduce the amount of computation performed on the source by using two-layer hierarchical attention. The first layer selects the important parts from the input document using a mecanism called hard attention. Then this layer feeds it/them into the second one which is a sequence-to-sequence model. Basically, They divided the document into chunks of text, sparsely attending to one or a few chunks at a time using hard attention, then applied full attention over those chunks. Unfortunately, their method failed to surpass the standard sequence-to-sequence model but showed promise for scaling up existing models to larger inputs.

Li et al. [89] proposed a sequence-to-sequence encoder-decoder model where the encoder is a variational encoder that has gated recurrent units, and the decoder is a variational decoder that has gated recurrent units too. The input is a variable-length sequence representing the source text, and its embedding is initialized randomly and learned during the optimization process. The output is also a sequence that represents the generated abstractive summaries. For latent structure modeling, they added historical dependencies on the latent variables of Variational Auto-Encoders and proposed a deep recurrent generative decoder to extract the complex latent structures. The decoder generates the summary using both the hidden state variables and the latent structural information.

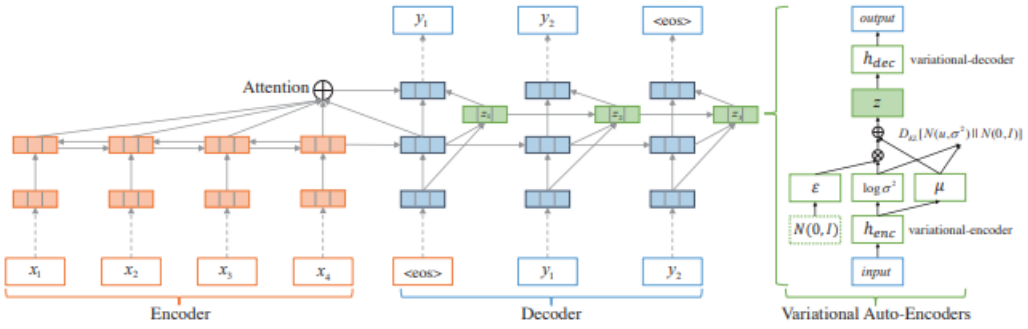

Figure 3.7: The deep recurrent generative decoder (DRGD) for latent structure modeling [89].

As figure 3.7 shows, in the decoder component, the input reflects not only the observed variable $y < t$ but also the previous latent structure information $z < t$; this allows the model to create effective representations for the generation of the next state. Besides, the architecture called the Transformer [90] works well for abstractive summarization, however its performance is quadratic, which makes the input processing as well as the training considerably slow. Sabran and Matton [91] introduced models that address the bottlenecks of the encoder.
The first one called Local Transformer divides the input sequence into chunks of a fixed size, and each chunk is processed independently. They made the cost linear instead of quadratic while preserving many benefits of self-attention by having a fixed-size window attention. The Local Transformer replaces one self-attention on the whole input by $\frac{n}{k}$ self-attentions on k tokens. Figure 3.8 illustrates the local Attention of the local Transformer.

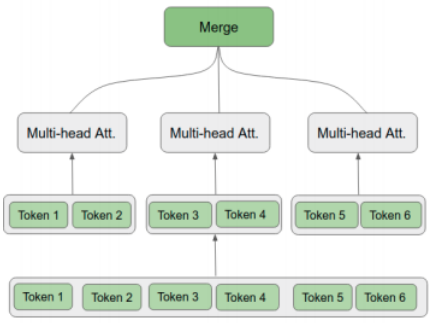

Figure 3.8: Local Attention [91]

This figure (3.8) clearly shows how the input is divided into chunks and how chunks are fed to self-attention. The main problem with the Local Transformer is that each chunk is processed independently, the information from other chunks cannot be used while processing a particular one. As a result, words that are at the edge of a chunk

can't see words at the other side of the chunk.

The Local Transformer with Shifts tries to solve that limit by shifting all chunks by half of their size in odd blocks of the transformer. As a result, information can travel between all parts of the input sequence. Figure 3.9 illustrates local Attention with Shifts.

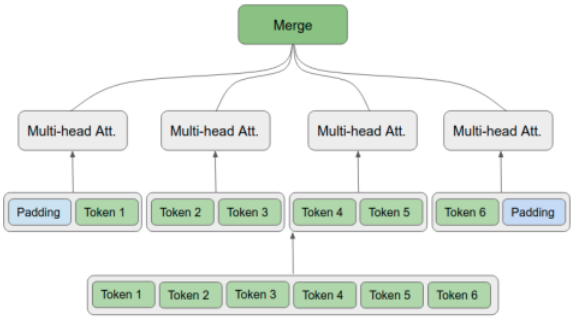

Figure 3.9: Local attention with Shifts [91]

The last model applies a stridden convolution layer to reduce the size of the input before feeding it to the Transformer. From a high-level perspective, the convolution abstracts small adjacent groups of words (typically 4), and the transformer processes the summarized inputs.

In addition to their extractive model, [80] also introduced the encode - encode - decode paradigm using Transformer and GRU-RNN for generating abstractive summaries. The reason behind is to improve the performance of the decoder by providing an interpretable and richly encoded sequence. The transformer encoder has the same implementation as in [90], except the inputs are sentence's vector representations, not document. Besides, those sentences represent the output of the extraction phase. The sentence representations in that module are not the average of the constituent word representations as in the extraction module but concatenated. That is, for each sentence, its vector representation is the concatenation of its constituent word embeddings. The second encoder is a unidirectional GRU-RNN whose input is the output of the transformer. The GRU-RNN encoder produces a fixed-state vector representation of the transformed input sequence. Finally, the output of the GRU-RNN encoder is fed to the decoder. At each time step, the decoder receives the previously generated word and hidden state. The softmax probability provides the output word, at each time step.

In [92] they proposed a model for abstractive summarization using a light version of BART [93] called Distilbart. BART is a pretrained denoising autoencoder implemented as a sequence to sequence model with a bidirectional encoder and a left to right autoregressive decoder.

The original BART architecture consists of a BERT like encoder with six layers or twelve for the large one, and a GPT2 [52] decoder with six or twelve layers for the large one; except that the encoder and decoder are connected by cross-attention, where each decoder layer performs attention over the final hidden state of the encoder output. And contrarily to BERT, BART does not use an additional feed-forward network before word prediction.

BART pretraining consists of two stages: the text is corrupted with an arbitrary noising function, then the sequence-to-sequence model learns to reconstruct the original text which is the standard way of training a denoising autoencoder. The noising function used is a combination of text infilling and sentence shuffling (permutation). They masked 30% of tokens in each text and permuted all sentences and ran the training on 160GB of news, books, stories, and web text for 500000 steps with a batch size of 8000. Finally, the training objective aims to optimize the reconstruction loss (cross-entropy) between the output and the original text.

The encoder of Distilbart is the same as the BART large and is composed of twelve layers. The decoder is a stack of six layers from BART large, more precisely the author took the layers: 0, 2,4, 7, 9, and 11 because he arbitrarily wanted to keep the first and last layer. Similarly to BART, each layer has a multi-head attention sub-layer with sixteen heads and a feed-forward network with 4096 hidden units. The activation functions in both the encoder and decoder are GELU [94] (Gaussian Error Linear Unit). And both expects as an input an embedding of a size equal to 1024. The summary generation is similar to a sequence to sequence model. The input is processed by the encoder, then the decoder autoregressively outputs one token at a time.

3.3 Multi-document summarization

3.3.1 Extractive summarization

Statistical approach

Nobata and Sekine [95] proposed different metrics to score sentences for multi-document summarization: $tf * idf$, sentence position, sentence length, and headlines (based on titles). They proposed three functions based on sentence position:

$$Score_{pos}(si) = \begin{cases} 1 & if(i < N) \\ 0 & otherwise \end{cases} \tag{3.18}$$

This function (see equation 3.18) gives a score of 1 to the first N sentences where i is the position of the sentence in the text. So it assumes the first ones as being the most relevant. As for the second function (see equation 3.19), the importance of sentences

is inversely proportional to their positions.

$$Score_{pos}(si) = \frac{1}{i} \tag{3.19}$$

Lastly, the third one (see equation 3.20) suggests that the first and last sentences are the most relevant:

$$Score_{pos}(si) = max(\frac{1}{i}, \frac{1}{N-i+1}) \tag{3.20}$$

Where N is the number of sentences and i is the position of the sentence. They also introduced two methods based on the title's words. The first method (see equation 3.21) measures the relevance between a title T and a sentence si using the $tf * idf$ values of words w (except for the stop words) in the title.

$$Score_{title}(si) = \frac{\sum_{w \in T \cap si} \frac{tf(w)}{tf(w)+1} idf(w)}{\sum_{w \in T} \frac{tf(w)}{tf(w)+1} idf(w)} \tag{3.21}$$

The second function (see equation 3.22) measures the score using only named entities instead of the nouns. According to the authors, the second method gives better results. For a named entity e, the following equation (3.22) is used to calculate the score of a sentence si:

$$Score_{title}(si) = \frac{\sum_{e \in T \cap si} \frac{tf(e)}{tf(e)+1} idf(e)}{\sum_{e \in T} \frac{tf(e)}{tf(e)+1} idf(e)} \tag{3.22}$$

Finally, for scoring sentences based on their length they proposed two functions. The first one assigns a score equal to the length of the sentence divided by a predefined length $Lmax$. If the length of the sentence is greater than $Lmax$, the score is one (see equation 3.23).

$$Score_{length}(si) = \begin{cases} \frac{Li}{Lmax} & if(Li \leq Lmax) \\ 1 & otherwise \end{cases} \tag{3.23}$$

The second one gives a negative score to penalize sentences that are shorter than a predefined value $Lmin$ (see equation 3.24). It turns out that it performs better than the first function.

$$Score_{length}(si) = \begin{cases} 0 & if(Li \leq Lmin) \\ \frac{Li-Lmin}{Lmin} & otherwise \end{cases} \tag{3.24}$$

Where si is a sentence and Li is the length of that sentence. They combine the scores of all metrics using a score function where each metric is also associated with a weight that determines its importance. The sentences are extracted following their order in the original documents.

Another statistical feature is by using clusters. Clustering is a data mining method that we can use in automatic text summarization as a mean to group sentences belonging to a document into clusters, or cluster sentences of different documents. For

instance, Radev, Jing, and Budzikowska [96] proposed the MEAD text summarizer. This summarizer works for both single and multi-document summarization. The first step consists of topic detection that aims to group news articles that describe the same event. For that, it represents the documents as $tf * idf$ vectors and removes words whose $tf * idf$ scores are below a threshold. Then, it applies a clustering algorithm, adding documents after another to clusters and recomputing the centroids by using the equation 3.25 described below.

$$c_j = \frac{\sum_{d \in C_j} d}{|C_j|} \tag{3.25}$$

Where: c_j corresponds to the j-th cluster and C_j is the collection of documents that belongs to that cluster. Thus, $|C_j|$ is the number of documents inside of it; finally, d represents the truncated version of the document after removing irrelevant words according to $tf * idf$.

The result of this step is a set of clusters. Each one of them is a collection; generally, two to ten news articles from multiple sources chronologically ordered.

The second phase consists of identifying sentences that are close to the centroid in each cluster. The summarizer uses two utilities to achieve that: cluster-based relative utility (CBRU) and cross-sentence informational subsumption (CSIS). The first one estimates the relevance of a sentence to the general topic of its clusters and the second one measures redundancy among sentences.

Finally, sentence selection relies on approximating the CBRU and CSIS. To rank the sentences, MEAD applies three parameters for each sentence S_i: a centroid value, a positional value, and a first-sentence overlap.

- The centroid value C_i is the sum of the centroid values of all the words in the sentence. For a sentence S_i the equation 3.26 shows how to calculate C_i:

$$C_i = \sum_{w \in S_i} C_{w,i} \tag{3.26}$$

  Where $C_{w,i}$is the centroid value of a word w in sentence S_i.

- The positional value P_i for a sentence S_i is calculated according to its position in the document with n sentences as follows (see equation 3.27):

$$P_i = \frac{n - i + 1}{n} * C_{max} \tag{3.27}$$

  Where C_{max} is the maximum centroid score in that document; which benefits leading sentences.

- The first-sentence overlap F_i computes the inner product between the word occurrence vector of a sentence i and the first sentence vector of the document.

The final score of each sentence is a combination of the three scores minus a redundancy penalty for each sentence that overlaps highly ranked sentences.

WEBINESSENCE [97] is a web-based summarizer for web pages similar to MEAD. This summarizer first collects different URLs from web pages and extracts news articles of the same event. Then, it clusters the sentences from different documents. A centroid algorithm is used to find the representative sentences, avoid redundancy, and generate the summary.

Latent semantic analysis (LSA) is an unsupervised method that attempts to analyze correlations between a set of documents and the terms they contain.

Gong and Liu [98] proposed a method using LSA to extract highly scored sentences for single and multi-document summarization in the news domain. They first built a term-sentence matrix (n x m matrix). Each row corresponds to a word from the input (n words), and each column corresponds to a sentence (m sentences). The entry a_{ij} of the matrix is the weight of the word i in the sentence j. These weights are the results of the $tf * idf$ calculation. If a sentence does not include a particular word, the weight of that word in the sentence is equal to zero. The next stage is the singular value decomposition (SVD). It is used on the matrix to transform it into three matrices. If A is the original matrix, then we can use the equation 3.28 to describe the decomposition:

$$A = U\Sigma V^T \tag{3.28}$$

The first matrix U is an (n x m) dimensional matrix that represents a term-topic matrix. The second matrix Σ is a diagonal matrix $(m * m)$ where each row i corresponds to the weight of a topic i. Finally, the matrix V^T is the topic sentence matrix. Moreover, the matrix $D = V^T$ represents how much a sentence describes a topic; thus, d_{ij} determines the weight of the topic i in sentence j. After that, based on the length of summary in terms of sentences, they retained the number of topics and chose one sentence per topic. The issue is that it is nearly impossible to preserve the overall topic information in just one sentence.

In [99], they also introduced an LSA-based method that achieved better performance than the previous work. They used almost the same algorithm, the only difference was the technique to calculate the sentence salience. To locate sentences, they defined the weight of the sentence as follows (see equation 3.29):

$$s_i = \sqrt{\sum_{j=1}^{n} d_{ij}^2} \tag{3.29}$$

Graph approach

In multi-document summarization, some documents are more important than others. To favor certain sentence documents over others Wan [100] proposed a graph-based

ranking algorithm by adding a sentence-to-document relationship into the ranking process. In addition to documents impact on sentences, the author explained that even sentences in the same document must not be treated evenly. The position of a sentence and its distance to the document's centroid are two factors that he thinks it should be included to score sentences.

LexRank [26] is a graph-based technique for automatic text summarization. This method uses cosine similarity to create a weighted graph where nodes with a weight less than the specified threshold are removed. The main idea is that if a sentence is very similar to many others, it is likely to be very significant.

In TextRank [101], the authors created an undirected graph from the input text, where each sentence represents a node, and the arc between two nodes is weighted by their similarity. They compute the similarity between sentences and store them in a matrix called a similarity matrix, the similarity between two sentences is calculated as the number of words they have in common (content overlap). Then they convert it into a graph where sentences are vertices and the scores are edges. Finally, they extract the top-ranked sentences to constitute the summary.

Wan [102] introduced a method called TimedTextRank. TimedTextedRank is a sophisticated version of the TextRank method that includes that temporal information.

3.3.2 Compression approach

MASC [103] is a system that does sentence compression before selecting the sentence. MASC performs single as well as multi-document summarization. It consists of three steps: filtering, compression, and candidate selection. First, a filterer selects sentences of high relevance and centrality for further processing. For multi-document summarization, they consider the first five sentences in a document as the most relevant ones, hence these are not filtered. Then, multiple alternative compressed versions of the source sentences are generated. The final step is the selection of candidates. The selection uses a linear combination of static and dynamic features to pick the highest scoring candidate for the summary. Static features include the position of the original sentence in the document, length, compression-specific features, and relevance scores. These are calculated before the candidate selection process and do not change. Dynamic features include redundancy according to the current summary state and the number of candidates already in the summary from a candidate's source document. A particularity of this system is that it generates multiple compressed versions of the source sentences, and the selection process decides which one to use.

Machine learning approach

Topic modeling is very similar to document clustering, only instead of each document belonging to a single cluster or topic, a document can belong to many clusters. Bayesian topic models are probabilistic models that cover the topic of documents. Consequently, they improve the topic and document representation. They are quite powerful for multi-document summarization since they make a distinction between documents as opposed to most automatic summarization methods that regard them as one giant document [27]. Celikyilmaz and Hakkani-Tür [104] introduced a two-tiered topic model consisting of two levels: the first level concerns distributions over words and is called the low level, the second level concerns correlations between the lower-level topics given sentences. One of its limitations is that it doesn't differentiate general words from specific ones. The authors' motivation was to have general frequent words to have more chances to be included in the summary. Consequently, they presented an enriched two-tiered model that has a generic process to sample words from high-level topics leading to three-word distributions. The first one is a low-level topic and contains corpus specific words. The second one is a high-level topic that includes corpus general words. Finally, background word distributions which are document specific. This model is capable of capturing focused sentences with general words related to the main ideas of the document and has much less redundant sentences containing concepts specific to the user's query compared to the first model.
Some works used reinforcement learning for multi-document summarization. For instance, Ryang and Abekawa [105] regarded the extractive approach as a search problem. It constructs an extractive summary through a process of reinforcement learning. Their system uses $TD(\lambda)$ to learn and then execute a policy for summarizing a cluster of documents. A state corresponds to a candidate summary, and an action refers to an insertion of a textual element. After each action execution, the system receives either a reward, which is the score of the current summary according to the scoring function or a penalty if the summary length exceeds the maximum allowed length. The scoring function compromises between the relevance and non-redundancy of the sentences and the reward function is a delayed reward based on $tf * idf$ values.

Rioux et al. [106] extended the method of Ryang and Abekawa [105] for single-document summarization by using an improved version of TD called SARSA which in addition to modeling state space, models actions space too. They used ROUGE as part of their reward function and used bi-grams instead of $tf * idf$ as features. Their reward function is immediate at every action to help the learner get immediate feedback. The difference between immediate rewards and delayed rewards is that the learner receives instant feedback at every action in the first one and feedback only at the end in the second one.

3.4 Comparaison between machine learning works

This section summarizes some of the previously mentioned works regarding the machine learning approach. This is illustrated in table 3.2; where the lines represent the works, and the columns represent whether it is for single or multi-document summarization, the way machine learning is used, as well as the advantages and disadvantages of each work, and the methods used by each work. We did not consider the other approaches and instead focused only on the machine learning approach since our work is oriented toward this approach.

work	Single / Multi doc	Extractive/ Abstractive	Machine learning usage	Benefits	Disadvantages
[74]	Single document	Extractive	Feature combination	Features are tuned according to the genre (more flexible). Can execute the adequate model for scoring.	Dependent on the corpus genre.
[75]	Single document	Extractive	Feature combination with a supervised learning classifier	Combine four features (surface,content, event, relevance) to get feature vectors for sentences.	Requires large labeled data. Sensitive to summary length: goes low when there is few sentences chosen.
[76]	Single document	Extractive	Approached as a statistical classification problem with supervised learning	Provides a method for selecting among potential features and chooses a weighted combination. Does not train features using a corpus.	Robustness: many text genres do not contain any of the indicator phrases that are common in the used corpus. As the number of sentences grows more dispersed informative material tends to be included. Features are rarely dependent.

[77]	Single document	Extractive	Decision function (semi-supervised learning)	Integrate together features that are ignoring useless features and exploit only useful ones. Well suited for sentence extraction and does not always need to process the entire document before assigning classification.	There is no direct way to control the size of the summary. Needs annotated material: costly to produce. Features which predict whether a sentence should be extracted tend to be specific and occur infrequently.
[104]	Multi document	Extractive	Decision function (unsupervised learning)	Discover typicallu coherent sentences that contain key shared information with little redundancy. Directly captures coherent topics in documents. Can identify salient sentences.	Complicated to allocate latent topics in only two layers. Relying on expert summaries for indicating topic level, and restricting topic layers limits the practical applications of the model.
[105]	Multi document	Extractive	Decision function (reinforcement learning)	Compromises between relevance and non-redundancy of sentences.	Converges sub-optimally and excessively depends on the formulation of features and score function.
[106]	Multi document	Extractive	Decision function (reinforcement learning)	Converges fast to a stable solution.	Not enough to calculate similarity with the query to produce a good query focused summary. The query focused summary is worse than the top system from DUC 2006 conference.

[79]	Single document	Extractive	Decision function (reinforcement learning)	Uses embedding features instead of hand crafted ones.	Lack of training data to update embeddings.
[84]	Single document	Abstractive	Attention (supervised learning)	Can easily scale on a large amount of data. Can be trained on any document summary pair.	Processes only documents with a size of around 500 words. Generates very short summaries.
[85]	Single document	Abstractive	Attention (supervised learning)	Able to identify the key concepts and entities in the document.	Requires a lot of time processing. The model cannot capture the meaning of complex sentences.
[88]	Single document	Abstractive	Attention (supervised learning)	Can handle long sequences. Reduces amount of computation performed.	Fails to surpass seq2seq models.
[91]	Single document	Abstractive	Uses Transformers (supervised learning)	The cost of self-attention is smaller. Faster than traditional transformers.	Sequences from 1000 to 1500 token processing is slow.
[80]	Single document	Extractive and Abstractive	Uses Transformers (supervised learning)	Does a great job at identifying the most salient parts of documents. Provides well formed abstractive summaries and is simple to train.	Computationally costing.
[81]	Single document	Extractive	Uses Transformers (BERT) (supervised learning)	Captures document level features. Provides well formed extractive summaries and is simple to train.	Computationally costing. The input sequence length is only 1024 tokens

[92]	Single document	Abstractive	Uses Transformers (supervised learning)	Combines the advantages of recent architectures Able to capture the key information of the original document.	Computationally costing. The input sequence length (1024 token)

Table 3.2: Machine learning approach works.

3.4.1 Discussion

Among the works cited in the table, only [80], [84], [85], [88], [91], [92] concern abstractive summarization, and most of them deal with single-document summarization. The extractive models attempt to identify salient sentences and use machine learning either in the hope to combine features, decide what features to use, or as a classification problem where the model classifies sentences as "should be included in the summary" or "should not be included in the summary". On the other hand, the abstractive models use attention to prioritize information over others when processing a token. Overall, they can identify key information and handle long sequences; however, they are computationally costing. Besides, in terms of the type of machine learning, they use supervised learning, meaning that the data has to be labeled, which points to another issue: gather enough labeled data for training. Although this detail is not mentioned in the table 3.2, it's evident that the training data is a huge factor to achieve good performance. Finally, since most recent works deal with abstractive summarization, we can conclude that researchers are finally starting to get familiar with this type of summarization.

3.5 Conclusion

In this chapter, we have presented the state of the art on automatic text summarization. First, we have presented a classification for the existing works. An automatic summarization system can be classified according to several criteria, and the choice of the class greatly influences the summarization method used. These methods are themselves part of one of the automatic summarization approaches. Indeed, different approaches have been proposed since Baxendale's first work that is based on the statistical approach.

Automatic text summarization continues to gain more importance due to a large amount of information available. In our case, we are interested in the machine learning approach since most of the recent works are based on it. The model we are interested in is Distilbart which is based on transformers. Transformers can easily scale on a large amount of data, handle long sequences, and deduce rules automatically [50]. Besides, in abstractive summarization, the goal is to generate summaries from scratch depending on the understanding of the documents. Since machine learning networks can acquire knowledge during training, and since transformer-based components for Distilbart represent language models that have a general understanding of the language, we think they have a great potential to reach our objectives. Also, we can exploit the full power of GPUs and perform parallel computations. Distilbart deals with single-document summarization, hence we will attempt to adapt it for multi-document summarization.

In the following chapter, we will introduce the deep neural network concepts and the most used architectures to better understand our solution that will be discussed in the third chapter.

Chapter Four

Design of our system Quiksum

4.1 Introduction

In the previous chapters, we have seen the different works on automatic summarization with an emphasis on deep learning, especially on models that use transformers [50]. We dedicate this chapter to cover all the aspects describing the purpose of our work. We will first start with the presentation of the problem that inspired the creation of our system which we named Quiksum, and describe the application; then, we will highlight the architectures of the models used and explain our share of contribution. Third, we will detail the different steps to generate an abstract using each model. Finally, before concluding by wrapping up what we have achieved; we will discuss how to generate the summaries after the fine-tuning which can be done in two ways: a simple preprocessing, or a preprocessing that eliminates redundant sentences.

4.2 Problematic

Nowadays, people use the Internet to find information through information retrieval tools such as Google, Yahoo, Bing, etc. Due to the exponentially increasing rate of data, people need meaningful information in a reasonable amount of time. Thus, users can't read every document to find the one that is useful, hence the need to condense or summarize this information. Among all modern technologies, multi-document text summarization is one of the recent research topics that researchers in Natural Language Processing have been interested in.

However, to create a summary from one or multiple texts there is no efficient, fixed model yet that provides satisfactory results. Summarizing a text in such a way as to keep only the key points of the original document is a difficult task; if on top of that, the system has to be able to reformulate this information instead of copying the sentences that contain it, the difficulty grows a step further. Furthermore, the summary must be readable and coherent. What is also interesting is to be able to summarize several documents dealing with the same subject while avoiding redundancy, as it is clear that

some information will be repeated in at least two documents.

The work carried out in the context of automatic text summarization has shown very encouraging results, but there is always room for improvement. Nonetheless, most of this work favors an extractive approach. Very few works are interested in the abstract multi-documentary summarization.

4.3 Quiksum overview

4.3.1 Functionalities

A part of the focus of this project was to build an application that enables us to implement our models in a concrete example.

Firstly, a user can summarize one or multiple documents, either by entering text(s) or by uploading the document(s). Secondly, before generating the summary, it is possible to include a reference summary to evaluate the result, again either by inserting the summary or by uploading it. Finally, for a multi-document abstract, the texts can be subjected to either a simple preprocessing or a more complex preprocessing that tries to eliminate any redundancy. Each preprocessing type will be further explained in the rest of the chapter.

Our application offers different (extractive/abstractive) models that can be used. We can imagine a scenario where the user wants to summarize a collection of documents obtained from a search about a specific subject in a text format; in this case, he can easily copy-paste it to include the content. In case the result is a file, he can upload it directly instead.

We include the summary evaluation so that if the user has already a summary of one of the results, he can choose the model that performs best on this particular topic. As this situation is very unlikely to happen, this functionality is purely optional.

Other minor but certainly useful details concern the possibility to access all the information before and after the generation of the summaries; this includes the number of documents (or text) inserted, the name of the chosen template, the type of preprocessing, the length of each document (or text) and the summaries as well as their content. Besides:

- Changing the model is optional. By default, we use the model that performs best on our test dataset for abstractive summarization.
- Each time the user inserts a text or uploads a document its content is displayed including the content length.
- Each time the user inserts a text or uploads a document its content is displayed including the content length.

- Finally, after generating the summary, our system can evaluate the resulted abstract if the evaluation was activated, hence it will either display the summary and the evaluation result including its length, or the summary alone.
- For multi-document summarization, the only difference is that there is also an option to change the simple preprocessing to a more complex one. Hence during generated summary will also be different as the input changed.

4.3.2 Components

Our system is a web application divided into two parts: the frontend, and the backend as a server-client architecture; where the backend does all the calculations needed and generates the results; as for the front end, it is the UI component for the users to manipulate inputs and request results. The request is sent with fetch API and contains the following arguments:

- Documents: represents all the documents inserted.
- Evaluation: whether the user wants to evaluate or not.
- Reference summary: the content of the reference summary if the evaluation is activated. In the other case, it will have an empty value in the request.
- The model name: which user will use to employ the correct model to generate the summary.

The response returns the content of the generated summary and ROUGE scores if the evaluation was activated. Figure 4.1 illustrates our system.

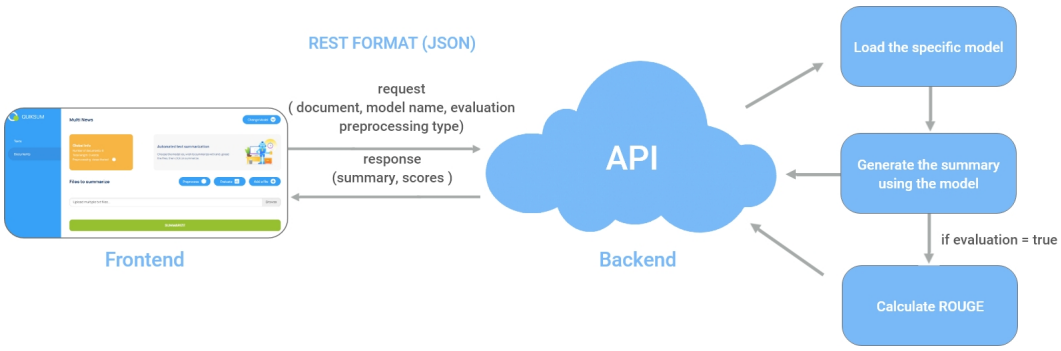

Figure 4.1: Representative schema of the architecture of our system.

4.4 Models architectures for summarization

4.4.1 Extractive summarization

In our case, we make use of the fine-tuned BERT large component of BERTSUM [81] (see chapter 2) to extract sentences features then uses a clustering algorithm which is K-means here to group them and output the closest sentence to each centroid. Features extraction happens at the last layer in the [CLS] token, each [CLS] symbol collects features for the sentence preceding it and will serve as input to the clustering algorithm. Figure 4.2 illustrates better the architecture.

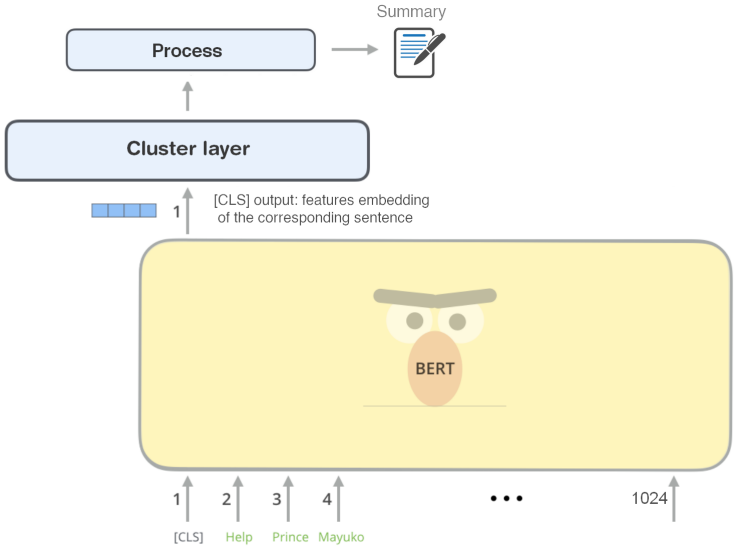

Figure 4.2: Represents the architecture of the extractive model consisting of two layers: BERT large and a clustering layer.

As we can see, it is a very simple architecture. For the process part we will provide more details in section 4.5.

4.4.2 Abstractive summarization

We chose Distilbart [92] which is a light version of BART [93] consisting of a BERT [51] like encoder and a GPT-2 [52] like decoder. BERT works well for tasks where the prediction at a certain position is allowed to use information from later positions; however, it performs poorly on generation tasks. On the other side, GPT-2 [52] is more effective for generation tasks but less effective on downstream tasks where the whole input yields information for the output [107]. The main reason that we chose BART is that it takes the best of both previous models. In total, BART large contains 406M parameters, as a consequence pretraining it would take a considerable amount of time

and requires a lot of memory. To compromise, we chose a compressed model called DistilBart [92] that has 100M fewer parameters.

Instead of finetuning the resulting model from scratch, we decided to finetune a model already capable of generating reasonable summaries, as we thought that it would give even greater results. The author finetuned Distilbart only on CNN / Daily Mail containing generated headlines of news articles which tends more to extractive summarization. Hence, We tried to finetune it on more relevant abstractive datasets.

On a high level, the process is similar to finetuning a transformer encoder-decoder: the encoder takes the input sequence, creates a representation of it, the decoder then receives the encoder representation, generates the output sequence autoregressively using teacher forcing. On a deeper level, the author ran the model on two epochs with a batch size of 32 and a half-precision, the learning rate was equal to 3e-5, the dropout value was equal to 0.1, and they set the maximum length of the generated summary to 142.

In our case, we tried with two learning rate values (2e-5 and 3e-5) and chose 3e-5 just to reduce the execution time. Also, similarly to the previous step, we used half-precision and a dropout value equal to 0.1. For the activation function we settled for GELU, and the cost function is cross-entropy. During generation, we set the beam size as 4, remove duplicated trigrams, and tuned the model with a maximum sequence length of 142 and a minimum sequence length of 56, and a length penalty equal to 2.0. The batch size is set according to how much our resources allow us to set, it was either 1,2, or 16. The figure 4.3 shows the script we execute for fine-tuning our models.

```
!python /content/transformers/examples/seq2seq/finetune.py \
    --learning_rate=3e-5 \
    --fp16 \
    --gpus 1 \
    --do_train \
    --do_predict \
    --n_val 1000 \
    --val_check_interval 0.1 \
    --sortish_sampler \
    --data_dir '/content/dataset' \
    --train_batch_size=2\
    --eval_batch_size=2\
    --output_dir=distilbart_multi_news \
    --num_train_epochs 1 \
    --model_name_or_path /content/model/best_tfmr
```

Figure 4.3: Models fine-tuning script

Most of the hyperparameters are set in other files. To finetune each model we store the datasets in Google Storage as well as the initial model before finetuning. Then we load the model and the dataset in our execution instance and run the script shown in figure 3. The data directory is where the dataset is loaded, the documents of the

dataset are grouped into three sets: train set, validation set, and test set. Each one of them is associated with a summary. For each dataset:

- We retrieve the articles one by one and their abstracts.
- We eliminate unnecessary spaces and line breaks so that each text is in a single line.
- Finally, we write these data in six files, three source-format for the texts (train, val, test), and three target-format files (train, val, test) for their references.

The output directory is the result of finetuning a part of the dataset, and the model name or path is the path to the current model. The output directory is then stored once again in Google Storage and is used in the same way, hence we iterate the process until the model is finetuned on the entire dataset.

4.5 Process of generating summaries

In this section, we will describe how we generate our abstractive or extractive summaries for one or multiple documents.

4.5.1 Single document summarization

Extractive summarization

Figure 4.4 is a simplified schema of our process for generating a single-document extractive summary.

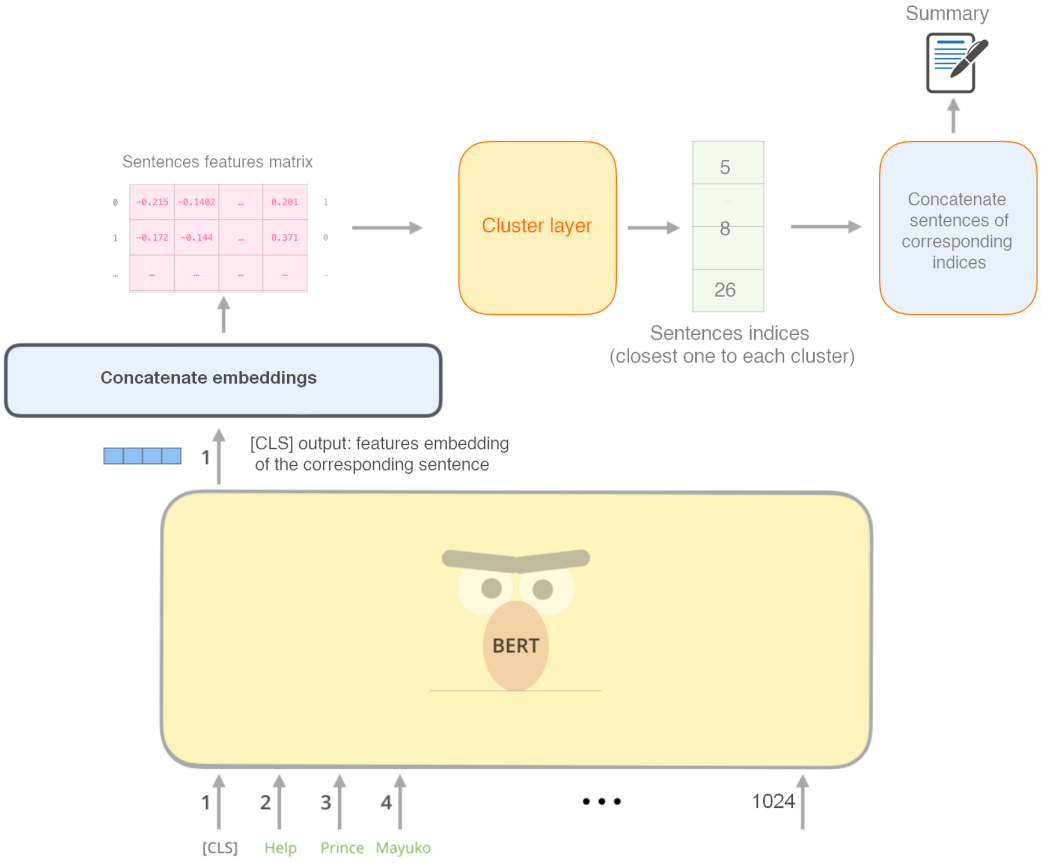

Figure 4.4: Illustrates the process for generating an extractive summary.

On a deeper level, to generate an extractive summary we:

- Take the text and tokenize it using BERT vocabulary; then we add the token embeddings representing the vocabulary id for each token, we also add the attention masks to distinguish paddings from tokens (1 for masked ones and 0 for non-masked), add the positional embeddings and sentence embeddings then feed it to BERT.
- Fetch the output of the CLS tokens at the last layer; this output represents the features of the sentence.
- Use those features to create a matrix, where the rows are the sentences, and the columns are the features of the corresponding sentence.
- Initiate the K-means model, such as it takes as input the features matrix and the number of clusters.
- Execute K-means and compute the centroid of each cluster until the clusters do not change.

- Search for the closest argument (sentence features) to the centroid then return for each cluster the index of the closest sentence to the centroid.
- Finally, extract the sentences based on these indices and concatenate them to get a summary of sentences.

Figure 4.5 represents the algorithm organigram which summarizes the steps for generating an extractive summary.

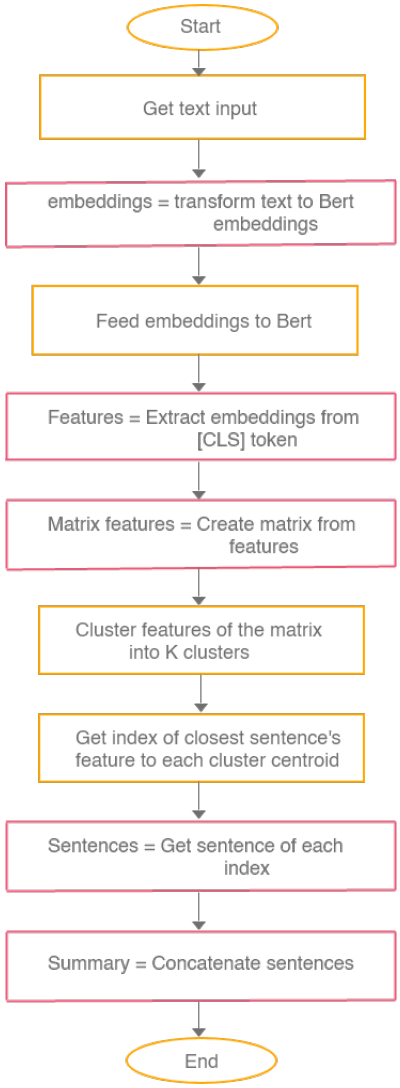

Figure 4.5: Shows and organigram explaining the generation of an extractive summary.

Abstractive summarization

To summarize one document, the process is very straightforward:

- We take the text and tokenize it.

- After breaking the text into tokens we convert the tokens from a list of strings to a list of vocabulary indices, such as each token has its id (the vocabulary include 50265 token).
- We also add the attention masks to avoid performing attention on padding token indices, we add a value of 1 for non masked tokens and 0 for masked ones.
- Then, we feed the input id and attention masks to the encoder to generate the representation of the document.
- The decoder then will use the encoder output to generate tokens that will constitute the summary auto-regressively:
  - The decoder takes the output of the encoder and the starting token to generate the first token.
  - When the last decoder produces its output, the model multiplies the resulting vector by an embedding matrix, which will project the vector to the size of the vocabulary; and each number is the score of a token of the vocabulary.
  - Beam search is applied to select the most appropriate token.
  - This token will be the next input to the decoder, we use then the output of the encoder again until the end of sequence is reached or the maximum length of sequence is reached.

Figure 4.6 represents the algorithm organigram which summarizes the steps for generating an extractive summary.

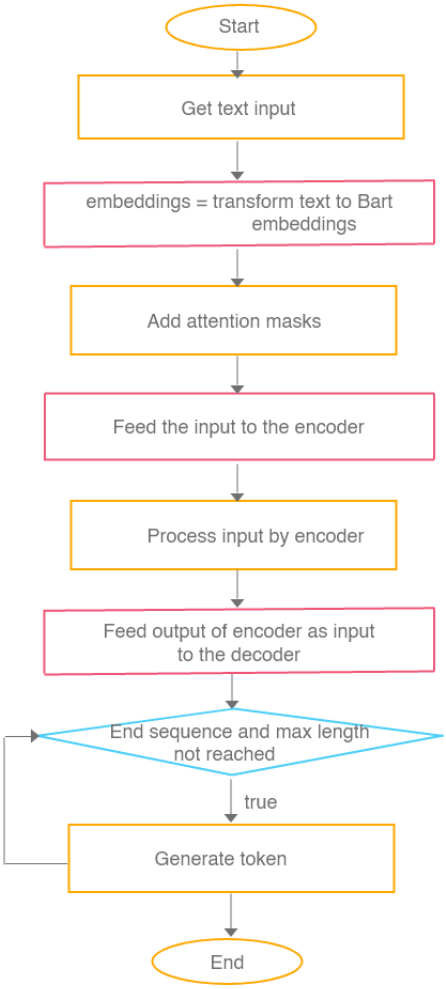

Figure 4.6: Shows and organigram explaining the generation of an abstractive summary.

4.5.2 Multi-document summarization

Whether it is abstractive summarization or extractive summarization, we preprocess our documents to get one global text and consider it as summarizing one document. Hence, we perform the same process described above. We applied one of two different preprocessing each time:

- **Simple preprocessing**
  The preprocessing is very basic: we take the texts and concatenate them together to get one single global text. The order of concatenation follows the order in which documents are inserted.

- **Similarity preprocessing**
  a- Bert for semantic similarity
  We used Bert to calculate the similarity between sentences. Two sentences with

different words and structure can convey the same meaning. Semantic similarity of sentences bases on the meaning of the words and their combination together. The easiest way to calculate the semantic similarity between a pair of sentences is by taking the average of the word embeddings and calculate the cosine between the resulting embeddings. Cosine similarity is a metric that performs an inner product between two vectors; if two vectors are similar then the angle between them would be equal to 0 degree, therefore the cosine value would be equal to 1. BERT can be used for a huge variety of tasks without task-specific architecture modifications. It can also be used for its features embedding, which in our case we extract the features embeddings from the [CLS] token.
To compute the semantic similarity between two sentences, we don't use basic word embeddings such as Word2Vec [108] or Glove [109] for multiple reasons:

- These basic result embeddings do not take word position into account as they are based on the bag-of-words method.
- These are context-free models that generate a single word embedding representation for each word in the vocabulary and do not capture polysemy.

In contrast, BERT has positional embeddings and is context-dependent, hence a word can have multiple embeddings.
An important note here is that BERT is not trained for semantic sentence similarity. Therefore, we cannot apply the cosine similarity directly on the BERT embeddings. In our work, we use a BERT base model finetuned on the AIINLI dataset, then on a train set of STS benchmark; these datasets are specifically suited for semantic textual similarity.
Now, to compute the similarity we:

- Feed the first sentence into BERT: we use the BERT tokenizer to convert the words into tokens. Then we add the [CLS] token at the beginning of a sentence, and the [SEP] token at the end. The tokenizer then replaces each token with its id from the embedding table. We also add the attention masks and positional embeddings.
- Extract the sentence embedding from the [CLS] token.
- Perform the same two previous steps for the second sentence.
- Finally, we compute the cosine value between those two embeddings: since two vectors can be equal (cosine = 1) only if the two sentences are comprised of the same words, we consider that two sentences are similar if the cosine value is equal or greater than a threshold value.

- **b- The preprocessing steps**

The main goal of this preprocessing is to eliminate redundant sentences. We suppose that, as we are going to summarize multiple documents there are high chances that we will have portions that has the same meaning. For example, we have s_1, s_2, s_3, s_4, and s_1, and s_3 have the same meaning, instead of summarizing s1, s2, s3, s4, we would abstract s_1, s_2, s_1, s_4, or s_3, s_2, s_3, s_4,.
To achieve that we:

- Take the texts and concatenate them to make them one long text and remove any unnecessary spaces (we should have only one space, not two consecutive spaces or more).
- Split the text into sentences:
  A sentence always begins with a capital letter and ends with a period, question mark, or exclamation mark. In some cases, however, a text can include abbreviations or names such as John D. Rockefeller, and it obviously cannot end in the D character. A simple solution is: we check the preceding two characters; if it's not a space then it means that there are more words after the D, we concatenate the sentence with the next one and perform the same process for each adjacent pair of sentences.
- We group similar sentences in clusters:
  Initially, each cluster contains one sentence; there are as many clusters as the number of sentences consisting of the text. For each sentence, we compare it with all other sentences; if they are similar (similarity score greater than a threshold), we add it to the cluster of that sentence.
- The previous step gives us a set of clusters; the number of clusters corresponds to the number of sentences in the text. This means that there are high chances that some clusters are redundant; however, this is not an issue, if we extract the same sentence from similar clusters and concatenate them, we would have exactly the desired input:
  At first, we take the first cluster and randomly choose a sentence inside it that will replace all the sentences of the cluster in the text, then we search for that phrase in the second cluster; if it is present then we assign to the second cluster the same sentence. We proceed in this way until all similar clusters would have the same chosen sentence.
- Finally, we concatenate the chosen sentences to get a coherent text that will be used to generate the summary of the initial documents.

Figure 4.7 represents the algorithm organigram which summarizes the steps to preprocess the text with sentence redundancy elimination.

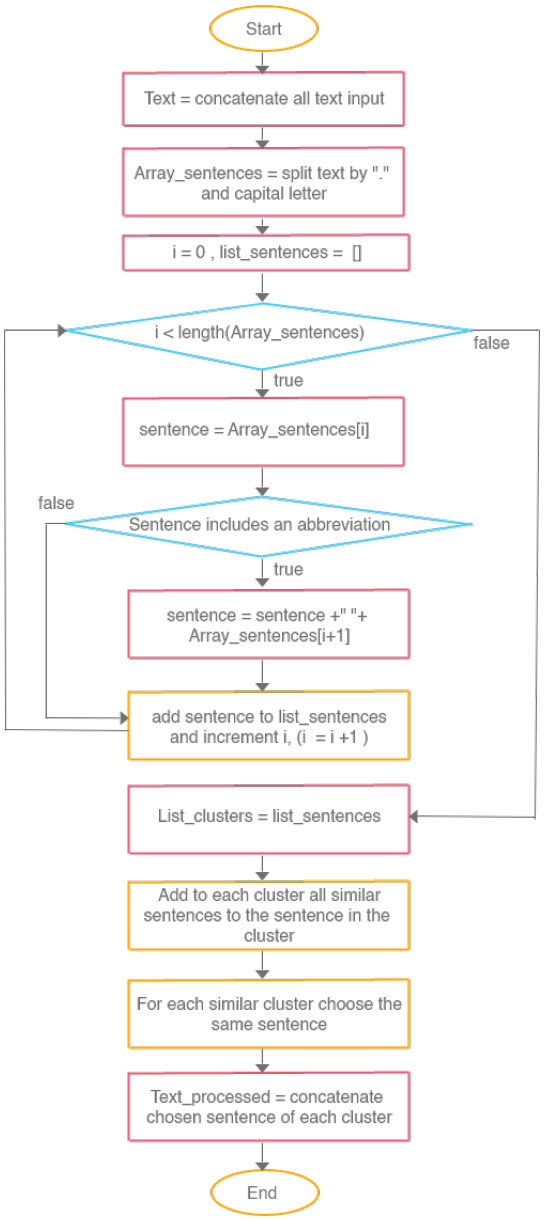

Figure 4.7: Organigram explaining the preprocessing for multi-document summarization.

4.6 Conclusion

In this chapter, we have covered everything about our contribution from the design point of view. We have described our application and presented our models for the abstractive and extractive type, as well as the steps for generating summaries using them. Finally, we explained the functioning of our proposed method that is used during preprocessing in the generation of multi-document summaries.

In the next and last chapter, we will present the technical details related to the implementation and the results obtained.

Chapter Five

Realization of Quiksum

5.1 Introduction

After presenting the architectures of our models as well as our application requirements in the previous chapter, this chapter discusses the implementation of the system. The first section provides technical information about the system, including hardware and software decisions we took. There are several libraries we used, these are explained along with the programming languages used in section 5.3 and 5.4. In section 5.5 we present the datasets for training as well as testing. In section 5.6 we will present some screenshots of our implemented system and describe it. Section 5.7 provides an overview of the evaluation metric, and section 5.8 describes in detail the experiments we conducted as well as the results we obtained. Finally, in section 5.9 we discuss the results and conclude this chapter by recapitulating what we have done.

5.2 Environment

5.2.1 Google colaboratory

Our finetuning and testing have been done on Google Colaboratory that is a Jupyter notebook saved on Google Drive composed of cells where it is possible to write python code, Linux commands, texts, and images on it. It is an executable file connected to a cloud instance that runs on variable spec systems through time with a superior upgrade each day passes. Below are the specs we took from google Colaboratory in 15/08/2020:
CPU: Intel(R) Xeon(R) CPU @ 2.20GHz.
RAM: 16 GB.
Disk: 70 GB.
GPU: Tesla T4 16 GB.
Python version: 3.8.

Personal Environment For our demo application we used a Windows 10 machine

with:
CPU: Intel I5-4460 @3.2GHz.
RAM: 8 GB.
GPU: Nvidia GTX 1050TI 4 GB GDDR5.
Disk: SSD 240 GB.
Python version: 3.8.

5.3 Programming language and IDE

5.3.1 Python in the backend

Figure 5.1: Python programming langage logo.

We used Python in our Backend system mainly in manipulating the Transformers models (fine-tuning and the evaluation as well as the production) this choice was made due to how popular this language is in the machine learning domain. It is easy to use, has a clear and readable syntax. Many useful libraries that we used such as Keras, Pytorch, and TensorFlow are built with it. Python has a growing community with over 8 million developers in 2019 [110] which made looking for help and solutions for some issues a lot easier. Figure 5.2 represents Python programming language ranking and its most popularusage cases and least usage case.

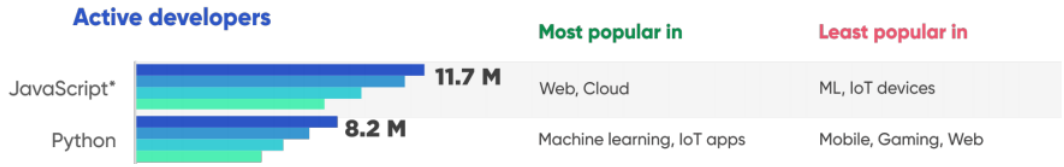

Figure 5.2: Python programming language ranking. [110]

As figure 5.2 shows, python is the second most popular programming language but ranks first when it comes to machine learning.

5.3.2 Javascript in the frontend

The main language used for our front end is Javascript due to its flexibility, the control over the UI elements, and the server communication using HTTP fetch as well as the popularity in web development, we used it to create the UI interface below. Javascript has also a growing community with over 11 million developers in 2019 [110].

5.3.3 Integrated Development Environment

An IDE, or Integrated Development Environment, is a software that enables programmers to write their code into a computer, it increases the productivity of writing code because of multiple services, offers syntax highlighting for more readable code, autocomplete for more fast typing. In our system we used Jupyter Notebook, and Visual Studio Code.

Jupyter Notebook

Jupyter Notebook is an open-source web-based IDE that allows creating documents containing code, images, narrative text heavily oriented to Machine learning, and data visualization because of its block system that allowed us to execute partial code and code snippets.

Figure 5.3: Jupyter Notebook IDE logo.

5.3.4 Visual Studio Code

We used Visual Studio Code IDE to build the demo application. It is free software provided by Microsoft for code writing famous by its flexibility of adding open-source plugins that help with the building of the application, like Live-Server plugin that will auto-reload the pages after each change saving us humongous amount of time.

Figure 5.4: Visual Studio Code IDE logo.

5.4 Used libraries and frameworks

5.4.1 Transformers by Hugging Face

We used the transformer repository provided by Hugging Face which is a library built with python. It uses Pytorch and TensorFlow 2.0 frameworks to provide the base architecture of the popular transformer models we were interested in described in the previous chapters (BERT, GPT, BART) as well as the orientation of that repository that tends to Natural language understanding (NLU) and Natural language generation (NLG) tasks which define the abstractive summarization. It contains a lot of features like its simplicity and entry-level code that makes researchers and educators execute and evaluate custom models faster, helper functions that allowed us to easily save checkpoints, and results while fine-tuning to make it easier for researchers to share their detailed results for more comparisons.

5.4.2 Pytorch

PyTorch is an open-source Machine learning framework built using python and the torch library. PyTorch uses tensors which are multidimensional arrays designed for employing the GPU power for computational operations like matrix multiplication.

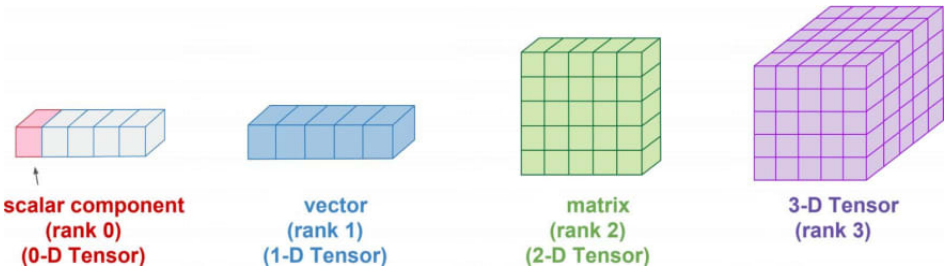

Figure 5.5: Tensors illustrated [111].

5.4.3 FastAPI

As we mentioned before, our demo application is an API with a frontend application as UI. The API is a python application built using a modern framework called FastAPI; light and very fast simple framework. We chose this framework primarily because it is built with Python which made the interaction with our model functions easier, acknowledged by major development companies, which gave us the trust in its capabilities. The implementation of the Web Service was very simple via HTTP Methods GET and POST routing system as well as JSON being the input and output of the backend.

5.5 Datasets

For fine-tuning, we used multiple datasets on the same architecture (DistilBart) separately to obtain multiple models, then evaluate each model on the same dataset to compare how each model will perform based on the dataset it was finetuned on.

5.5.1 CNN/DailyMail

They represent two datasets created by Hermann et al. [112] containing over 300,000 articles in total (93,000 for CNN and 220,000 from Dailymail newspaper). This dataset has each article paired with a short set of summarized bullets points that represent meaningful highlights of the piece. Journalists are typically trained to communicate the big ideas of an article in the first few sentences as bullets, a strategy which this dataset also adopted. We used the non-anonymized version provided by Tensorflow organized as two features:

- Article: which we considered as the text input for our model.
- Highlight: a joined text of news headlines and highlights that we considered a reference summary.

5.5.2 AESLC

Created by Zhang and Tetreault, AESLC [113] consists of 18,000 email bodies and their subjects from the Enron Corpus, a collection of email messages of employees in the Enron corporation, designed for abstractive email summarization and available for public usage.

The original Enron dataset contains 517,401 email messages from 150 user mailboxes, AESLC was obtained after a series of email body cleaning, email filtering, and email de-duplication.

The email subjects are typically much shorter than summaries generated from other datasets. Although our datasets are mainly news related, we were interested to see how it would perform on our model, if the model would learn more abstraction. We used the dataset provided by Tensorflow with two features.

- Email body: considered as an input document to the model.
- Subject line: considered a target summary for the model.

5.5.3 Multi-news

A large scale dataset created by Fabbri et al. [114] containing around 56,000 articles collected from newser.com with their summaries, and each summary is professionally

written. The difference between Multi-news and other popular news datasets like CNN/DailyMail is that the articles come from diverse sources: more than 1,500 sites appearing as source documents five times or greater. Furthermore, a total of twenty editors contributed to 85% of the total summaries on newser.com, and the thing about those summaries is that they are notably longer with around 260 words as an average. Fabbri et al. claim that their dataset tests its ability in the abstractive summarization models where the generated text must be fluent and coherent. We used the dataset provided by Tensorflow with two features:

- Document: which we considered as the text input for our model.
- Summary: considered a target summary for the model.

5.5.4 Gigaword

Gigaword fifth edition (v5) is a very large dataset created by Parker et al. [115]. It contains 10 million documents with over 4 billion words collected from newswire in several years by the Linguistic Data Consortiume (LDC).

We used the Annotated version of Gigaword created by Courtney Napoles, Matthew R. Gormley, Benjamin Van Durme in LDC [116], it adds automatically-generated syntactic and discourse structure annotation to English Gigaword Fifth Edition.

The goal of the annotation is to provide a standardized corpus for knowledge extraction and distributional semantics which enables broader involvement in large-scale knowledge-acquisition efforts by researchers. The annotation layers tokenize and segment sentences, it names entities as well as provides lemmas for words and also describes some words like "Algiers" is a "location". We used a sample of the dataset provided by Tensorflow containing 4 million articles organized as two features:

- Document: which is the article and we considered it as input.
- Summary: which is a headline for the article and we considered it as reference summary.

Table 5.1 describes the characteristics of each dataset.

Dataset	#pairs (train/val/test)	# words (doc)	#words (summary)
Multi-News	44970/5622/5622	1781	216
CNN/DailyMail	287113/13368/11490	683	51
AESLC	14436/1960/1960	119	4
Gigaword	3803957/189651/1951	30	8

Table 5.1: The dataset splits and the average number of words in each document and reference summary.

5.5.5 DUC Document Understanding Conferences 2004

To further progress in summarization and enable researchers to participate in large-scale experiments, the National Institute of Standards and Technology began a new evaluation series in the area of text summarization, which they called the Document Understanding Conference (DUC) [117]. The basic design for the evaluation followed ideas in a recent summarization road map that was created by a committee of researchers in summarization. Aiming to create reference data consisting of documents with their respective summarie for training and testing. The training data were distributed in early March of 2001. The test data were distributed in mid-June, and results were due for evaluation on the first of July 2001. Then they discussed these results during the same year in September.

The DUC became an annual event that every organization with interest in text summarization can participate in a series of coordinated experiments. The results are presented in the annual workshop where a comparison is made to preserve the competitive nature of the conference. The NIST staff developed a series of guidelines that protect the value of the conference as well as the participants.

The most important evaluation measure used in DUC 2001 to 2004 is coverage, which means how well the system performed the content selection with other text quality like readability and structure. The coverage of the submitted summaries was manually evaluated by humans by comparing the system summary with a human-written one. The system summary is split into clauses for each clause the system summary answers the question "To what extent is the information conveyed in the model clause expressed in the peer summary?" In 2001 they scored the summary on a five-point scale where 1 means "The model clause is not covered at all by the peer" and 5 means "The meaning of the model clause is fully expressed in the peer summary". In later years they added another scale, making it six points, ranging from 0 to 1 with 0.20 increments. A choice X was interpreted to mean that X percent of the content of the model clause is covered.

The strict laws of the conference and the evaluation system gave the data credibility to be used in summarizer systems. We used the DUC 2004 to test our models and compare results, it contained 30 clusters, each cluster represented a topic and had 10 documents, and each cluster had an averaging of 3 to 4 reference summaries.

As we mentioned before, the distilBart base model is finetuned on a different dataset each time to give us a new model ready to be tested. As each model performs best on the test split of the dataset it was finetuned on, evaluating our model on one of these datasets would give unfaithful results overall because it will favor the test split that it was finetuned on. To have objective results, it is necessary to test each model on a single dataset that it was not fine-tuned on and we chose DUC 2004.

5.6 Quiksum preview

Figure 5.6: Screenshot of the web application.

Quiksum offers a very refined UI that's very friendly and simple to use. Figure 5.6 provides a preview of the documents summarization page, and we will explain each element:

- 1: The menu enables us to change the model to summarize with. By default, the summaries are generated using the model fine-tuned on Multi-news; in fact, we do not load the model directly when the user changes the model, instead we load it after clicking on the summarization button since he may change it multiple times before actually starting the summarization process.
- 2: Model name - indicates the name of the model chosen.
- 3: By default, there's always this area for single-document summarization. It enables us to upload a document.
- 4: In the case of multi-document summarization, this button is used to upload other files to summarize.
- 5: menu to change the page, either to documents summarization or to text summarization.

- 6: Each time a file is uploaded, its content is displayed by clicking on the arrow in this content area, it can also be hidden again by clicking on that same arrow.
- 7: The orange button activates or inactivates the evaluation. When it's activated an area appears to upload a reference summary, otherwise, it's hidden.
- 8: This element will contain the generated summary.
- 9: After evaluation, scores will be displayed there.
- 10: Summarization button, used to start generating the summary.
- 11: This area contains the information relative to summarization such as the number of documents to summarize, the evaluation state (activated or not), the type of preprocessing (simple or the other one that changes when clicking on the button in the left of the orange one).

In the text page, the only difference is that instead of uploading, we offer to insert or type the texts to summarize. By default, there's only one text area but similarly to the document's page, it is possible to add other text areas. Also, the reference summary has to be inserted or typed instead of uploaded.

For user experience during generation we display a loading screen while the model is generating the summary as shown in figure 5.7.

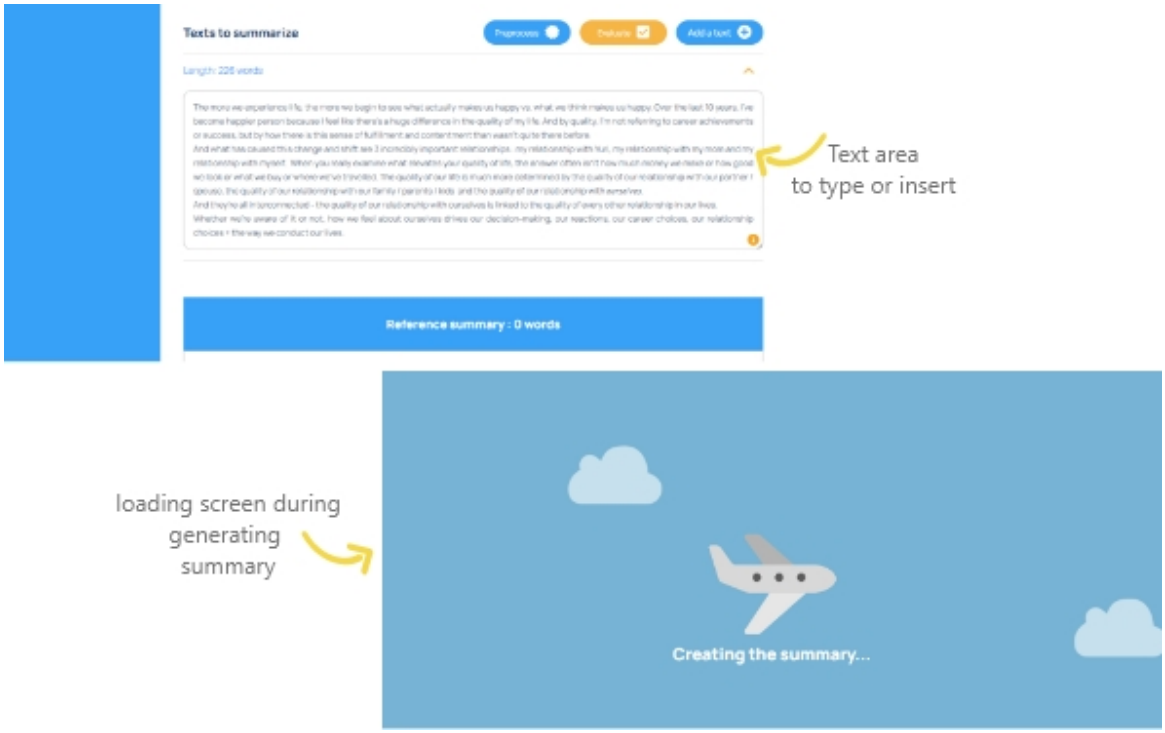

Figure 5.7: Screenshot of the text area and animation of the web application.

5.7 Evaluation metrics used

5.7.1 ROUGE

Recall-Oriented Understudy for Gisting Evaluation or ROUGE is an evaluation metric used essentially to evaluate summaries. It works by comparing the generated summaries against one or a set of other summaries usually human-written ones and is recall-oriented. In general, it compares overlapping words in n-grames between two texts; however, it is not enough to just compare overlapping words. That is why we calculate the precision and the F-measure as well. It was our choice for evaluating the summaries generated by our system; precisely, we used ROUGE-Ngram(1 and 2) as well as ROUGE-L.

Recall: The recall is the quantity of right information recovered by a system compared to what it should recover. It refers to the ratio of text units common to the generated and reference summary over the number of all text units in the reference summary . Recall in the context of ROUGE refers to how much the system summary was able to capture from the reference summary. If we are just considering the individual words (1-grams), it is computed as:

$$R = \frac{number\_of\_overlapping\_words}{total\_words\_in\_reference\_summary} \tag{5.1}$$

Precision: The precision is the quantity of right information collected by a system compared to what it has recovered. It measures how much of the system summary was supposed to be included. Precision is measured as:

$$P = \frac{number\_of\_overlapping\_words}{total\_words\_in\_system\_summary} \tag{5.2}$$

In an ideal situation, a summary would have high recall and high precision. Unfortunately, precision and recall are often in tension. Increasing precision would reduce recall and increasing recall would also decrease the precision. As a result, different metrics have been proposed that rely on both precision and recall.

F-Measure: The F-measure is a mix between the recall and the precision that balances both matters in one number. F-measure is calculated as:

$$F_\beta = \frac{(1+\beta^2)P * R}{\beta^2 P + R} \tag{5.3}$$

Where F_β is the F-score. If β is more than one, the recall is advantaged. If it is less than one, the precision is advantaged.

ROUGE-N:

ROUGE-N is a recall of n-grams between the candidate summary and a set of reference summaries. The value of ROUGE-N can be computed by equation 5.4:

$$ROUGE - N = \frac{\sum_{S \in Summ_{ref}} \sum_{N-gram \in S} Count_{match}(N - gram)}{\sum_{S \in Summ_{ref}} \sum_{N-gram \in S} Count(N - gram)} \tag{5.4}$$

Where N is the size of the N-gram,$Count_{match}(N - gram)$ is the number of N-grams in both the candidate and the reference summaries, and $Count(N - gram)$ is the number of N-grams in the reference summary. However, this metric does not take the order of words into account; that is, it does not influence the result.

ROUGE-L:

To overcome the weakness of the ROUGE-N, ROUGE-L employs the concept of the longest common subsequences (LCS). The motive is: the longer the LCS between two summary sentences, the more similar they are. To apply LCS, the sentences in the abstract represent a sequence of words. The problem with ROUGE-L is that it calculates only the main sequence; therefore, other alternative or shorter LCS will not be considered in the calculation of the score. Equation 5.5 describes how to compute ROUGE-L (recall and precision) between a reference abstract S and a candidate abstract C.

$$R_{LCS} = \frac{\sum_{i=1}^{u} LCS_U(s_i, C)}{m}, P_{LCS} = \frac{\sum_{i=1}^{u} LCS_U(s_i, C)}{n} \tag{5.5}$$

Where u is the number of sentences in the reference summary, $LCS_U(s_i, C)$ is the LCS score of the union of the longest sequences between the reference sentence and the sentences in the candidate summary. Finally, m is the size of the reference summary, and n is the size of the candidate summary.

5.8 Experiments

5.8.1 Extractive summarization

In our experiments, we used the model for multi-document summarization in two ways: We used it to generate a summary directly from the input which represents the concatenation of multiple documents. We also used it to generate a summary after preprocessing the input as described in chapter three where we exclude similar sentences and choose only one of them.
In both cases we experimented with different summary length:

- Ratio = 0.2: where the summary represents 20% of the input document. This is a very reasonable ratio considering that the documents are very long. This

parameter is used to compute the number of clusters, where the number of clusters = number of sentences * ratio.

- Clusters = 6: where the summary contains the six most relevant sentences of the input.
- Clusters = 4: where the summary contains the 4 most relevant sentences of the original input; the clusters are obtained using K-means (k = 4) and these sentences are the closest to their respective centroid.

The second way of generating the summaries relies heavily on the similarity value we use to decide if two sentences are similar, as this method replaces similar sentences with one random sentence of those sentences. The reason we settled to make it random and not fix the first or third sentence is that these sentences are most likely located in different documents:

Suppose we have two sentences s1, s2, one from document A, and the second from document B; if both are located at the beginning of a paragraph, this means that they are equally important. Hence, picking s1 or s2 would be the same.

The similarity metric as mentioned in chapter 3 is cosine similarity between the sentence embedding generated by the BERT model. The highest value, the cosine value would have is 1 (identical sentence), as this situation is nearly impossible we had to choose a threshold value by running a test using human written samples. We noticed that the minimum threshold for which they are indeed similar was equal to 0.8 or higher. In our case, we tried with 0.8 and 0.9. Figure 1.1 shows an example of two similar sentences and two different sentences with their given cosine value.

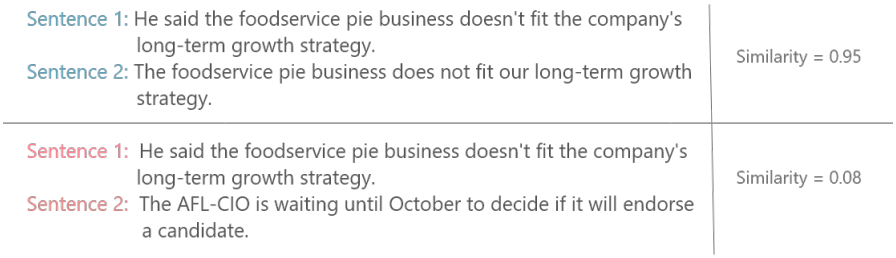

Figure 5.8: Represents examples of similarity values between sentences.

The table 5.2 shows the results of the extractive model with the same ratio (0.2) and two different preprocessing strategies. The average length of the documents is equal to 4964 words and the average length of the generated summaries is equal to 908 words.

Preprocessing		ROUGE 1	ROUGE 2	ROUGE L
Simple	Precision	75.17	23.19	40.23
	Recall	9.25	2.81	4.9
	F-measure	16.36	**4.98**	8.67
Similarity = 0.8	Precision	74.73	22.57	40.35
	Recall	9.29	2.74	4.95
	F-measure	**16.42**	4.85	**8.76**
Similarity = 0.9	Precision	74.92	23.00	41.07
	Recall	9.22	2.77	4.87
	F-measure	16.31	4.91	8.63

Table 5.2: Extractive model with ratio = 0.2 test set results on the DUC 2004 dataset using ROUGE.

As illustrated in the table 5.2, the best scores on a ratio = 0.2 are obtained when using the similarity preprocessing with a threshold = 0.8 when excluding ROUGE 2 which isn't that far. We notice that the precision in all rouge variations is the highest, meaning that the number of correct sentences included in both the generated and reference summary is higher than the number of sentences that should not be included in the generated summary. For instance, if we focus on rouge-1 scores 74% of the 1-grams in the generated summary are also in the reference summary. We also notice that the recall value is very low, this is due to the fact that the generated summary does not include all the sentences that should be included. Because the recall value is very low, the F-score gets affected and results in a low score as well. Table 5.3 shows the rouge scores obtained when the number of clusters = 6.

Preprocessing		ROUGE 1	ROUGE 2	ROUGE L
Simple	Precision	48.17	10.34	23.16
	Recall	25.07	5.41	12.04
	F-measure	32.69	7.04	15.70
Similarity = 0.8	Precision	47.42	10.37	22.16
	Recall	25.83	5.68	12.41
	F-measure	33.10	7.26	15.92
Similarity = 0.9	Precision	46.98	10.34	22.66
	Recall	26.06	5.70	12.56
	F-measure	**33.28**	**7.29**	**16.04**

Table 5.3: Extractive model with clusters = 6 test set results on the DUC 2004 dataset using ROUGE.

The average length of the generated summaries when the number of clusters = 6 is

equal to 190 words.

When the number of clusters = 6, the length of the generated summaries is much smaller compared to when using a ratio = 0.2; we noticed that the recall got three times higher, so the average of the n-grams captured in the generated summary and reference summary compared to the reference summary was higher. However since recall and precision go against each other, the precision scores got slightly lower. Though the balance between these two scores is much better than the first one, and the F-scores are also better. Besides the best scores are obtained when using a similarity preprocessing with a threshold = 0.9. Table 5.4 shows the rouge scores obtained when the number of clusters = 4.

Preprocessing		ROUGE 1	ROUGE 2	ROUGE L
Simple	Precision	42.15	8.84	20.20
	Recall	29.51	6.20	14.15
	F-measure	34.52	7.24	16.55
Similarity = 0.8	Precision	40.11	8.19	19.52
	Recall	29.80	6.17	14.60
	F-measure	33.82	6.95	16.52
Similarity = 0.9	Precision	41.73	9.11	20.04
	Recall	30.61	6.70	14.67
	F-measure	**35.16**	**7.67**	**16.83**

Table 5.4: Extractive model with clusters = 4 test set results on the DUC 2004 dataset using ROUGE.

Overall, the model gave the best results with these parameters: number clusters = 4, similarity = 0.9. A good summary is expected to have good recall and precision values when compared to a reference summary, as these are often in tension, we rank the results based on the F-measure. The gain in score was 1.88/ 0.33/ 0.79 for rouge1 / rouge2/ rouge L.

5.8.2 Abstractive summarization

Because our resources are limited, we used different batch sizes varying from 1, 2, and 16 depending on the dataset input length; we also split the training set and keep the validation and test set as it is:

- Multi-news: the maximum batch size we chose to finetune Multi-news is equal to 2. The training set is split into portions of 2000 training data each. We report one issue during finetuning Multi-news: as the GPU memory was not enough we could not complete the entire dataset and only manage to run the model on

half of it. It took us around 70 hours to finetune it on 20,000 data using Google colaboratory instance.

- AESLC: we finetune it on a batch size equal to 1 (which is also the maximum for this dataset with our GPU capacity) for 31 hours and split the training set by four portions, each time we use around 5000 data.
- Gigaword: we finetune our model on a batch size equal to 16 (also the maximum capacity) for around 160 hours and split the dataset to portions of 10,000 each.

Besides, since the model is first fine-tuned on CNN/DailyMail, we only run it for one epoch as it already takes a considerable number of hours (+100 hours knowing that we are limited to 10 hours maximum daily) to be complete. CNN/DailyMail and Multi-news datasets contain input documents longer than the maximum input length of a base model (length input $>$ 512 tokens). If we chose a model that accepts only 512 tokens, this would present a problem for position embeddings which would never be updated for longer input lengths. Hence, having a model that accepts 1024 tokens ensures it will be finetuned correctly.

Single-document summarization

Results of finetuning are presented in the table 5.5 which shows ROUGE scores on each dataset. The last row represents the baseline model before the additional finetuning.

	ROUGE 1	ROUGE 2	ROUGE L	Average summary length
Gigaword	21.23	8.13	18.25	61 words
AESLC	12.33	5.51	11.45	57 words
Multi-News	41.96	15.35	23.40	140 words
CNN/DailyMail	43.30	20.50	30.29	78 words

Table 5.5: Represents ROUGE-F1 scores and average length of generated summaries when fine-tuned on each dataset.

These results show that the baseline obtains the highest score, while the highest score between our models is attributed to the one finetuned with Multi-News with a small gap of 1.34/5.15/6.89 only for ROUGE 1, ROUGE 2, and ROUGE L respectively. For simplicity, we will attribute names to each model: DistilBart-CM for the model finetuned with CNN/DailyMail then with Multi-News, DistilBart-CG for the model finetuned with CNN/DailyMail then with Gigaword, and DistilBart-CA for the model finetuned with CNN/DailyMail then with AESLC.

We can see that DistilBart-CM performed best on its test set. The generated summaries have an average length of 140 words which is somewhat close to the average

length of the reference summaries (216 words). In comparison, DistilBart-CA generated summaries of around 57 words while the average reference summaries contain only 4 words; DistilBart-CG outputs are also far from the reference summaries length (61 words while in the test set an average of 9 words). We think that there is a possibility that these huge gaps of length resulted in lower scores compared to DistilBart-CM.

5.8.3 Multi-document summarization

Although the scores are lower for our models than the baseline on their test respective set, this does not mean the baseline is the most reliable for multi-document summarization. Therefore, to evaluate the models on multi-document summarization, we conduct a series of experiments using the DUC 2004 dataset:

- We feed the models the documents concatenated with each other, then observe the outputs and compare them with the reference summaries.
- We preprocess our input as in our extractive multi-document summarization approach, once with similarity threshold >= 0.8 and another one with threshold >= 0.9, and also compare them with the reference summaries.

Tables 5.6 shows the performance on DUC 2004 when doing a simple preprocessing.

	Rouge 1	Rouge 2	Rouge L
DistilBart-CG	7.11/23.04/10.83	0.66/2.13/1.01	5.07/16.44/7.72
DistilBart-CA	14.36/39.42/20.96	2.89/8.17/4.25	9.09/25.20/13.30
DistilBart-MC	**29.94/31.26/29.94**	**5.96/6.34/5.98**	**15.88/16.79/15.93**
CNN/DailyMail	18.20/39.41/24.75	3.83/8.38/5.22	10.45/22.73/14.24

Table 5.6: Results of our models on DUC 2004 compared with the basline with just concatenating the documents. These scores represent recall, precision and F-measure respectively. Best ROUGE numbers are bolded.

Our model (DistilBart-CM) outperform the baseline version with higher 5.19 /0.76/ 1.69 values considering F-measure in ROUGE 1/ROUGE 2/ROUGE L respectively. We analyse the quality of its generated summaries manually; overall we observe a high linguistic quality in terms of fluency and coherence, plus it was able to capture the topic of the input. However, sometimes the last sentence is incomplete, this is due to reaching the maximum sequence length. We can solve this issue by eliminating the last sentence but we prefer to keep the few information included in the last sentence instead.
Figure 5.9 shows an example of generated summary from DistilBart-CM compared to a reference summary.

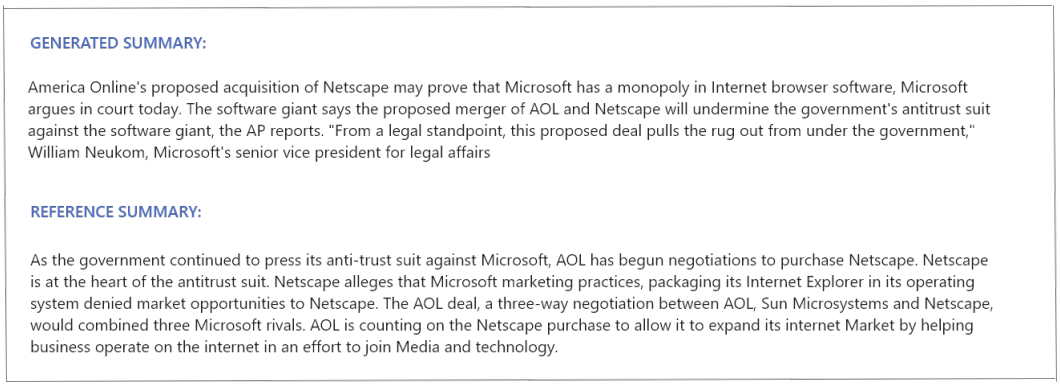

Figure 5.9: Generated summary from DistilBart-CM alongside the gold summary.

The worst results concern DistilBart-CG, sometimes the decoder predicted summaries with the correct subjects and sometimes not. The summaries were not well-formed syntactically and sometimes demonstrated a lack of semantic understanding of the input article. For example, it would display the word "georgia" in the summary even though the input was clearly not about georgia. After training on the entire dataset, we noticed a large amount of tokens in the summaries that made them completely unreadable. Investigation on the dataset revealed that the real dataset is not available for free usage, and the only dataset we used is one containing inadequate preprocessing. We think that it is nearly impossible to produce coherent summaries using the available Gigaword with DistilBart unless it is trained on the paid version. Figure 5.10 shows an example of a generated summary using DistilBart-CG along side with a reference summary.

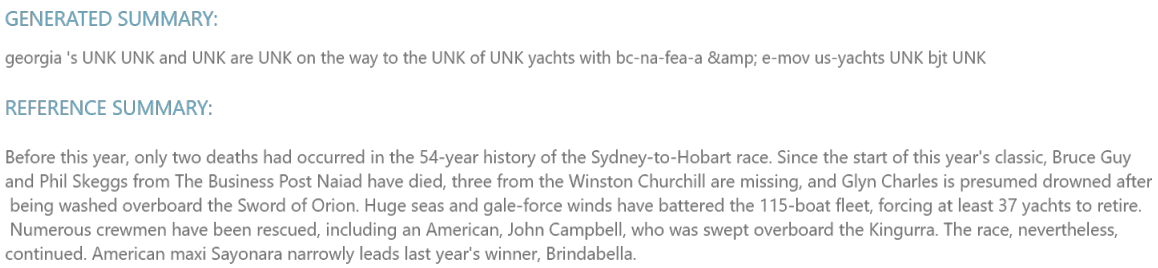

Figure 5.10: Represents a generated summary from DistilBart-CG alongside the gold summary.

Table 5.7 shows ROUGE scores when the input is preprocessed with our similarity algorithm using a threshold equal to 0.8.

	ROUGE 1	ROUGE 2	ROUGE L
DistilBart-CG	7.09/23.12/10.81	0.61/2.01/0.94	4.91/16.05/7.49
DistilBart-CA	15.13/39.80/21.87	3.18/8.50/4.62	9.66/25.57/13.99
DistilBart-CM	**30.77/ 30.66/30.27**	**6.10/5.97/5.96**	**16.30/16.44/16.11**
CNN/DailyMail	20.02/40.34/26.53	4.44/9.89/5.89	11.16/22.62/14.82

Table 5.7: Results of our models on DUC 2004 compared with the basline when preprocessed using our similarity algorithm and threshold = 0.8. These scores represent recall, precision and F-measure respectively. Best ROUGE numbers are bolded.

As we can see, all models besides DistilBart-CG have increased in scores compared to when doing just a concatenation. Besides, DistilBart-CG value did not decrease drastically.

Finally, the table 5.8 presents ROUGE scores when the input is preprocessed with our similarity algorithm using a threshold equal to 0.9.

	ROUGE 1	ROUGE 2	ROUGE L
DistilBart-CG	7.52/24.30/11.46	0.72/2.31/1.09	5.23/16.98/7.97
DistilBart-CA	18.30/44.47/25.87	4.26/10.45/6.04	10.90/26.60/15.43
DistilBart-CM	**32.69/35.85/33.48**	**6.96/7.75/7.71**	**16.94/18.86/17.45**
CNN/DailyMail	20.55/42.21/27.47	4.68/9.59/6.25	11.66/24.23/15.65

Table 5.8: Results of our models on DUC 2004 compared with the basline when preprocessed using our similarity algorithm and threshold = 0.9. These scores represent recall, precision and F-measure respectively.Best ROUGE numbers are bolded.

As we can see, there is a significant improvement compared to the result shown in table 5.6; the best scores are attributed to DistilBart-CM. We observed that recall and precision values are very close. In addition, it surprisingly exceeded the scores of the extractive model in ROUGE L and has very close values for ROUGE 1 and ROUGE 2.

5.9 Discussion

A part of the results of our approach was on the DUC dataset for multi-document summarization. Several experiments have been conducted, with and without application of sentence redundancy elimination. Experimentation producing highest scores exploits our proposed algorithm to preprocess the input with a threshold of 0.9. These results allow us to confirm that there is room for improvement; that is, since the preprocessing occurs before feeding the input to the model, there is a high chance that this method

would work for other models as well. The results we got also prove the power of such models to provide summaries with high linguistic quality; the same evaluation procedure has been applied to all our models. Our best abstractive system outperformed the baseline and had scores that can rival the scores obtained from the extractive model knowing that extractive models have higher scores than abstractive ones most of the time. Also, we observed that the choice of training dataset has a huge impact on the performance of a model. In most of the previous works, no matter whether it is extractive or abstractive summarization, they have used the ROUGE metric as an evaluation measure. Similarly, we used unigram and bigram overlap (ROUGE 1 and ROUGE 2) as a means of assessing informativeness, and the longest common subsequence (ROUGE L) as a means of assessing fluency. The major problem when using ROUGE in abstractive summarization is that, since the model generates its own sentences based on the context of the text documents, the output summaries contain a significant amount of paraphrasing and may use different words from reference summary even though they express similar meaning. And since ROUGE calculates the overlapping of the n-gram in the gold summary and generated summary, what if both summaries do not hold any n-gram in common? In our case, we had very good ROUGE 1 scores, but ROUGE 2 is even harder to use for abstractive summaries. This is probably why we obtained a kind of low value. We think that an appropriate metric for this task is necessary as those results do not fully describe the performance of our models.

5.10 Conclusion

This chapter described the technical side of our project including its software and hardware information. The actual results were presented with an in-depth analysis and explanation. We compared our models with a baseline model and were surprised that DistilBart-CM had very close scores to the extractive one. In the discussion section, we mentioned the need of an evaluation metric dedicated to abstractive summarization, since the ROUGE metric has many weaknesses and does not provide 100% reliable scores for this specific task. Despite that, we are positive that transformer-based models have great potential and can be further improved.

General Conclusion

In this work, our objective was to propose a solution to the problem of multi-document abstractive summarization. Automatic text summarization methods are essential as the amount of available data and the need of relevant, fast, and concise information is continuously increasing.

Our choice leaned towards the machine learning approach, which has proven its effectiveness in this area. Thus, we proposed to finetune an existing model called DistilBart to generate abstractive summaries. We experimented with different datasets: Gigaword, Multi-news, AESLC and we found that the best results were obtained when the model is finetuned on Multi-news. We also attempted to find a method that improves the quality of our generated summaries and proposed a solution based on replacing similar sentences with one that has the same meaning, and that is a part of their cluster, as our algorithm groups sentences into clusters. This solution was first used with an extractive model, then on our abstractive resulted models and we found that all of them achieved better performance when using it with a threshold equal to 0.9.
The main contributions of our work are:

- We fine-tuned an existing model on new datasets for single document summarization, then we adapted the resulting models for generating abstractive summaries of multiple documents.

- We proposed an algorithm to adapt our models to generate a summary for multiple documents for both the extractive and abstractive type.

The principal issue we faced during this project is related to the resources needed for the execution. Google Colaboratory resources are not unlimited, although it is free to use, the GPU provided is not powerful enough for big models to train fast. Also, an instance connection limit is around 10 hours then it disconnects; so when our execution was not complete and saved, we lost multiple times hours of progress. Besides, the GPU usage is also limited, after exceeding the limit one has to wait more than 8 hours to be able to use it again. To overcome this particular issue, we had to create multiple Google accounts then save the resulted model each time and execute it on another account. Finally, the biggest problem is the limit of RAM available since users can use up to 12GB of memory. As a consequence, we were not able to complete finetuning on Multi-News which is unfortunate since the DistilBart performed best when fine-tuned on this dataset.

To conclude, our best model (DistilBart-CM) gave very encouraging results, however, there is still room for improvement for it to be used in a real context. Besides, since the datasets are news articles it works only for this specific type of documents.

Also, the fixed maximum length of the generated summaries (142 tokens) makes that sometimes the last sentence is incomplete. Finally, the biggest weakness of our model is related to the maximum input length. DistilBart can process up to 1024 token as the input document. In the case of multi-document summarization, it is a huge problem. Processing longer sequences than that is still a topic of ongoing research. As a perspective, we would like in the future to:

- Complete the fine-tuning on Multi-News.
- Fine-tune the model on the paid version of Gigaword.
- Propose a post-processing method to clean the text by eliminating the last sentence if it returns no information or completes the sentence using a trained model that takes the summary and non summarized document as input.
- Try extractive summarization followed by abstractive summarization, where in the extractive step, we would like to choose the top k sentences without exceeding the model maximum length.
- Try to apply successive abstractive summarization either by splitting the document into shorter parts, summarize each part and concatenate the respective summaries with post-processing; or summarize a part of the document then concatenate the resulted summary with another part of the document, and so on until we would get the final summary. As expansive as it looks, we think it may give good results.

References

[1] Luis Gonçalves. *Automatic Text Summarization with Machine Learning — An overview*. Apr. 2020. URL: https://medium.com/luisfredgs/automatic-text-summarization-with-machine-learning-an-overview-68ded5717a25.

[2] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, et al. *Text Summarization Techniques: A Brief Survey*. 2017. eprint: arXiv:1707.02268.

[3] Roshna Chettri and Udit Kr. Chakraborty. "Automatic Text Summarization". In: 2017.

[4] Hamza Shabbir Moiyadi, Harsh Desai, Dhairya Pawar, et al. "NLP Based Text Summarization Using Semantic Analysis". In: *International Journal of Advanced Engineering, Management and Science* 2.10 (Oct. 2016).

[5] Eduard Hovy and Chin-Yew Lin. "Automated Text Summarization and the SUMMARIST System". In: *Proceedings of a Workshop on Held at Baltimore, Maryland: October 13-15, 1998*. TIPSTER '98. Baltimore, Maryland: Association for Computational Linguistics, 1998, pp. 197–214. DOI: 10.3115/1119089.1119121. URL: https://doi.org/10.3115/1119089.1119121.

[6] A. Sibsa K. Bouchelouche. "Vers un meilleur résumé automatique multi-documents". University Saad Dahlan Blida 1, 2018.

[7] Sindhu L Saranyamol C S. "A Survey on Automatic Text Summarization". In: *International Journal of Computer Science and Information Technologies* 5.6 (2014), pp. 7889–7893. ISSN: 0975-9646. DOI: 10.1.1.666.536. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.666.5367&rep=rep1&type=pdf.

[8] Kavita Ganesan, ChengXiang Zhai, and Jiawei Han. "Opinosis: A Graph Based Approach to Abstractive Summarization of Highly Redundant Opinions". In: vol. 2. Dec. 2010.

[9] Horacio Saggion. "Automatic Summarization: An Overview". In: *Revue française de linguistique appliquée* 8.1 (2008), pp. 63–81. DOI: 10.3917/rfla.131.0063. URL: https://www.cairn.info/revue-francaise-de-linguistique-appliquee-2008-1-page-63.htm.

[10] H. Van Lierde and Tommy W.S. Chow. "Query-oriented text summarization based on hypergraph transversals". In: *Information Processing and Management* 56.4 (2019), pp. 1317–1338. ISSN: 0306-4573. DOI: https://doi.org/10.1016/

j.ipm.2019.03.003. URL: http://www.sciencedirect.com/science/article/pii/S0306457318300372.

[11] Min-yen Kan, Kathleen McKeown, and Judith Klavans. "Domain-Specific Informative and Indicative Summarization for Information Retrieval". In: (Oct. 2001).

[12] Inderjeet Mani. "Summarization Evaluation: An Overview". In: *NTCIR*. 2001.

[13] Abdelkrime Aries, Djamel Eddine Zegour, and Walid-Khaled Hidouci. "Automatic text summarization: What has been done and what has to be done". In: *ArXiv* abs/1904.00688 (2019).

[14] Xiaojun Yuan, Ning Sa, Grace Begany, et al. "What Users Prefer and Why: A User Study on Effective Presentation Styles of Opinion Summarization". In: *Human-Computer Interaction – INTERACT 2015*. Ed. by Julio Abascal, Simone Barbosa, Mirko Fetter, et al. Cham: Springer International Publishing, 2015, pp. 249–264. ISBN: 978-3-319-22668-2.

[15] Samuel Pecar. "Towards Opinion Summarization of Customer Reviews". In: *Proceedings of ACL 2018, Student Research Workshop*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 1–8. DOI: 10.18653/v1/P18-3001. URL: https://www.aclweb.org/anthology/P18-3001.

[16] Ismini Lourentzou, Kabir Manghnani, and C. Zhai. "Adapting Sequence to Sequence models for Text Normalization in Social Media". In: *ArXiv* abs/1904.06100 (2019).

[17] R. Satapathy, C. Guerreiro, I. Chaturvedi, et al. "Phonetic-Based Microtext Normalization for Twitter Sentiment Analysis". In: *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. 2017, pp. 407–413.

[18] Vairaprakash Gurusamy and Subbu Kannan. "Preprocessing Techniques for Text Mining". In: Oct. 2014.

[19] D. Palmer. "Chapter 2 : Tokenisation and Sentence Segmentation". In: 2007.

[20] David D. Palmer and Marti A. Hearst. "Adaptive Multilingual Sentence Boundary Disambiguation". In: *Computational Linguistics* 23.2 (1997), pp. 241–267. URL: https://www.aclweb.org/anthology/J97-2002.

[21] Michael D. Riley. "Some Applications of Tree-Based Modelling to Speech and Language". In: *Proceedings of the Workshop on Speech and Natural Language*. HLT '89. Cape Cod, Massachusetts: Association for Computational Linguistics, 1989, pp. 339–352. ISBN: 1558601120. DOI: 10.3115/1075434.1075492. URL: https://doi.org/10.3115/1075434.1075492.

[22] Jeffrey C. Reynar and Adwait Ratnaparkhi. "A Maximum Entropy Approach to Identifying Sentence Boundaries". In: *Fifth Conference on Applied Natural Language Processing*. Washington, DC, USA: Association for Computational Linguistics, Mar. 1997, pp. 16–19. DOI: 10.3115/974557.974561. URL: https://www.aclweb.org/anthology/A97-1004.

[23] Hongyan Jing. "Sentence Reduction for Automatic Text Summarization". In: *Proceedings of the Sixth Conference on Applied Natural Language Processing*. ANLC '00. Seattle, Washington: Association for Computational Linguistics, 2000, pp. 310–315. DOI: 10.3115/974147.974190. URL: https://doi.org/10.3115/974147.974190.

[24] Adhika Pramita Widyassari, Supriadi Rustad, Guruh Fajar Shidik, et al. "Review of automatic text summarization techniques and methods". In: *Journal of King Saud University - Computer and Information Sciences* (2020). ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2020.05.006. URL: http://www.sciencedirect.com/science/article/pii/S1319157820303712.

[25] Victoria McCargar. "Statistical Approaches to Automatic Text Summarization". In: *Bulletin of the American Society for Information Science and Technology* 30.4 (2004), pp. 21–25. DOI: 10.1002/bult.319. eprint: https://asistdl.onlinelibrary.wiley.com/doi/pdf/10.1002/bult.319. URL: https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/bult.319.

[26] Günes Erkan and Dragomir R. Radev. "LexRank: Graph-Based Lexical Centrality as Salience in Text Summarization". In: *J. Artif. Int. Res.* 22.1 (Dec. 2004), pp. 457–479. ISSN: 1076-9757.

[27] Ani Nenkova and Kathleen McKeown. "Automatic Summarization". In: *Foundations and Trends in Information Retrieval* 5.2-3 (2011), pp. 103–233. ISSN: 1554-0669. DOI: 10.1561/1500000015. URL: http://dx.doi.org/10.1561/1500000015.

[28] Techieness. "Machine learning how involved in our day to day life ?" In: (July 2019). URL: https://www.techieness.org/post/machine-learning-how-involved-in-our-day-to-day-life.

[29] Constantin Orasan. "Pronominal anaphora resolution for text summarisation". In: 2007.

[30] Josef Steinberger and Karel Jezek. "Evaluation Measures for Text Summarization." In: *Computing and Informatics* 28 (Jan. 2009), pp. 251–275.

[31] Gerard Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN: 0201122278.

[32] Horacio Saggion, Dragomir Radev, Simone Teufel, et al. "Developing Infrastructure for the Evaluation of Single and Multi-document Summarization Systems in a Cross-lingual Environment". In: *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*. Las Palmas, Canary Islands - Spain: European Language Resources Association (ELRA), May 2002. URL: http://www.lrec-conf.org/proceedings/lrec2002/pdf/158.pdf.

[33] Dragomir R. Radev, Simone Teufel, Horacio Saggion, et al. "Evaluation Challenges in Large-Scale Document Summarization". In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. Sapporo,

Japan: Association for Computational Linguistics, July 2003, pp. 375–382. DOI: 10.3115/1075096.1075144. URL: https://www.aclweb.org/anthology/P03-1048.

[34] Ani Nenkova, Rebecca Passonneau, and Kathleen McKeown. "The Pyramid Method: Incorporating Human Content Selection Variation in Summarization Evaluation". In: *ACM Trans. Speech Lang. Process.* 4.2 (May 2007), 4–es. ISSN: 1550-4875. DOI: 10.1145/1233912.1233913. URL: https://doi.org/10.1145/1233912.1233913.

[35] 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://www.aclweb.org/anthology/W04-1013.

[36] Kishore Papineni, Salim Roukos, Todd Ward, et al. "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://www.aclweb.org/anthology/P02-1040.

[37] Inderjeet Mani, David House, Gary Klein, et al. "The TIPSTER SUMMAC Text Summarization Evaluation". In: *Ninth Conference of the European Chapter of the Association for Computational Linguistics*. Bergen, Norway: Association for Computational Linguistics, June 1999. URL: https://www.aclweb.org/anthology/E99-1011.

[38] Andrew H. Morris, George M. Kasper, and Dennis A. Adams. "The Effects and Limitations of Automated Text Condensing on Reading Comprehension Performance". In: *Information Systems Research* 3.1 (Mar. 1992), pp. 17–35. DOI: 10.1287/isre.3.1.17. URL: https://ideas.repec.org/a/inm/orisre/v3y1992i1p17-35.html.

[39] A. L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM J. Res. Dev.* 3.3 (July 1959), pp. 210–229. ISSN: 0018-8646. DOI: 10.1147/rd.33.0210. URL: https://doi.org/10.1147/rd.33.0210.

[40] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[41] J. A. Hartigan and M. A. Wong. "A k-means clustering algorithm". In: *JSTOR: Applied Statistics* 28.1 (1979), pp. 100–108.

[42] Glorot Xavier. "Apprentissage des reseaux de neurones profonds et applications en traitement automatique de la langue naturelle". Theses. Universite de Montreal, Nov. 2014. URL: http://hdl.handle.net/1866/11989.

[43] Facundo Bre, Juan Gimenez, and Víctor Fachinotti. "Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks". In: *Energy and Buildings* (Nov. 2017). DOI: 10.1016/j.enbuild.2017.11.045. URL:

https://www.researchgate.net/figure/Artificial-neural-network-architecture-ANN-i-h-1-h-2-h-n-o_fig1_321259051.

[44] Christoph Münker. "One perceptron with n inputs and one output. The circle describes the perceptron, in which is applied the linear combination and the activation function." MA thesis. URL: https://www.ke.tu-darmstadt.de/lehre/arbeiten/master/2016/Muenker_Christoph.pdf.

[45] Genesis. "Forward propagation illustrated". In: *Fromthegenesis* (June 2018). URL: https://www.fromthegenesis.com/artificial-neural-network-part-8/.

[46] MissingLink. "7 Types of Neural Network Activation Functions". In: *missinglink.a* (Nov. 2018). URL: https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/.

[47] MissingLink. "Activation Functions Explained - GELU, SELU, ELU, ReLU and more". In: *missinglink.a* (Aug. 2019). URL: https://mlfromscratch.com/activation-functions-explained.

[48] Shervine Amidi and Afshine Amidi. "Architecture of a traditional RNN". In: *Stanford* (). URL: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

[49] Jonathan Hui. *GAN Training Process Illustrated.* 2018. URL: http://miro.medium.com/max/1600/1*9qW0I-2M6qKGBwifhnPKPQ.png (visited on 06/13/2020).

[50] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. "Attention is All you Need". In: *ArXiv* abs/1706.03762 (2017).

[51] Jacob Devlin, Ming-Wei Chang, Kenton Lee, et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://www.aclweb.org/anthology/N19-1423.

[52] A. Radford, Jeffrey Wu, R. Child, et al. "Language Models are Unsupervised Multitask Learners". In: 2019.

[53] A. Radford. "Improving Language Understanding by Generative Pre-Training". In: 2018.

[54] Jay Alammar. "Illustrated GPT-2". In: *jalammar* (Aug. 2019). URL: https://jalammar.github.io/illustrated-gpt2/.

[55] P. B. Baxendale. "Machine-Made Index for Technical Literature: An Experiment". In: *IBM J. Res. Dev.* 2.4 (Oct. 1958), pp. 354–361. ISSN: 0018-8646. DOI: 10.1147/rd.24.0354. URL: https://doi.org/10.1147/rd.24.0354.

[56] H. P. Luhn. "A Business Intelligence System". In: *IBM J. Res. Dev.* 2.4 (Oct. 1958), pp. 314–319. ISSN: 0018-8646. DOI: 10.1147/rd.24.0314. URL: https://doi.org/10.1147/rd.24.0314.

[57] G. Salton and C. S. Yang. "On the specification of term values in automatic indexing." In: *Journal of Documentation.* 29.4 (1973), pp. 351–372.

[58] H. P. Edmundson. "New Methods in Automatic Extracting". In: *J. ACM* 16.2 (Apr. 1969), pp. 264–285. ISSN: 0004-5411. DOI: 10.1145/321510.321519. URL: https://doi.org/10.1145/321510.321519.

[59] Mohamed Abdel Fattah and Fuji Ren. "GA, MR, FFNN, PNN and GMM Based Models for Automatic Text Summarization". In: *Comput. Speech Lang.* 23.1 (Jan. 2009), pp. 126–144. ISSN: 0885-2308. DOI: 10.1016/j.csl.2008.04.002. URL: https://doi.org/10.1016/j.csl.2008.04.002.

[60] You Ouyang, Wenjie Li, Qin Lu, et al. "A Study on Position Information in Document Summarization". In: *Coling 2010: Posters*. Beijing, China: Coling 2010 Organizing Committee, Aug. 2010, pp. 919–927. URL: https://www.aclweb.org/anthology/C10-2106.

[61] Elena Baralis, Luca Cagliero, Saima Jabeen, et al. "Multi-Document Summarization Exploiting Frequent Itemsets". In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. SAC '12. Trento, Italy: Association for Computing Machinery, 2012, pp. 782–786. ISBN: 9781450308571. DOI: 10.1145/2245276.2245427. URL: https://doi.org/10.1145/2245276.2245427.

[62] Gerard Salton, Amit Singhal, Mandar Mitra, et al. "Automatic text structuring and summarization". In: *Information Processing and Management* 33.2 (1997). Methods and Tools for the Automatic Construction of Hypertext, pp. 193–207. ISSN: 0306-4573. DOI: https://doi.org/10.1016/S0306-4573(96)00062-3. URL: http://www.sciencedirect.com/science/article/pii/S0306457396000623.

[63] Khushboo Thakkar, Rajiv Dharaskar, and Manoj Chandak. "Graph-Based Algorithms for Text Summarization". In: *Emerging Trends in Engineering and Technology, International Conference on* 0 (Nov. 2010), pp. 516–519. DOI: 10.1109/ICETET.2010.104.

[64] C. D. Paice. "The Automatic Generation of Literature Abstracts: An Approach Based on the Identification of Self-Indicating Phrases". In: *Proceedings of the 3rd Annual ACM Conference on Research and Development in Information Retrieval*. SIGIR '80. Cambridge, England: Butterworth and Co., 1980, pp. 172–191. ISBN: 0408107758.

[65] Kenji Ono, Kazuo Sumita, and Seiji Miike. "Abstract Generation Based on Rhetorical Structure Extraction". In: *COLING 1994 Volume 1: The 15th International Conference on Computational Linguistics*. 1994. URL: https://www.aclweb.org/anthology/C94-1056.

[66] Yuta Kikuchi, Tsutomu Hirao, Hiroya Takamura, et al. "Single Document Summarization based on Nested Tree Structure". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, June 2014, pp. 315–320. DOI: 10.3115/v1/P14-2052. URL: https://www.aclweb.org/anthology/P14-2052.

[67] Kevin Knight and Daniel Marcu. "Summarization beyond sentence extraction: A probabilistic approach to sentence compression". In: *Artificial Intelligence* 139 (July 2002), pp. 91–107. DOI: 10.1016/S0004-3702(02)00222-9.

[68] Hongyan Jing and Kathleen R. McKeown. "The Decomposition of Human-Written Summary Sentences". In: *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '99. Berkeley, California, USA: Association for Computing Machinery, 1999, pp. 129–136. ISBN: 1581130961. DOI: 10.1145/312624.312666. URL: https://doi.org/10.1145/312624.312666.

[69] Su Jeong Choi, Ian Jung, Seyoung Park, et al. "Abstractive Sentence Compression with Event Attention". In: 2019.

[70] Trevor Cohn and Mirella Lapata. "Sentence Compression Beyond Word Deletion". In: *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*. Manchester, UK: Coling 2008 Organizing Committee, Aug. 2008, pp. 137–144. URL: https://www.aclweb.org/anthology/C08-1018.

[71] Katja Filippova. "Multi-Sentence Compression: Finding Shortest Paths in Word Graphs". In: *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*. Beijing, China: Coling 2010 Organizing Committee, Aug. 2010, pp. 322–330. URL: https://www.aclweb.org/anthology/C10-1037.

[72] Katja Filippova, Enrique Alfonseca, Carlos A. Colmenares, et al. "Sentence Compression by Deletion with LSTMs". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 360–368. DOI: 10.18653/v1/D15-1042. URL: https://www.aclweb.org/anthology/D15-1042.

[73] Liangguo Wang, Jing Jiang, Hai Leong Chieu, et al. "Can Syntax Help? Improving an LSTM-based Sentence Compression Model for New Domains". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, July 2017, pp. 1385–1393. DOI: 10.18653/v1/P17-1127. URL: https://www.aclweb.org/anthology/P17-1127.

[74] V. A. Yatsko, M. S. Starikov, and A. V. Butakov. "Automatic Genre Recognition and Adaptive Text Summarization". In: *Autom. Doc. Math. Linguist.* 44.3 (June 2010), pp. 111–120. ISSN: 0005-1055. DOI: 10.3103/S0005105510030027. URL: https://doi.org/10.3103/S0005105510030027.

[75] Kam-Fai Wong, Mingli Wu, and Wenjie Li. "Extractive Summarization Using Supervised and Semi-Supervised Learning". In: *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*. Manchester, UK: Coling 2008 Organizing Committee, Aug. 2008, pp. 985–992. URL: https://www.aclweb.org/anthology/C08-1124.

[76] Julian Kupiec, Jan Pedersen, and Francine Chen. "A Trainable Document Summarizer". In: *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '95. Seattle, Washington, USA: Association for Computing Machinery, 1995, pp. 68–73. ISBN: 0897917146. DOI: 10.1145/215206.215333. URL: https://doi.org/10.1145/215206.215333.

[77] Miles Osborne. "Using maximum entropy for sentence extraction". In: *Proceedings of the ACL-02 Workshop on Automatic Summarization*. Phildadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 1–8. DOI: 10.3115/1118162.1118163. URL: https://www.aclweb.org/anthology/W02-0401.

[78] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262193981.

[79] Gyoung Ho Lee and Kong Joo Lee. "Automatic Text Summarization Using Reinforcement Learning with Embedding Features". In: *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Taipei, Taiwan: Asian Federation of Natural Language Processing, Nov. 2017, pp. 193–197. URL: https://www.aclweb.org/anthology/I17-2033.

[80] Elozino Egonmwan and Yllias Chali. "Transformer-based Model for Single Documents Neural Summarization". In: *Proceedings of the 3rd Workshop on Neural Generation and Translation*. Hong Kong: Association for Computational Linguistics, Nov. 2019, pp. 70–79. DOI: 10.18653/v1/D19-5607. URL: https://www.aclweb.org/anthology/D19-5607.

[81] Yang Liu. "Fine-tune BERT for Extractive Summarization". In: *ArXiv* abs/1903.10318 (2019).

[82] Jimmy Ba, J. Kiros, and Geoffrey E. Hinton. "Layer Normalization". In: *ArXiv* abs/1607.06450 (2016).

[83] Evan Sandhaus. "The New York Times Annotated Corpus". In: abs/1607.06450 (Oct. 2008).

[84] Alexander M. Rush, Sumit Chopra, and Jason Weston. "A Neural Attention Model for Abstractive Sentence Summarization". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 379–389. DOI: 10.18653/v1/D15-1044. URL: https://www.aclweb.org/anthology/D15-1044.

[85] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, et al. "Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond". In: *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 280–290. DOI: 10.18653/v1/K16-1028. URL: https://www.aclweb.org/anthology/K16-1028.

[86] Dzmitry Bahdanau, Kyunghyun Cho, and Y. Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *ArXiv* 1409 (Sept. 2014).

[87] Junyoung Chung, Çaglar Gülçehre, Kyunghyun Cho, et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *ArXiv* abs/1412.3555 (2014).

[88] Jeffrey Ling and Alexander Rush. "Coarse-to-Fine Attention Models for Document Summarization". In: *Proceedings of the Workshop on New Frontiers in Summarization*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 33–42. DOI: 10.18653/v1/W17-4505. URL: https://www.aclweb.org/anthology/W17-4505.

[89] Piji Li, Wai Lam, Lidong Bing, et al. "Deep Recurrent Generative Decoder for Abstractive Text Summarization". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 2091–2100. DOI: 10.18653/v1/D17-1222. URL: https://www.aclweb.org/anthology/D17-1222.

[90] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. "Attention Is All You Need". In: (June 2017).

[91] Alexandre Matton and Amaury Sabran. "Faster Transformers for text summarization". In: 2019. URL: https://pdfs.semanticscholar.org/34be/49a5a23343cb69508ed7b3a4b25f26210d57.pdf?_ga=2.68453907.36616089.1598978775-852984914.1598022094.

[92] Sam Shleifer. 2020. URL: https://github.com/huggingface/transformers/tree/master/examples/seq2seq#distilbart.

[93] Mike Lewis, Yinhan Liu, Naman Goyal, et al. "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 7871–7880. DOI: 10.18653/v1/2020.acl-main.703. URL: https://www.aclweb.org/anthology/2020.acl-main.703.

[94] Dan Hendrycks and Kevin Gimpel. "Gaussian Error Linear Units (GELUs)." In: *arXiv: Learning* (2016).

[95] Chikashi Nobata and Satoshi Sekine. "CRL/NYU Summarization System at DUC-2004". English. In: *Document Understanding Workshop 2004*. May 2004.

[96] Dragomir R. Radev, Hongyan Jing, and Malgorzata Budzikowska. "Centroid-based summarization of multiple documents: sentence extraction, utility-based

evaluation, and user studies". In: *NAACL-ANLP 2000 Workshop: Automatic Summarization*. 2000. URL: https://www.aclweb.org/anthology/W00-0403.

[97] Dragomir Radev and Weiguo Zhang. "WebInEssence: A Personalized Web-Based Multi-Document Summarization and Recommendation System". In: (Sept. 2001).

[98] Yihong Gong and Xin Liu. "Generic Text Summarization Using Relevance Measure and Latent Semantic Analysis". In: *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '01. New Orleans, Louisiana, USA: Association for Computing Machinery, 2001, pp. 19–25. ISBN: 1581133316. DOI: 10.1145/383952.383955. URL: https://doi.org/10.1145/383952.383955.

[99] Josef Steinberger and Karel Jezek. "Using Latent Semantic Analysis in Text Summarization and Summary Evaluation". In: Jan. 2004.

[100] Xiaojun Wan. "An Exploration of Document Impact on Graph-Based Multi-Document Summarization". In: *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*. Honolulu, Hawaii: Association for Computational Linguistics, Oct. 2008, pp. 755–762. URL: https://www.aclweb.org/anthology/D08-1079.

[101] Rada Mihalcea and Paul Tarau. "TextRank: Bringing Order into Text". In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 404–411. URL: https://www.aclweb.org/anthology/W04-3252.

[102] Xiaojun Wan. "TimedTextRank: Adding the Temporal Dimension to Multi-Document Summarization". In: *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '07. Amsterdam, The Netherlands: Association for Computing Machinery, 2007, pp. 867–868. ISBN: 9781595935977. DOI: 10.1145/1277741.1277949. URL: https://doi.org/10.1145/1277741.1277949.

[103] Nitin Madnani, David Zajic, Bonnie Dorr, et al. "Multiple alternative sentence compressions for automatic text summarization". In: *In Proceedings of the 2007 Document Understanding Conference (DUC-2007) at NLT/NAACL 2007*. 2007.

[104] Asli Celikyilmaz and Dilek Hakkani-Tür. "Discovery of Topically Coherent Sentences for Extractive Summarization". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 491–499. URL: https://www.aclweb.org/anthology/P11-1050.

[105] Seonggi Ryang and Takeshi Abekawa. "Framework of Automatic Text Summarization Using Reinforcement Learning". In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Jeju Island, Korea: Association for Computational Linguistics, July 2012, pp. 256–265. URL: https://www.aclweb.org/anthology/D12-1024.

[106] Cody Rioux, Sadid A. Hasan, and Yllias Chali. "Fear the REAPER: A System for Automatic Multi-Document Summarization with Reinforcement Learning". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 681–690. DOI: 10.3115/v1/D14-1075. URL: https://www.aclweb.org/anthology/D14-1075.

[107] Sam Shleifer. *Introducing BART*. Mar. 2020. URL: https://github.com/sshleifer/blog_v2/blob/master/_notebooks/2020-03-12-bart.ipynb.

[108] Tomas Mikolov, Ilya Sutskever, Kai Chen, et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems* 26 (Oct. 2013).

[109] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://www.aclweb.org/anthology/D14-1162.

[110] SlashData. *The global developer population 20019: how many developers are there*. 2019, p. 17. URL: https://slashdata-website-cms.s3.amazonaws.com/sample_reports/EiWEyM5bfZe1Kug_.pdf.

[111] Juan Carlos. *Scalars, vectors, matrices and tensors*. May 2019. URL: https://dev.to/juancarlospaco/tensors-for-busy-people-315k.

[112] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, et al. "Teaching machines to read and comprehend". In: *Advances in neural information processing systems*. 2015, pp. 1693–1701.

[113] Rui Zhang and Joel Tetreault. *This Email Could Save Your Life: Introducing the Task of Email Subject Line Generation*. 2019. arXiv: 1906.03497 [cs.CL].

[114] Alexander R. Fabbri, Irene Li, Tianwei She, et al. *Multi-News: a Large-Scale Multi-Document Summarization Dataset and Abstractive Hierarchical Model*. 2019. arXiv: 1906.01749 [cs.CL].

[115] Junbo Kong Robert Parker David Graff and Kazuaki Maeda Ke Chen. "English Gigaword Fifth Edition". In: (June 2011).

[116] Matthew R. Gormley Courtney Napoles and Benjamin Van Durme. "Annotated English Gigaword". In: (Nov. 2012).

[117] Ani Nenkova. "Automatic Text Summarization of Newswire: Lessons Learned from the Document Understanding Conference". In: *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*. AAAI'05. Pittsburgh, Pennsylvania: AAAI Press, 2005, pp. 1436–1441. ISBN: 157735236x.