Automatic Text Summarization

João Saraiva

Dissertação para obtenção de Grau de Mestre em
Engenharia Informática

Supervisor: Prof. João Cordeiro

Junho, 2021

Dedication

I dedicate this thesis and all the work it meant to my beloved family and girlfriend. They will be forever in my heart.

Acknowledgment

As acknowledgment, i would like to start by stating the huge support that my closest family members gave me, with special attention to my parents, my girlfriend, my sister and brother in law and also my grandmother and aunt, during this years of college. It was crucial and allowed me to reach this noble phase of my life. My first master's year was pretty relaxed and i have learned a lot in many subjects. In the second year, i began working on my master's thesis. Having had a couple of subject's regarding artificial intelligence and natural language processing made it easier for me in deciding what thesis to work on. The natural language processing's subject sparked an interested in me, with all the possibilities that it presents, so i decided to follow the path of Automatic Text Summarization. So, i also would like to thank my teacher, João Paulo Cordeiro, for helping me throughout this tough phase, since he never gave up on me or doubted about my skills. He was very patient and gave me a lot of support and knowledge in many aspects.
Finally, i would like to mention the help that my colleagues and friends gave me during this two years of my masters degree.

Resumo

O texto é um dos utensílios mais importantes de transmissão de ideias entre os seres humanos. Pode ser de vários tipos e o seu conteúdo pode ser mais ou menos fácil de interpretar, conforme a quantidade de informação relevante sobre o assunto principal.
De forma a facilitar o processamento pelo leitor existe um mecanismo propositadamente criado para reduzir a informação irrelevante num texto, chamado sumarização de texto. Através da sumarização criam-se versões reduzidas do text original e mantém-se a informação do assunto principal.
Devido à criação e evolução da Internet e outros meios de comunicação, surgiu um aumento exponencial de documentos textuais, evento denominado de sobrecarga de informação, que têm na sua maioria informação desnecessária sobre o assunto que retratam.
De forma a resolver este problema global, surgiu dentro da área científica de Processamento de Linguagem Natural, a sumarização automática de texto, que permite criar sumários automáticos de qualquer tipo de texto e de qualquer lingua, através de algoritmos computacionais.
Desde a sua criação, inúmeras técnicas de sumarização de texto foram idealizadas, podendo ser classificadas em dois tipos diferentes: extractivas e abstractivas. Em técnicas extractivas, são transcritos elementos do texto original, como palavras ou frases inteiras que sejam as mais ilustrativas do assunto do texto e combinadas num documento. Em técnicas abstractivas, os algoritmos geram elementos novos.
Nesta dissertação pesquisaram-se, implementaram-se e combinaram-se algumas das técnicas com melhores resultados de modo a criar um sistema completo para criar sumários.
Relativamente às técnicas implementadas, as primeiras três são técnicas extractivas enquanto que a ultima é abstractiva. Desta forma, a primeira incide sobre o cálculo das frequências dos elementos do texto, atribuindo-se valores às frases que sejam mais frequentes, que por sua vez são escolhidas para o sumário através de uma taxa de compressão. Outra das técnicas incide na representação dos elementos textuais sob a forma de nodos de um grafo, sendo atribuidos valores de similaridade entre os mesmos e de seguida escolhidas as frases com maiores valores através de uma taxa de compressão. Uma outra abordagem foi criada de forma a combinar um mecanismo de análise das caracteristicas do texto com métodos baseados em inteligência artificial. Nela cada frase possui um conjunto de caracteristicas que são usadas para treinar um modelo de rede neuronal. O modelo avalia e decide quais as frases que devem pertencer ao sumário e filtra as mesmas através deu uma taxa de compressão. Um sumarizador abstractivo foi criado para para gerar palavras sobre o assunto do texto e combinar num sumário. Cada um destes sumarizadores foi combinado num só sistema. Por fim, cada uma das técnicas pode ser avaliada segundo várias métricas de avaliação, como por exemplo a *ROUGE*. Segundo os resultados de avaliação das técnicas, com o conjunto de dados *DUC*, os nossos sumarizadores obtiveram resultados relativamente parecidos com os presentes na comunidade cientifica, com especial atenção para o codificador-descodificador que em certos casos apresentou resultados promissores.

Abstract

Writing text was one of the first ever methods used by humans to represent their knowledge. Text can be of different types and have different purposes.
Due to the evolution of information systems and the Internet, the amount of textual information available has increased exponentially in a worldwide scale, and many documents tend to have a percentage of unnecessary information. Due to this event, most readers have difficulty in digesting all the extensive information contained in multiple documents, produced on a daily basis.
A simple solution to the excessive irrelevant information in texts is to create summaries, in which we keep the subject's related parts and remove the unnecessary ones.
In Natural Language Processing, the goal of automatic text summarization is to create systems that process text and keep only the most important data. Since its creation several approaches have been designed to create better text summaries, which can be divided in two separate groups: *extractive* approaches and *abstractive* approaches.
In the first group, the summarizers decide what text elements should be in the summary. The criteria by which they are selected is diverse. After they are selected, they are combined into the summary. In the second group, the text elements are generated from scratch. Abstractive summarizers are much more complex so they still need a lot of research, in order to represent good results.
During this thesis, we have investigated the state of the art approaches, implemented our own versions and tested them in conventional datasets, like the DUC dataset.
Our first approach was a frequency-based approach, since it analyses the frequency in which the text's words/sentences appear in the text. Higher frequency words/sentences automatically receive higher scores which are then filtered with a compression rate and combined in a summary.
Moving on to our second approach, we have improved the original TextRank algorithm by combining it with word embedding vectors. The goal was to represent the text's sentences as nodes from a graph and with the help of word embeddings, determine how similar are pairs of sentences and rank them by their similarity scores. The highest ranking sentences were filtered with a compression rate and picked for the summary.
In the third approach, we combined feature analysis with deep learning. By analysing certain characteristics of the text sentences, one can assign scores that represent the importance of a given sentence for the summary. With these computed values, we have created a dataset for training a deep neural network that is capable of deciding if a certain sentence must be or not in the summary.
An abstractive encoder-decoder summarizer was created with the purpose of generating words related to the document subject and combining them into a summary. Finally, every single summarizer was combined into a full system.
Each one of our approaches was evaluated with several evaluation metrics, such as ROUGE. We used the DUC dataset for this purpose and the results were fairly similar to the ones in the scientific community. As for our encoder-decode, we got promising results.

Keywords

Natural Language Processing, Automatic Text Summarization, Extractive Summarization, Abstractive Summarization, Deep Learning.

Resumo alargado

Um texto é um utensílio muito importante e consiste numa das maneiras mais simples que os seres humanos têm de partilhar e representar os seus conhecimentos. Pode ter diferentes naturezas, desde ser um simples texto de uma notícia a ser um documento científico. Existem textos mais pequenos e simples de interpretar, como também outros maiores e mais complexos.

Dependendo do caso, torna-se mais ou menos difícil um leitor interpretar um documento, porém há mecanismos capazes de tornar este processo mais fácil. Um documento pode desta forma, ser sumarizado ou resumido, ou seja, deve-se manter o seu assunto principal e representá-lo de uma forma mais clara.

Desde a criação do primeiro computador e consequente globalização da Internet, tem havido um aumento exponencial na criação de novos conteúdos e partilha dos mesmos, originando um fenómeno denominado sobrecarga de informação. A quantidade de informação disponível é absurdamente grande e em alguns casos desnecessária. Muitos documentos de texto na Internet contêm informações consideradas irrelevantes para quem está a ler, deste modo, torna-se demoroso e difícil de o leitor encontrar a informação representativa do tema. O leitor necessita que este processo de interpretação seja o mais eficiente possível facilitando assim a sua aprendizagem sobre o assunto.

Tal como foi referido em cima, existem mecanismos que filtram as informações desnecessárias do texto, a fim de proporcionar uma experiência única ao leitor, visto que ele pode simplesmente focar-se na informação apenas referente ao assunto. Dentro da área científica de Processamento de Linguagem Natural, existe um ramo denominado de sumarização automática de texto, cujo objectivo é precisamente resolver problemas deste tipo de uma forma automática, através de algoritmos computacionais. Este ramo começou a ser desenvolvido na década de 60 e desde então inúmeras soluções foram propostas.

O objetivo principal desta dissertação é portanto, procurar e estudar as várias técnicas de sumarização automática de texto e implementá-las com vista a criar sumários que possuam apenas a essência dos documentos e que sejam gramaticalmente correctos, coerentes e interessantes de interpretar.

As técnicas de sumarização automática podem-se classificar em dois grupos principais: extractivas e abstractivas. Nas técnicas extractivas, os pedaços de um texto mais importantes, sejam eles frases ou palavras são diretamente removidos do mesmo e aglomerados de forma sistemática e ordenada no sumário. Assim os elementos do texto são avaliados e os considerados mais relacionados com o assunto principal do texto devem permanecer no sumário. Nas técnicas abstrativas, pelo contrário, são gerados novos elementos do texto, desde palavras, frases e parágrafos. Este tipo de abordagens ainda é relativamente recente e muito complexa, pelo que, ainda necessita de muita investigação e os seus resultados ainda não são muito relevantes, quando comparados com os resultados de algumas abordagens extractivas.

Nesta dissertação foram exploradas abordagens extractivas e abstractivas, segundo o estado da arte. Muitas delas apresentam resultados inovadores, pelo que, focamos a nossa atenção nelas.

Na fase da implementação começámos por incorporar os avanços obtidos em algumas técnicas e adaptá-los, criando a nossa própria versão das técnicas com o objectivo de melhorar os resultados das mesmas. Foi criada uma técnica que tem por base a avaliação da frequência das palavras e frases do texto, atribuindo maiores valores às que são mais frequentes. Posto isto, os elementos mais frequentes e consecutivamente com maiores resultados, são os escolhidos para o texto através de uma taxa de compressão. Noutra técnica, combinou-se um algoritmo muito conhecido para sumarização automática, o algoritmo *TextRank*, com vetores de incorporação de palavras, ou seja, *Word Embeddings*, a fim de representar os elementos do texto num grafo e avaliar como eles são relacionados, com a ajuda dos vetores de palavras. Às frases consideradas mais importantes, são atribuidos maiores valores e tal como na técnica anterior, estas são as frases escolhidas para criar o sumário, sendo filtradas igualmente com uma taxa de compressão. Seguidamente foi criada uma nova técnica, completamente de raiz, que combina a avaliação dos elementos do texto com métodos baseados em Inteligência Artificial. A lógica subjacente a este algoritmo consiste em avaliar os elementos do texto, segundo diversas maneiras, como por exemplo a frequência em que aparecem no texto, a sua localização no texto, o nº de elementos raros que possuem (palavras titulo, siglas, palavras numéricas ou temporais, etc), atribuir valores aos mesmos e utilizar o resultado desta avaliação como dados para um modelo de inteligência artificial, mais concretamente uma rede neuronal. O papel do modelo, após treino com estes dados, é se uma frase percente ou não ao sumário. As frases que forem classificadas como frases do sumário, são filtradas com uma taxa de compressão e combinadas no sumário. Seguidamente foi criado um sumarizador abstractivo que utiliza um modelo de *deep learning* para gerar palavras do texto, segundo as suas distribuições probabilísticas, combinando-as nos sumários. Por fim, combinamos todos os sumarizadores apresentados num só.

Cada um destes sumarizadores pode ser avaliado segundo diferentes métricas de avaliação, sendo a mais conhecida a métrica *ROUGE*. Da mesma forma, foram utilizados vários conjuntos de dados para treinar e testar os sumarizadores, com especial atenção para o conjunto de dados DUC, que possui dados de conferências para sumarização automática de texto.

De acordo com os resultados, podemos ver que a maioria das nossas abordagens apresentou resultados similares aos encontrados na literatura. Podemos ver que abordagens baseadas em inteligência artificial apresentam resultados interessantes, com especial atenção para o nosso sumarizador abstractivo. Este apesar de criar sumários sem pontuação e com pouca coerência apresenta os melhores resultados com a métrica *Recall-Oriented Understudy for Gisting Evaluation (ROUGE)* entre todos os algoritmos que experimentámos. Tanto esta abordagem como a da rede neuronal precisam de muitos dados de treino e este é um processo custoso e demoroso.

Contents

List of Figures

List of Tables

Acronyms

NLP Natural Language Processing

NLTK Natural Language Toolkit

JSON JavaScript Object Notation

NLG Natural Language Generation

ATS Automatic Text Summarization

DUC Document Understanding Conference

ATG Automatic Title Generation

TAC Text Analysis Conference

POS Part Of Speech

FFNN Feed Forward Neural Network

CRF Conditional Random Fields

LSA Latent Semantic Analysis

SVD Singular Value Decomposition

HITS Hyperlink-Induced Topic Search

MMR Maximal Marginal Relevance

SVD Singular Value Decomposition

TF Term Frequency

DF Document Frequency

IDF Inverse Document Frequency

TFIDF Term Frequency Inverse Document Frequency

BERT Bidirectional Encoder Representations from Transformer

ROUGE Recall-Oriented Understudy for Gisting Evaluation

CDST Cross-Document Structure Theory

HTML Hypertext Markup Language

NER Named Entity Recognition

HMM Hidden Markov Models

CPU Central Processing Unit

GPU Graphic's Processing Unit

RSG Rich Semantic Graph

TFISF Term Frequency * Inverse Sentence Frequency

AI Artificial Inteligence

API Application Programming Interface

GUI Graphical User Interface

IDE Integrated Development Environment

SCU Summarization Content Units

IR Information Retrieval

CSV Comma-Separated Values

LDA Latent Dirichlet Allocation

LSTM Long-Short Term Memory

BLEU Bilingual Evaluation Understudy

ML Machine Learning

RELU Rectified Linear Unit

Chapter 1

Introduction

Textual information has been around for centuries, it was first conceived as a set of signs, that primordial humans used to transmit their ideas to each other. Since then, it has been used for so much more: representing simple thoughts, elaborate art or scientific documents, send messages or even create social media posts. We can abstractly define text as a fine and delicate fabric of ideas that we weave together.

Nowadays, with the advancements in technology, analogical textual information is slowly losing its importance in the way we express ourselves, because we have the possibility to create documents, without having to move a single pencil.

Thanks to the Internet, we have at our disposal a huge amount of information, that can be accessed from anywhere. The Internet has made possible the search and creation of new data, in a more efficient way. We no longer have to go to the library to search for a specific document.

Since we are exponentially increasing the amount of data online, we need more powerful mechanisms to assure that a specific document is the one we are looking for and that we have the means to understand it. Most documents are often too big and have pieces of text that are irrelevant for our needs.

A way of fighting this problem is to manually create summaries. This however, is an impracticable solution, because it is often hard to just create a summary of a single text. If we take into count that we need to create summaries of hundreds or thousands of documents, it is an even more complicated task. It would take a lot of time to be done and we would also need very experienced people.

Instead of using humans to create summaries, we can use automatic text summarizers that can create a big number of summaries and represent their content in a different way than humans. This is due to the fact that they do not depend on the level of knowledge of a human summarizer, but they are guided by metrics to process the data.

This and other text modeling problems can be fixed with the help of Natural Language Processing (NLP), which is a field that combines both Computer Science, Artificial Intelligence and Linguistics. In this case, we are interested in Automatic Text Summarization (ATS), a very important sub-field of NLP. It is responsible for understanding and creating powerful techniques that, when combined, make possible the generation of a smaller version of a text document. This smaller version is called a summary and even though there is not a clear and precise definition accepted by the scientific community, a set of rules must be followed in the summary generation process. The two most important rules are: the summary must represent all the important aspects of the original text; the summary must not be bigger than half of the original document size [10].

To decide what should be the length of the summary, one can use compression or retention rates, that assume the best proportion between the dimensions of the original text and the

summary text. It can be calculated by defining a percentage of the desired text length, regarding the original text size or a certain number of sentences.

In terms of the evolution during the years, Luhn and Baxendale were one of the pioneers in the expansion of automatic text summarization. In the 1950s, they studied possible ways of creating text summaries with the help of machines. Baxendale proposed a method that analysed some text features, such as: sentence position and the amount of title and important words in the text [11]. Later in that decade, Luhn also had an important role in proposing a different approach for creating automatic summaries.

In that year, even though the technology was not very evolved, a huge amount of information was created in a daily basis, and machines from that time, had difficulties processing the information. There was a need to transform the textual information, into a much simpler representation for the machines.

The next years there was a lack of research in automatic text summarization, in which only in the decade of 1960 was proposed a new approach: an improvement of Lunhs research, by Edmunson. In this new approach, the machine must select words from the text that convey more information about the theme of the document [12]. In the 1990s, after a few decades of weak research, newer automatic approaches were developed, like: better approaches based on feature analysis, approaches based on lexical and semantic analysis, approaches based on artificial intelligence, approaches based on graph theory, etc.

Later, some conferences were created for Text Summarization, being the Document Understanding Conference (DUC) and Text Analysis Conference (TAC) the more important ones. In these conferences, newer techniques or improvement to existing techniques are proposed by researchers. The evaluation process uses a set of metrics that decides how good are the summaries generated by those techniques. Examples of metrics for summary evaluations are ROUGE.

Below, we will present two pieces of text: the first one is part of an original text document and the second is an example of an automatic summary for the same text. The summary was generated by extracting a certain number of sentences according to a compression rate.

Original text and its summary

Portugal was once at the edge of the known world. It made it the perfect launchpad for many pioneering expeditions during the rampant age of discovery. Travel and adventure simply runs in the Portuguese blood. Once one of the world's biggest empires, it's now a more humble but fascinating slice of western Europe containing endless evidence of its former conquests and distant colonies.

Portugal was once at the edge of the known world. Once one of the world's biggest empires, it's now a more humble but fascinating slice of western Europe containing endless evidence of its former conquests and distant colonies.

1.1 Motivation

Nowadays with the advancements in the area of NLP, more specifically in the field of ATS, several approaches and resources have been used with the goal of creating efficient and coherent text summaries. Most of those techniques fail or they are not good enough. Therefore, the main motivation for this thesis is to explore the current state of the art approaches, discover those that are potentially more beneficial and attempt to create an efficient summarizer.

1.2 Objectives

The main goal of this thesis is to elaborate an automatic text summarizer that is able to reduce the textual information of a document, originating a smaller text that keeps all the essential information.
Since there are various possibilities to create an automatic text summarizer, we first need to study the majority of them, focusing on the ones that present the best results.
After the study has been done, we must create an hybrid system, that combines characteristics from the best approaches and creates a text summary.
The performance of our system must be evaluated with the use of summarization metrics, such as ROUGE, and with data-sets such as DUC, TAC, among others.

1.3 Document Structure

The structure of this thesis is quite simple and is as follows:

i **Introduction** - In this first and current chapter, we give an overview on what consists our project, motivation and the objectives;

ii **State of the art** - In this chapter, we identify and explain, in detail, the advancements that the area of automatic text summarization had during the years;

iii **Implementation** - This third chapter consists in explaining how our automatic text summarizer was made, in detail;

iv **Evaluation measures and results** - The results obtained with our summarizers are present and explained in this chapter;

v **Conclusions and future word** - In the final chapter we conclude the themes addressed in this thesis and present some ideas and improvements that could be implemented in the future;

Chapter 2

State Of The Art

Text summarization is an activity that keeps gaining more and more popularity nowadays, due to the fact that new information is continuously being created in large quantities and fed to the world. This information consists of simple concepts, social media posts or even whole documents, such as: newspaper articles, book chapters, whole books (in physical or digital formats, for instance *E-books*) or scientific papers. Considering the fact that the amount of this textual information available worldwide is enormous, the whole process of reading, interpreting and consequently writing a smaller version of a text document is, in general, a very difficult and slow task to be performed.
Human beings perform and practice this kind of reasoning process, most of the times in a subconscious way, because we need to simplify the ideas that go through our minds in order to acquire new knowledge, but this requires the elaboration of high complexity thoughts.
It is at this point that text summarization enters, in an automatic fashion, as a result of fastening a computational mechanism that is capable of simulating the way a human summarizes textual information. This computational mechanism must receive a text set as input, analyse it with help of mathematical calculations and heuristics, and then create a shortened version, that contains all the essence and most important aspects of the original text (without any human intervention).
Along this chapter, all the subjects regarding text summarization will be explained in detail. We will start by explaining the concept of text summarization, followed by human summarization and finally automatic text summarization. The main focus of the chapter will be on automatic text summarization, in which we will present the computational approaches that have been developed during the years and spotlight those that had the biggest impact in the area of NLP. Later we present the metrics for evaluation of automatic text summarization and finally the biggest problems with automatic text summarization are explained.

2.1 Human Summaries

Summarization, as it was briefly introduced above, consists of an activity that is very common nowadays because in the different areas in which humans work, from academic to professional, there is a need to efficiently process big amounts of text data in order to understand the subjects involved.
Different authors tried to propose a clear and unique definition of what a summary is, but they have universally failed because they could not create a definition that was accepted by everyone.
Sparck Jones, in 1999, defines a summary as: “a reductive transformation of source text to summary text through content reduction by selection and/or generalisation on what is im-

portant in the source“ [1]. A more concise definition of a summary was proposed by Eduard Hovy: **“a text that is produced from one or more texts, that contains a significant portion of the information in the original text(s), and that is no longer than half of the original text(s)“** [13].
A summary can be made from all kinds of pieces of text, such as: multimedia documents, on-line documents, hyper-texts, among others [13]. It should be accurate and the writer must strive to stay as close as possible to the maximum objectivity, even though 100% is not achievable. A summary writer must be able to paraphrase, which means he must express the author's ideas in his own words. He also should not include his own interpretations in the summary text [14].
Summarizing a text document allows the reader to understand the subject of the text more easily, because it removes the parts that have no relevant information and keeps the essence of the subject. Therefore, a summary conveys important information from the original text and its size is usually no longer than half of the original text or even less than that [15] [16].
The Colorado State University's website, *Writing@CSU*, defines three main types of human made summaries, as follows [14]:

i **Main Point Summary** - presents the main facts of the original text with an identification of the title, author and main topic. A simple citation of the text is normally used in this kind of summary and it shall represent the main topic. A simple example would be the creation of a college scientific paper with the aim of introducing a subject to the reader;

ii **Key Point Summary** - this type of summaries is much a like the one presented above, in which there is a small citation along with the main facts. In addition, it has the reasons and evidences that the writer used to support the subject of the text. It is normally used to explain the author's idea;

iii **Outline Summary** - this kind of summary includes the main points and arguments as they appear in the text.

2.2 Automatic Text Summarization

In this section we will explain in detail, what is in fact the field of automatic text summarization. We will begin by referring the main aspects regarding its history and then proceed to how is the constitution of an automatic text summarizer, how it is evaluated and classified.

2.2.1 Historical Evolution

In the past few years, due to the increase of information that has been created and spread across the internet and social media, there has been a surge of interest in systems that can automatically summarize textual information. They must receive a document as input, analyse it and return a smaller text as output, retaining its information. This is an automatic procedure because the system must do it without the need of human interaction.

These systems are created with the use of NLP, a field with roots in Computer Science, Artificial Intelligence and Linguistics. The expression "natural language" mean that it is a language that is used by humans in a daily basis for communication.
NLP has a wide variety of tasks, such as machine translation, language identification, sentiment analysis, automatic speech recognition, Part Of Speech (POS) tagging, among others[17].
Text summarization is also part of this gigantic field and NLP scientists have been researching on it, during the last half-century.
Before the Internet appeared, in the early sixties, there was a special effort by Lunh with a research basically driven by statistics, [1]. In that time, the machines did not have much textual information and the information they had was difficult to process, unfortunately leading to low quality of the summaries and hinder the research for some while. In 1969, Edmunson continued the work done by Lunh, by proposing his own method: an heuristic method that uses a combination of features as opposed to traditionally used feature weights generated using a corpus. He combined the features proposed by Luhn, such as: position and word frequency, with two other features: cue words and document structure [18]. In the mid eighties and early nineties, research was continued by Endres-Niggemeyer, Hobbs and Sparck Jones (1995), in particular after the famous "The Dagstuhl Seminar" in 1993. It is also worth mentioning the work done by Mani and Maybury (1997) and the 1997 ACL Workshop (ACL-97), specially dedicated to the subject of automatic text summarization [1].
Statistical and hybrid techniques in other areas also helped the implementation of text summarizers, the NLP tools now available, e.g., for parsing, have made system building and task experiment easier.
Due to the growth of publicly available text, resulted from the creation of the Web, the work on summarization increased significantly, having it pushed to new methods of summarization with the introduction of new kinds of inputs, summarising purposes, and output styles. Also, new challenges occurred: summarising multiple text documents that have very similar subjects; using summaries to quickly filter text in search engines. So, having the need to summarise a big amount of documents, increase the need to have even better summarising systems, therefore encouraging the evaluation programs.

2.2.2 Stages of Automatic Summarization

Researchers like Karen Sparck Jones, Hovy and Lin, Mani and Maybury have done quite an amount of research in the vast area of automatic text summarization and have identified three main stages in this field: **topic interpretation**, **topic fusion (interpretation)** and **summary generation** [13] [19] [20] [21]. Figure 2.1 exemplifies the three stages that we have just discussed.

i **Topic interpretation** - Topic interpretation is the first of the three stages and it is in this stage that the interpretation of the topic happens. A topic of a text is a particular subject. In topic interpretation, the summarizer uses different independent modules that assign a value to each unit of input, for instance: words or sentences. After this, another module must combine the score for each one of those units. This module will obtain a number of the most important units, which is equal to the summary length that the user wants [22].

In order to increase the efficiency of this stage, [23] proposed the extraction of smaller units, creating smaller summaries, but with more information. [24] proposed a different method that increased coherence, by including the sentences that are adjacent to other major sentences in the text. Each one of the independent modules have a certain degree of performance associated to how the topic interpretation process occurs.

ii **Topic transformation** - Topic transformation, is the stage following topic interpretation. After the topic has been analysed and scores have been assigned to the input units, this stage creates new terms, using a new formulation and new concepts that are not used in the original text. The problem with this stage is that the summarizer needs to know about the domain of the text *a priori* and only a few summarizers are capable of doing it.

In order to solve the fusion problem, Lin and Hovy (1997-1998) invented the topic signatures [22]. A topic signature consists of a *headword* and a set of other words, with the respective values that represent the associations between these words and the headword, i.e. they identify a complex concept, that relates to other words[25].

Topic signatures identify important and complex concepts that are related to a headword. This method relies on a big amount of texts (from Wall Street Journal) and the statistical values of each word in the texts, to guide the summarizer to identify what are the words that relate to each topic [25] [22] [13] [21].

iii **Generation** - Finally we reach the third and last stage of summarization: summary generation. After the summarizer finished creating the content of the summary by fetching information, it is stored in the computer in an internal abstract representation, natural language techniques are needed to reformulate this representation into a new coherent and simple text. This stage is important for the quality of the final summary, without it the output would consist of meaningless quotations of the input[25] [21].

If the content has been created by extraction, then the result tends to have some dysfluencies[1] [13]. These are problems with the grammatical connections between sentences of the summary, because it implies that, the sentences that are chosen to be part of the summary, may be printed in a wrong order of importance or in a wrong text order. Examples of these problems are: repetition of words/clauses or named entities or simple inclusion of unnecessary material, like parenthesis [13].

Knight and Marcu proposed a solution to these problems with an award-winning paper about *Text Compression* [26]. This solution allows the conversion of sentences into a smaller ones, through the compression of syntactic parse trees of those sentences.

Other also important approaches for summary generation have been proposed by: Jing and McKeown, using a hidden Markov model to identify the major fragments of sentences and then combine them grammatically into the summary [27]; and by Witbrock and Mittal: by simply extracting words from the document and the order them, using a bi-gram model [28].

[1] Involuntary disruptions or discourse blocks

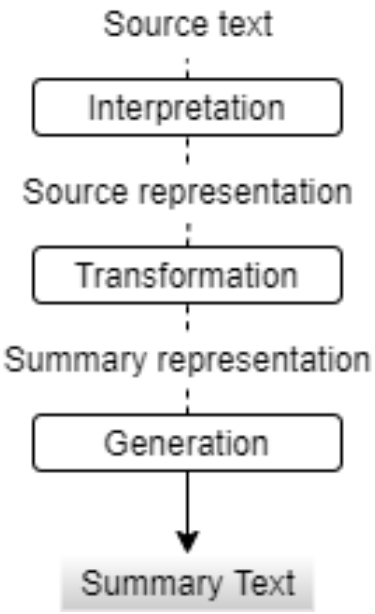

Figure 2.1: Text summarization phases
by Spark Jones [1].

2.2.3 Summary Classifications

Summaries can be very different from each other, in a lot of aspects. In 1998, Sparck Jones proposed a taxonomy[2] to classify summarization systems according to different aspects, named **context factors**. These context factors are divided in three classes: **input factors, output factors** and **purpose factors** [1] [21]. Hovy, Linn, Mani and Maybury also proposed their own taxonomies. Karen's taxonomy is pretty complete in its own, but we will add some more aspects to it that are considered important.
Context factors are factors (some of them are kind of abstract) that characterize a summarization system. They are, as proposed by Karen Sparck, very hard to define [19]. They are divided in three main classes:

i **input factors** give the possibility to characterize the input document, i.e. the source text;

ii **output factors** characterize the summary text;

iii **purpose factors** express the use of the summary.

In the class of **input factors**, we have the following set of characteristics:

i **Number of documents**: the input received by the summarizer can consist of just one document, being classified as **single-document** or can consist of multiple documents, therefore classified as **multi-document**. In this last case, the summary has content from several documents that might be related to each other;

[2]The process of naming and classifying things into groups within a larger system, according to their similarities and differences.

ii **Specificity**: the specificity means how specific to a certain domain is the input of the summarizer. If the input is related to a single domain and the summarizer must use techniques that are useful for that specific domain, then we are talking about a **domain-specific** summary. In a different way, if the summary is made from characteristics from a certain genre it is a **genre-specific** summary. Additionally, if the summary is created for a general use, not being dependent on the domain and using a superficial analysis of the input it is an **independent** summary.

iii **Document language/s**: a summary can be made of documents that are written in several languages. If the input text is written in a single language and the output is also written in a single language, we have a **mono-lingual** summary. If the input document or documents have been written in different languages, for instance: one document is written in Portuguese and the other in English, the summary will be created in both of this languages, being named **multi-lingual** summary. In the last case, if the summarizer deals with input documents written in one language and the summary is created in a different language, then it is a **cross-lingual** summary.

iv **Genre and size**: The genre of the input documents can be of several types, such as: books, articles, newspaper text, opinions, stories, reports, biblical texts, encyclopedia texts, etc. As for the size, it is the size of the document, for instance: a full book or just a small paragraph [21];

v **Media**: Media is also considered an input factor, in the sense that it classifies the input document in simple text, images, video, speech or even hypertext [29].

As for the **output factors**, they are as follows:

i **Derivation**: in terms of derivation a summary can be of three types: **extractive**, **abstractive** or **semi-extractive** If the summary is made of sets of terms (words or sentences) extracted from the text, then it is an **extract**. Methods known for extractive summarization are for instance: graph-based [30], centroids [31], latent semantic analysis [32] [33], machine learning [34], among others [35].

If the summary is made of newly generated text from some kind of analysis and transformation of the input, then it called an **abstract**.

Very similar to extractive approaches are the **semi-extractive** approaches, where they try to create a summary from the extracted text, but with the main difference that they do not just select sentences, but they also select words or phrases, and the merging of phrases from different sentences [35];

ii **Coherence**: A summary can be analysed in terms of coherence, which means that a summary can be correctly written by following an order and a structure, where the sentences are related to each other, thus being classified as **fluent**. In the opposite case, if the summary is badly written, fragmented and not coherent, then it is classified as **disfluent** [21];

iii **Partiality**: In certain cases, the summarizer may add his own "bias". This is the summarizers opinion. In such cases, the summary is classified as **evaluative**, otherwise it is classified as **neutral**;

iv **Conventionality**: The final output factor to consider is the conventionality, which is the scenario for which the summary is made, that can be divided in two classifications: **fixed summary** or **floating summary**. If it is meant for a single scenario then it must be created for a pre-defined use and specific reader — **fixed summary** — otherwise it is made for a wide variety of uses, readers, and purposes — **floating summary**.

Finally, we have the following **purpose factors**:

i **Audience**: A summary that is made of all the major themes of the text is a **generic** summary, while a summary that just focuses on a small group of themes of the text summary, due to the need of the user to know more about those themes in particular is a **query-oriented**;

ii **Usage**: When the summary is made to indicate the principal subject of the input text, then it is an **indicative** summary. When the summary has relevant information about what was the input text about, but does not explain the information in the input text, then it is an **informative** summary. There are also **critical** summaries (reviews), that add personal opinions to the informative gist. By doing this, they add value that is not available from the source text [36];

iii **Expansiveness**: Expansiveness can be divided in two different types: **background** and **just-the-news**. In a case of a background type of summary, the reader does not know anything about the content of the input text; while in case of a just-the-news summary the reader already has some background knowledge about the theme, allowing him to completely understand the content.

In Table 2.2.3, we have all the factors previously presented in a condensed format.

Summary classifications		
Input factors	Number of documents	Single-document Multi-document
	Specificity	Domain-specific Genre-specific Independent
	Document language	Mono-lingual Multi-lingual Cross-lingual
	Genre/Size	Book size Articles Newspaper texts Reports Biblical texts
	Media	Text Images Video Speech Hypertext
Output factors	Derivation	Extractive Abstractive Semi-Extractive
	Coherence	Fluent Disfluent
	Partiality	Evaluative Neutral
	Conventionality	Fixed Floating
Purpose factors	Audience	Generic Query-oriented
	Usage	Indicative Informative Critical
	Expansiveness	Background Just-the-news

Table 2.1: Summary classifications according to the different factors

2.3 Automatic Summarization Approaches: Extractive and Abstractive

Text Summarization methods can be classified into extractive and abstractive.
Extractive summarization approaches focus on selecting vital material, such as sentences, paragraphs or passages from the original document, based on a set of rules defined by the user.
Along the years, several techniques for creating extractive summaries have been proposed: from classic approaches to machine learning approaches, among others like we will see in the following sections. In some of these approaches, the summarizers consider that the "most im-

portant" content is most of the times the "most frequent" or the "most favorably positioned" content. By doing this, they avoid efforts on understanding the context of words, their meaning, their purpose and so on. Therefore, do not require linguistic knowledge to select the most relevant aspects of the source text to create the summary [37] [38] [39].
In general, extractive summarizing systems consist of two main phases: pre-processing and processing. The first phase consists in creating a structural representation of the text for a better understanding by the computer.
Pre-processing the input text is usually formed by the following steps: [40] [41] [42] [43] [44]:

i **Removing punctuation characters;**

ii **Removing numbers and roman numerals;**

iii **Removing Hypertext Markup Language (HTML) tags;**

iv **Removing stop-words** - stop-words are the functional words that do not contribute to the meaning of a sentence. For example: "or", "once", "too", like others;

v **Removing white spaces;**

vi **Converting every word to lowercase;**

vii **Fix possible spelling errors;**

viii **Convert plural words to singular words;**

ix **Convert words to their base/canonical form** - using a *Stemming* or *Lemmatization* algorithm, like it was mentioned in the Section 2.3.1;

x **Apply POS tagging** - POS explains how a word is used in a sentence. It takes each word with its syntactical tag: noun, verb, etc;

xi **Apply Shallow Parsing** - extracting phrases from unstructured text and grouping them into "chunks". (It can be done after applying POS);

xii **Named Entity Recognition (NER)** - extracting important information from text that does not have a simple structure. This allows the classification of words into their respective categories, for example the word "Bill" belongs to the person category, and so on. NER allows a detailed knowledge about the text and how are the relations between the different entities;

xiii **Tokenization** - after the above steps have been completed, the tokenization step splits the resulting text into a list of tokens. This can be done by three ways: text into sentences, sentences into words and sentences using regular expressions tokenization, depending on the purpose of the summarizer. A very popular library for tokenization, and many other NLP tasks, is Natural Language Toolkit (NLTK) [45];

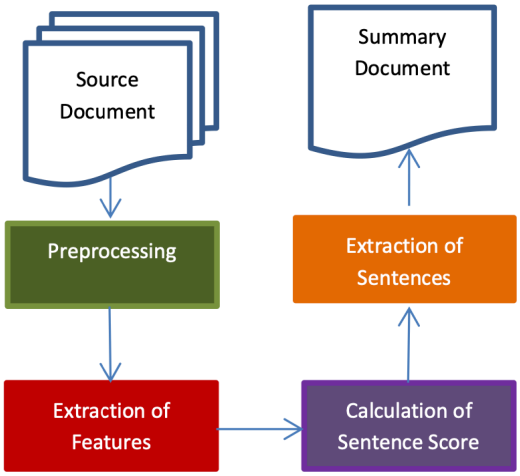

Figure 2.2: Extractive summarization steps by [2].

xiv **Creating Word Embeddings** - creating word embeddings is the process of converting words into vectors of numerical values, according to a vocabulary. Each word is represented as a dense vector with real values in a predefined vector space, often having tens or hundreds of dimensions, which is a contrast to the number of dimensions used to represent sparse words using a one-hot encoding[3]. The values from the word embedding vectors are learned, mainly through neural network techniques and are based on how the words are used. Words that are used in identical ways will have identical vector representations, naturally capturing their meaning [46] [47].

Using word embeddings represents a powerful tool that can be used for several tasks, such as text classification, sentiment analysis, and text summarization [48] [49].

Word embeddings can be created with the help of libraries like *Word2Vec* [50], *GloVe* [51], *ELMo elmo*, *BERT* [52], *XLNet* [53] or *fastText* [54].

Subsequently, in the processing phase the system must weight the sentences and choose the more relevant ones to combine them into a summary.

Abstractive summarization, as opposed to extractive summarization, do not extract information from the text, instead, they structure the document information so that it can be processed by advanced natural language generation techniques [16]. These techniques analyse the internal characteristics of the text such as: semantics to generate information in a new form [55]. Abstractive summarization creates a generalized summary,that has all the important information about the input text, but it is relatively costly,because it requires powerful machinery for the natural language processing and it is harder to use in other domains [16]. Since most of the research done in text summarization focus on extractive approaches, there are only a few abstractive approaches worth mentioning. Abstractive summarization approaches are divided in two main groups: structure based approaches and semantic based approaches. In the first group, we have methods such as: tree base method, template based method, ontology based method, lead and body phrase method, and rule based method;

[3]One-hot encoding represents words as sparse vectors of 0s and 1s, according to the vocabulary size.

whilst in the second group we have the following methods: Multimodal Semantic model, Information item based method, and semantic graph-based method.
Below we will present all the summarization approaches, starting with extractive approaches and finishing with abstractive approaches.

2.3.1 Classic Approaches

Classic approaches have come a long way, since the beginning of the research on automatic text summarization during the decades of 1950 and 1960, until the present days.
Lunh began the research on text summarization [56]. His research focused on fetching the most significative words from the text to create a summary. The process began by applying a simple pre-processing phase, where the unnecessary words and symbols were removed and the remaining ones are reduced to their canonical form[4].
This reduction is known as *stemming*, and was developed by names like Lovins, Dawson, Paice/Husk and Porter (the most known algorithm) [57] [58] [59]. Later, it helped develop *lemmatization* algorithms. The goal of both *stemming* and *lemmatization* is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.
A *stemming* algorithm obtains the stem of a word, i.e, it is morphological root, by clearing the affixes that carry grammatical or lexical information about the word [59].
In the other hand, a *lemmatization algorithm*, uses a vocabulary and the analysis of words to remove the endings of words and return the dictionary form of a word, which is known as *lemma* [60] [61] [62] [63].
After applying a *stemming* or a *lemmatization* algorithm, the words are ordered in a descending form , according to their frequencies in the text. After ordering the words, the relevance of each sentence is obtained by counting the number of relevant words and the distance of each word and their occurrences. The summary is then created with the ordering of sentences according to their relevance scores.

2.3.2 Feature Based Approaches

Classic automatic text summarization approaches were mainly driven by statistical calculations, such as: Term Frequency (TF), Document Frequency (DF), Inverse Document Frequency (IDF) and a combination of both (TF*IDF). They did not evaluate the properties of the sentences, which led to the creation of a new type of approaches, the feature-based approaches. These approaches analyse the features of both sentences and words individually, in order to rank them according to their relevance [64].
Baxendale proposed one of the first feature-based approaches, that consisted in evaluating the importance of sentences in the creation of the summary, according to their location in the text. After analysing several texts and their paragraphs, the author concluded that the first and last sentences of each paragraph are important for the summary [11].
Edmunson adapted the above method and added two new features: presence of indicative words and title words. Indicative words are words that convey information about the topic of

[4]Basic form of a word used as a dictionary entry.

the text, such as: “introduction“, “investigation“, “lecture“, while title words are words that appear on titles or headings [18] [12].
Every feature-base summarizer is constituted by three phases: feature identification, sentences score assignment and summary creation.
In the first phase, the summarizer analyses both word-level and sentence-level features. The first are relative to the words while the latter are related to the sentence [39]. Below we have the word-level features: [65] [66] [64] [2] [67] [68] [34]:

i **Content Words** - content words (keywords) are nouns, verbs, adjectives and adverbs. Sentences that have a bigger quantity of these words, are most likely to be included in the final summary;

ii **Title Words** - sentences that have words that are mentioned in the title, have greater probability of going to the final summary, because they indicate the theme of document;

iii **Cue words** - cue words illustrate the structure of the document flow and are words like: “because”, “develop” or “summary”. Sentences that have these words, will most likely be in the summary;

iv **Positive words** - positive words are words that frequently occur in a summary. The more frequent they are, the more important they are;

v **Negative words** - in the contrary, negative words are words that never occur in a summary and should not be used to create a summary;

vi **Biased words** - biased words are words that are predefined in a domain. They also help describe the theme of the document;

vii **Word co-occurrence** - words that co-occur in the different sentences in the same manner position are considered important;

viii **Upper case words** - words in uppercase are, in most cases ,important words. Take for example the word “UNICEF”;

ix **TF*IDF** - Term Frequency Inverse Document Frequency (TFIDF) evaluates the importance of a term in a document. It combines the calculation of two formulas: TF and IDF. TF shows how frequent is a certain term in the document. This value is higher if the term appears more in the document, but it does not mean that the term is of great importance. TF gives equal importance to every single word in the document, even though there are terms more important then others. IDF actually tells how important a term is, because it allows us to decrease the weights of frequent terms increase the weights of true rare terms;

x **InfoTermo** - as opposed to the TFIDF metric, the InfoTermo metric can inform us about importance of a term in a certain part of the text. It was proposed by our teacher, João Paulo Cordeiro.

Likewise we have the sentence-level features [66] [64] [2] [69]:

i **Sentence location** - correctly written documents are structured hierarchically, so sentences that appear in the beginning or end of the documents are more important;

ii **Sentence similarity** - if there is a vocabulary overlap in two sentences, they are considered similar therefore important. Similarly, one can calculate the sentence similarity with lexical analysis;

iii **Sentence length** - the length of sentences is a key feature, because sentences with shorter length does not usually constitute essential information neither very long sentences. What is usually done is normalizing the length of the sentences to utilize the most important parts of the sentences in the summary;

iv **Numerical value in sentence** - sentences that have numbers or roman numerals convey more importance;

v **Paragraph location** - likewise sentence location feature, paragraphs that appear in the beginning or end are more important;

vi **Number of nouns and verbs in sentence** - sentences that have more nouns and verbs are important;

vii **TF*ISF** - an alternative to the TFIDF feature is Term Frequency * Inverse Sentence Frequency (TFISF). it is very used in information retrieval tasks. In this feature, we do not focus on how frequent is the term in the document, instead we focus on how frequent is the term in a sentence. Therefore, this measure removes the impact of higher frequency terms which are not useful for the summary;

viii **Average-TS*ISF** - after calculating the value for TFISF, we can then calculate the value of the Average-TS-ISF attribute for a certain sentence, which is the the average value of the TF-ISF values for all the terms in the sentence;

ix **Word similarity among paragraphs** - words from a certain paragraph that occur more frequently in sentences from other paragraphs, are considered important;

x **Word similarity among sentences** - a sentence whose words occur more in other sentences then the sentence itself, should be considered important;

xi **Named entities** - sentences that have named entities are more important;

xii **Temporal values in sentence** - sentences that have event containing date or time expressions should be added to the summary;

xiii **Latent Semantic Analysis (LSA) scores** - a complex sentence type feature consists in decomposing a word-sentence matrix with Singular Value Decomposition (SVD) and getting two important things: the hidden topics of the document and the projects of each sentence on each topic. With these projections, we can then assign scores to sentences and select the top scoring sentences for the summary [70].

In the second phase, the summarizer assigns a value to each sentence. A higher value is assigned to those words and sentences that better represent the topics of the text.
In the last phase, the summarizer creates the final summary by selecting the *n* most important sentences.
Feature based approaches can be combined with other summarization approaches, whether they are of extractive or abstractive nature, to create a more efficient summarizer. One example is combining them with logistic regression[5].
By doing so, after the system extracts the features, it can calculate their weights, thus discovering which features are the most important for the summarization task. The features of manually written summaries are used as input to the model and the corresponding dependent variables are used as output in the training phase. Then the model figures out the relations between inputs and outputs to create an efficient model, that will receive new data in the testing phase.

2.3.3 Machine Learning Approaches

The research of machine learning approaches in NLP began in the 1990's, with the appearance of publications that used statistical techniques to create extracts from documents. At that time, most systems used algorithms that relied on the Naive-Bayes method [16]. Eventually many other systems tried to implement their own machine learning models to create extractive summaries out of text.
Machine learning based approaches can be divided in three classes: **supervised learning**, **un-supervised learning** and **semi-supervised learning** [71] [72].

i **Supervised learning approaches**: are general machine learning methods that analyse the training data (input data points and respective output) in order to classify data and compute predictions on unknown data [55]. In the case of text summarization, they label words, sentences or passages from a document in the training phase, with the use of a set of documents and respective human-made summaries, to find features or parameters to summarize the document [73]. Each sentence is classified as relevant or non-relevant for the summary. After this training step, they apply the same process on unlabeled text of a new and unknown document. The sentences of this new document are ranked according to their relevance for the summarization task [74] [75].

   Supervised approaches often perform better, but they have two big problems: the first one is to assume that human-made summaries are sufficient to be used in automatic summarization as "gold standards"; and the second problem is the portability of a supervised learning text summarizer, because deploying it in a new domain will require a new collection of data, that in turn needs to be manually annotated to be used in the training process. These two problems are very time consuming and costly [71] [73]. Examples of supervised learning approaches are: Neural Network Based approaches, Conditional Random Fields and machine learning approaches based on Bayes Rule [39].

[5]Logistic regression is a predictive statistical approach for modelling the relationship between a dependent variable and a set of independent variables.

ii **Unsupervised approaches**, on the contrary, find ways to classify the text without regard to external information, which means they can be applied to any text data without requiring any manual effort [72] [73] [55]. This means they don't have to deal with the two problems stated above.

Unsupervised learning can be divided in two kinds of methods: *clustering* and *topic modeling*.

In **clustering** based approaches, the documents are segmented into partitions, where each partition corresponds to a cluster. The problem is that sometimes it is hard to assign a document to a certain cluster.

As for **topic modeling**, each document is a probability combination of topics, with each topic being a probability distribution over words, where the representative words have the highest probability. Topic models are quite similar to clustering in general, because a topic can be analogous to a cluster and the condition of a document belonging to a cluster is a probabilistic value [55].

General unsupervised learning approaches are: graph based approaches, concept based approaches, fuzzy logic based approaches, LSA approaches and clustering based approaches [39].

As for examples of their usage in text summarization, there are a couple of scientific papers published, such as [76] [77] [78].

iii **Semi-supervised approaches** are a combination of several characteristics from both supervised and unsupervised learning approaches. They were created to reduce the time and costs of labeling data for supervised learning. An example of success was created by [71], that reduced the labeling time by 50%.

Approaches based on graphs represent the texts as graphs, where each text unit is a node of a graph. A graph is like a network linking sentences and graph-based ranking methods are used to generate a summary
Finally, there are also the approaches based on latent semantic analysis that identify the semantic relationships between sentences.

2.3.4 Neural Network Based Approaches

Neural networks are a type of supervised learning approaches that try to mimic the behaviour of the human brain. They are constituted of artificial neurons that are connected with each other in layers, much like the human brain, with the purpose of processing data. The input layer, which is the initial layer, consists of the neurons that receive the input data, then they are connected to the hidden layer(s) that perform the desired calculations, and in its turn they are connected to the output layer that is used to return the output of the calculations, i.e, a prediction [79]. A neural network model is created and then trained with an initial data set. After this, if the success rate of the predictions made by the model is acceptable, the model can be used to predict new data. The data can be several things, like images, sound, text, etc.

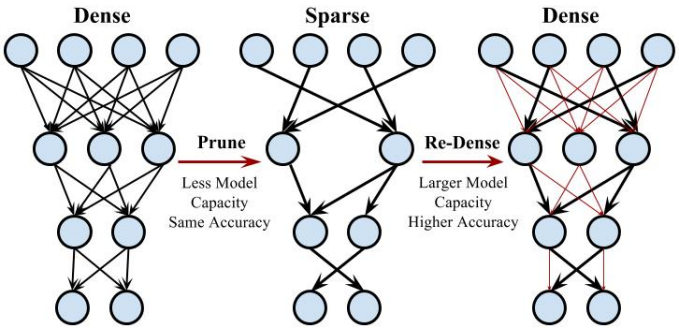

Figure 2.3: Neural Network phases by Khosrow Kaikhah [3].

With regard to natural language processing, their applicability is huge, because they help classify the textual data and come up with outputs that can be used for tasks like sentiment analysis, text summarization or part-of-speech tagging.
In the case of text summarization, a neural network can be used to determine which sentences are used to create a summary, by being constituted of three phases: training, feature fusion and sentence selection respectively. In the first phase, the neural network is fed with a corpus of documents and it decides whether the sentences are added in the summary or not. The network trains itself and learns certain patterns about the sentences, according to the text features defined by the specialist, such as: paragraph location in document, sentence location, sentence Length, etc.
After the training phase comes the feature fusion phase, where the features are analysed and it is decided which features are deleted and which are kept (some neurons and respective synapses/edges are deleted). This process reduces the size of the artificial neural network model, turning it into a much more memory and power-efficient model [80].
Finally, in the third phase the neural network selects which sentences are used to create the summary, according to their ranks [2] [3].

i **Netsum** was created by [81]. It consists of a neural network that labels the training data and then does the extraction of features from the sentences in the document, using the RankNet algorithm to assign scores to the sentences.

   The RankNet algorithm was developed using neural nets by Chris Burges and his colleagues at Microsoft Research. It is an algorithm used to rank a set of inputs, in our case, the set of sentences in the text document. RankNet optimizes the cost function using Stochastic Gradient Descent.

ii **Khosrow Kaikhahs approaches** is a Feed Forward Neural Network (FFNN) with three layers (one input, one hidden and one output [82]) that analyses the characteristics from the original text, splitting them in summary and non-summary sentence characteristics [3].

   In Figure 2.4, we have the representation of the neural network according with all the phases combined.

2.3.5 Conditional Random Fields

Conditional Random Fields (CRF) is a probabilistic and machine learning approach (supervised learning) used in sequential prediction problems, in areas like natural language processing, computer vision and bio-informatics, for tasks like named entity recognition, POS, gene prediction, noise reduction and object detection problems, text segmentation, to name a few [83] [84] [85].
A system using CRF was proposed in Hong Kong University of Science and Technology, by [70] that extracts the features from the text in order to determine the important sentences of a certain document. Using this method allows the identification of the right features thus providing a better representation of the sentences and terms into the respective segments in the model. The problem with this approach is that it needs a specific corpus to be used in the training process. If it does not have a domain corpus, it can not be used generically to any document, which takes a lot of time to be done.

2.3.6 Latent Semantic Analysis

LSA is an unsupervised learning approach that deals with the similarity among sentences and among words, by extracting the hidden semantic structures of these text units [4] [86] [39].
[87] define LSA as a statistical and algebraic technique that extracts and represents the context of words as groups. If a certain word appears in a group of word contexts, implies that the word has similarities with the rest of the group words. While according [88], LSA is an important technique that it is capable of finding out the semantic relations between words, in a corpus. After a statistical analysis, one can identify how close two words are, in terms of synonymy and polysemy.
LSA being an unsupervised approach does not need any training sets of documents and respective human-made summaries. The main goal of this approach is to discover the words that are used together and what common words are used in different sentences. Having a bigger number of common words in different sentences, means that these sentences are related semantically.
This technique is very useful for various kinds of NLP tasks. In 2002, Yihong Gong and Xin Liu proposed that it could be used for automatic text summarization [89], according to two steps:

i **Matrix creation** - the text is represented as a term-sentence matrix, where each column represents a weighted term-frequency vector for each sentence in the respective document. For a document d, m and n stands for the terms and sentences of that document respectively. Since every word does not normally appear in each sentence, the matrix is sparse[6].

ii **SVD application** - by applying the SVD to the matrix, the relations between sentences and words are found. This is a powerful technique used with vector spaces and in this case, it improves accuracy by reducing unnecessary terms in the text [4] [90] [89].

[6] A matrix in which most of the elements are zero.

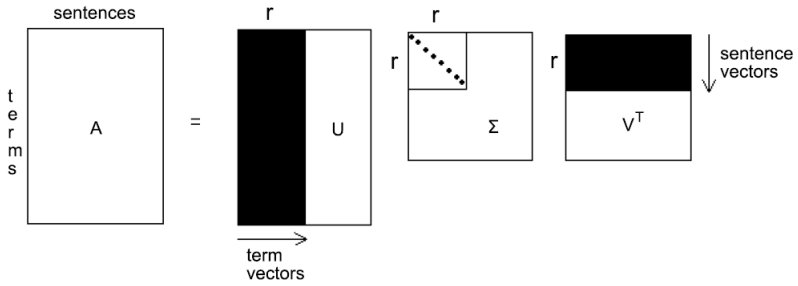

Figure 2.4: Singular Value Decomposition applied to a matrix of *m* terms and *n* sentences [4].

The technique proposed by Gong and Liu was very important, for both single and multi-document text summarization, because it did not require the use of WordNet to identify the most important topics in a document[89] [55]. WordNet is an online lexical database that stores English words, such as: nouns, verbs, adjectives and adverbs, organized into groups of cognitive synonyms, much like a thesaurus. These synonyms are connected to each other by semantic and lexical relations [91] [92]. It is used in many NLP tasks.

2.3.7 Fuzzy Logic Based Approaches

Fuzzy logic consists in a mathematical term that is very useful in dealing with uncertain and imprecise scenarios, mostly in engineering and computer sciences. Fuzzy logic, as opposed to binary logic, can have partial truth values, i.e. the range of truth values can go from 0 to 1, while in binary logic the truth values can only be 0 or 1. This is useful with scenarios where the truth values are between completely true and completely false.
The main characteristics of fuzzy logic are: the ease of use (because of it's concepts and it's basis on natural language), the tolerance to imprecise data, as mentioned above and it can model non linear functions, which makes it a flexible tool for any imprecise scenario [93].
The architecture of a fuzzy logic system is essentially constituted of: a fuzzifier, a rule base, an inference engine and a defuzzifier.
Fuzzy logic is very useful for text summarization because it solves the problems of uncertainty when choosing the right features and sentences in documents.
Most of the approaches for text summarization that are based on fuzzy logic, can be divided in the following phases: pre-processing of the document, feature extraction, fuzzification, defuzzification and generation of the summary [93]
After all the features, such as: title word, term weight, sentence length, sentence position, etc [93], are extracted, the fuzzy logic system begins its work. In this moment, the definition of the fuzzy rules and membership function occurs. A membership function is a simple function that represents the degree of truth in fuzzy logic, i.e. says if the elements in the fuzzy sets are discrete or continuous [94]. Several membership functions can be used, such as: Triangular membership functions [95] [5] or Gaussian membership functions [69].
The value for each sentence is derived using a fuzzy logic method, that combines the fuzzy rules with the membership function. All the fuzzy rules have a format such as *IF-THEN*. What the membership does is fuzzifiying the sentences scores into one of three values: *LOW, MEDIUM, HIGH* that concludes if the sentence is not important, average or important (de-

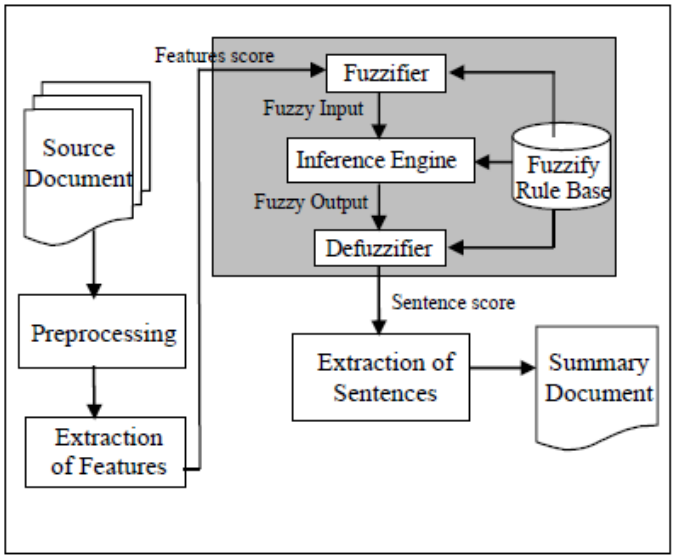

Figure 2.5: Fuzzy logic system for text summarization [5].

pending of the value that it has).
The final part consists of choosing the sentences, according to the output values obtained before. The higher the output value of a sentence is, the more important it is for the summary [5] [69] [93].

2.3.8 Graph Based Approaches

Graph based approaches are one of the many unsupervised learning approaches used in automatic text summarization and one of the most searched methods.
Like the name suggests, they use graphs to represent a document in a set of nodes and the connections between them. Each node of the graph represents a piece of text and the edges represent the relations between those pieces. These edges have a weight, that represents how close are the pairs of nodes, in terms of similarity or lexical and semantical relations [96] [30]. These connection weights are used by the ranking algorithms to decide the importance of a certain node. By ranking each nodes according to their importance, one can decide if a certain node will be used to create the summary.
The first ever graph-based approach was proposed by [97]. In his approach, the text was represented by a group of connections between paragraphs. Much like many other graph approaches, the nodes consisted of pieces of text, in this case: whole paragraphs, and the affiliations between them were stored in the edges. The output summary was created by the identification of the most important paragraphs. To figure out which paragraphs were the most important, the summarizer counted the number of edges of each node. The more edges a certain node had, the more important it was.
[97]'s work was one of a kind and it led to an increase in the research of text summarization. It inspired engineers like [98] to go forward in the research of text summarization with graphs. They were able to create a very well known summarizer, named **LexRank** that consists of a stochastic graph-based method that calculates the importance of textual units for NLP tasks.

It's main focus was on the text summarization task, regarding sentence salience. Salience is defined as: "the presence of particular important words" or "similarity to a centroid pseudo-sentence" [99]. LexRank calculates the importance of sentences based on *eigenvectors*, i.e. characteristic vectors, in a graph of sentences with the use of a matrix based on intra-sentence cosine similarity. Cosine similarity is used to evaluate how similar are two non-zero vectors (sentences), by measuring the cosine of the angle between them [100].
Another graph-based approach for multi-document text summarization was proposed by [101] in the 90's. The importance of the nodes in a text graph is calculated with the use of a search algorithm based on the concept of "activation propagation". This algorithm is used after representing the graph with relations such as: adjacency, co-reference, synonyms between nodes.
A few years later, the research on graph-based ranking algorithms increased, originating algorithms such as *Hyperlink-Induced Topic Search (HITS)*, *PageRank* and *Positional Power Function*. Some of these algorithms were then applied to NLP tasks, which is the case of the *PageRank* algorithm as we will see below.

i ***HITS*** - is a graph-based ranking algorithm created by Jon Kleinberg, to evaluate web pages and the connections between them. Each page has a pair of values: the first is the *authority* value, which is the value of the content of the page in terms of incoming links; the second is the *hub* value, that is the value of the connection between that page and the rest (outgoing links) [102]. After searching for a web page, the algorithm calculates both *authority* and *hub* values for that page. This is done for only a certain page, and not all of the ones that related to each other, in order to improve efficiency [30].

ii ***PageRank*** - is a graph-based ranking algorithm that was created in 1998 by Brin and Page with the purpose of analysing the connections between web pages [102]. Each connection between pages have a numerical weight that represents the relative importance within a set of connections. When one page links to another one, it is casting a vote for that other page. The higher the number of votes that belong to a page, the more important that page is [103]. If a page has a lot of pages with high ranks connected to it, then it will also have a high rank. In the opposite way, if there are no links to a web page then the rank for that page will be the lowest possible [104]. It is considered one of the most popular ranking algorithms and it is even used by the Google Internet search engine[30]. A common problem is the existence of spam pages, that are created with the purpose of manipulating search engine indexes. When these pages use the PageRank algorithm, along side Google, they use spam to increase their ranking [105].

iii **Positional Power Function** - much like the algorithms presented above, the positional power function is a ranking algorithm that is used to assign scores to nodes in digraphs. It combines the numbers of successors and the scores of sucessors of a certain node [106] [102].

The ***PageRank*** algorithm listed above, inspired the researchers in the field of automatic text summarization to use graph-based algorithms - originating new algorithms, such as: ***TextRank*** and ***LexRank***.

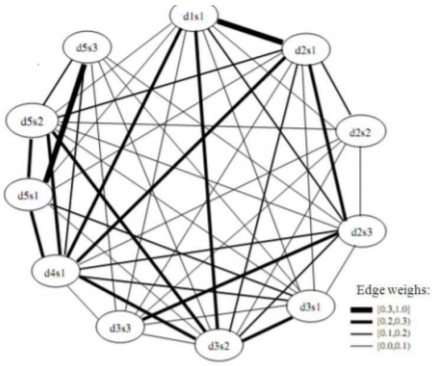

Figure 2.6: Example of weighted cosine similarity graph by [6].

In the ***TextRank*** algorithm, each node does not represent a web page, instead it represents a sentence of a document. The same way, instead of calculating the probability of transition between pages, the algorithm calculates the similarity between sentences. Much like the *PageRank* algorithm, the scores of the similarities are stored in a square matrix.
Before applying the *TextRank* algorithm to natural language texts, one must first build a graph to represent the text and creating the relations between sentences. Each node can represent a single word or words, collocations or even entire sentences. Depending on what each node represents, the type of relations that exist between nodes may vary. Examples of relations can be: lexical, semantic, contextual, etc [102] [103].
In general, the steps of applying the *TextRank* algorithm to natural language texts are [103] [30]:

i Creating the nodes of the graph, that best represent the desired text;

ii Identify the relations that connect the nodes and then create the edges of those relations;

iii Iterate the algorithm until convergence is reached;

iv Sort vertices based on their final score. The scores are used for taking decisions for the task.

The *TextRank* algorithm is very useful for extraction tasks, such as keyword or sentence extraction. This is important for our subject of automatic text summarization, because through this algorithm we can fetch the most important keyword or sentences from a document and then create a summary.

2.3.9 Clustering Based approaches

Clustering is an unsupervised learning technique that attempts to create groups of entities that have similar properties. In text summarization, it can be used to create groups, i.e,. clusters, of sentences that relate to each other [107] [55]. In order to discover how the sentences relate to each other, one must analyse the similarity between them. A very common way to do this is using the cosine similarity measure[7] [108]. Sentences that are very similar to each

[7]Cosine similarity is a measure of similarity between two non-zero vectors that measures the cosine of the angle between them.

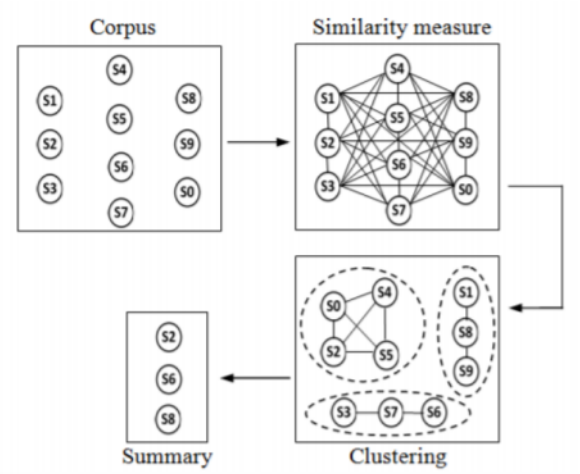

Figure 2.7: Text summarization of a corpus with a clustering approach [7].

other are saved in one cluster.
Clustering can be used for both single and multi-document summarization.
In one hand, if the input consists of only one document, the clustering mechanism will evaluate the similarity between sentences and create the respective clusters.
In the other hand, if the input consists of more than one document, then the mechanism will evaluate each one of the documents and discover if they have common topics. If this happens, they are put together in the same cluster. Finally, the summarizer chooses the sentences that are related, from the clusters and combines them into a single summary.
In Figure 2.7 we have a visual representation of this process.
The first ever implementation of a clustering algorithm for text summarization was proposed by Radev [109]. *MEAD* consists in a multi-document summarizer that uses a centroid-based summarization technique. The process is divided in three phases: feature extraction, sentence scorer and sentence reranker. In feature extraction, the features of each sentence are calculated. The sentences are then used in the second phase, where the sentence scorer assigns a value to each sentence, according to the linear combination of their features. In the subsequent phase, the sentences are ordered by their scores and they are added to the summary, where the first added sentence is the one with the highest score. The next steps consists on the re-ranker calculating the similarity of the sentence that will be added with all the sentences that already are in the summary. If the similarity is above a given threshold, the sentence is not added to the summary and the re-ranker moves on to the next sentence. This process lasts until the summarizer reaches a desired compression rate.
Following the work done by Radev and his partners, Jaime Carbonell and Jade Goldstein combined their efforts and developed the Maximal Marginal Relevance (MMR) algorithm. This algorithm maximizes the *marginal relevance* of sentences and documents. This concept is a metric that evaluates how a document is relevant for a certain topic and how it is similar to documents that were analysed before. It consists in an important metric because it reduces the redundancy between sentences of documents. In general, the higher the MMR value of a document, the more relevant it is to a topic and the less it is similar to another document [110] [107].

2.3.10 Discourse Approaches

Discourse approaches attempt to analyse semantics, by finding the discursive relations in the text discourse, i.e. the relations between text units, with the purpose of creating a summary. The first ever discourse based summarizer was proposed by Radev, giving it the name: Cross-Document Structure Theory (CDST). This system evaluates the semantic relations between words, phrases or sentences, in the same or different documents, if they are indeed connected in a semantic way. The main problem is that these relations should be explicitly determined by humans [111] [112] [113].

2.3.11 Bayesian Topic Models

Bayesian topic models represent a probabilistic approach that can be very useful in determining the text topics. They succeed where other approaches fail, in terms of topic identification. The main goal is to identify words that relate to a particular topic and the topics that are present in a particular document, with the analysis of a set (corpus) of documents. The core of these approaches is the usage of Bayesian inference to calculate how probable is a given sentence included in the extract. Newer extracts can then be created with the ranking of sentences according to the value of this probability and picking a certain number of the top scoring sentences [114] [2]. Bayesian topic models extractive summarizers have been proposed by [115] [116].

2.3.12 Hidden Markov Models

Hidden Markov Models Hidden Markov Models (HMM) are based on Markov Chains. These chains are probabilistic models that can predict data about sequences of random states (variables), taking into account that these states can take values from a set and there are transition probabilities between states [117].
Markov Chains are quite powerful in predicting a sequence of observable events, but they lack in potential when we are interested in predicting hidden events, i.e., events that are not observed directly. This is where HMM become important: they can be used for both observed and hidden events [118].
HMM have been successfully used for NLP tasks such as POS tagging and NER. In POS tagging, given a set of words as a sequence, the HMM model can predict the respective POS tags for the words.
In NER, the HMM models figures out the dependencies between words by giving each word the respective name entity type. In terms of architecture, the model admits states that are organized into regions, one region for each name entity type. Then, for each region we use a statistical bi-gram language model to calculate the probability of the words within that region, i.e. the named entity type. [119] [55]
The first work for using HMM for text summarization was proposed by [120]. They computed three sentence features: position of a sentence in the document, number of terms in a sentence and how likely are the terms of a sentence. With these features, the model computes the value of the probability of each text sentence being a summary sentence. Given a

certain sentence as a sequence for the model, the probability that the next sentence is indeed a summary sentence, depends from the present sentence. This is done for all the sentences in the text and, step by step, the sentences are chosen for the summary creation.
[121] suggested another approach: an extractive summarizer that combines clustering techniques with Hidden Markov Models. They used a modified version of the well known K-means [122] clustering method with HMM, to analyse the text in terms of cohesion and assign tags to the sentences. The system can then pick salient sentences for summary creation or simply detect topics in the text.

2.3.13 Lexical Chain Approaches

Lexical Chains are extractive summarization techniques that analyse the lexical cohesion between words of a certain corpus.
Lexical cohesion was defined by Halliday and Hasan, in 1976, as a process of combining different parts of the text through the semantical relations the terms have with each other. This process does not only occur between pairs of words, but also between sequences of words that have some sort of relation (i.e. lexical chains) [123] [124].
Halliday and Hasan divided lexical cohesion into two types: reiteration category and collocation category. The first one can be done by analysing the repetitive words or synonyms and hyponyms. The second one is achieved by analysing the words that co-occur in the same lexical context [125].
Lexical chains can be used for several NLP tasks, such as information retrieval, topic tracking, text summarization, etc.
The first ever computational implementation of lexical chains was presented by Morri and Hirst, in 1991. They used the Peter Mark Roget Thesaurus (Roget Thesaurus [126]) and the WordNet database to determine the relations between words. The importance of the WordNet database is that it contains words such as: nouns, verbs, adjectives and adverbs in english, organized in sets of synonyms. These sets are then related with synonymy and hyponymy [127].
A few years after the work done by Morris and Hirst, some other approaches have been presented, by the names of Hirst and St-Onge, in 1997, and by Barzilay and Elhadad, also in 1997. They both used WordNet as a knowledge database to figure out the semantic relations between words. The difference was that the first approach computed a chain that could include a word if that word was first found, while the latter approach computed all the possible chains for a certain word and then defined the best interpretation [128].
The great advantage of using lexical chains for text summarization is that they do not need the full understanding of the text, but only some available knowledge sources [125], for instance WordNet, like was mentioned above.

2.3.14 Structure Based Approaches

Structure based approaches consist on mechanisms that focus on the vital data from text documents and encode them, using structures based on: tree, template, lead and body, rule,

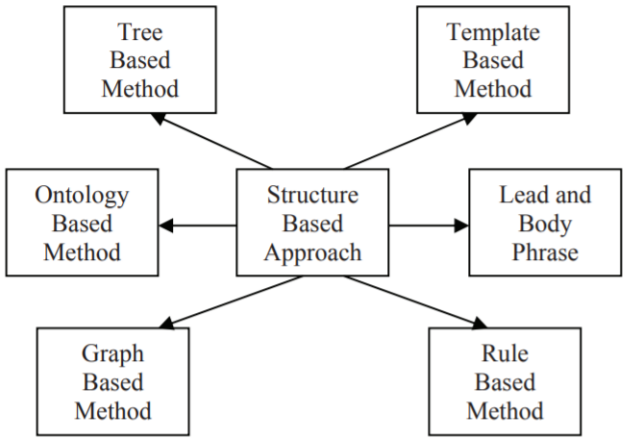

Figure 2.8: Six structure based approaches by [8].

graph and ontology [8].
The Diagram 2.8 consists of an overview of this structure.

1. **Tree based approaches** - These approaches represent the text data using a dependency tree. The tree is used together with a language generator or associate degree to create the summary.

   An abstractive summarization approach was proposed by Regina Barzilay and McKeown [129]. The most common phrases are identified with the help of a bottom-up multi-sequence alignment. The input consists of various documents whose theme is identified with a process of theme selection. After the theme has been identified, the sentences are ordered with a clustering algorithm and then fused, generating the summary;

2. **Template based approaches** - Template based approaches use a template to represent the entire text document. According to extraction rules or linguistic patterns some text units are mapped to the model and in the future are used as indicators of the summary content.

   Sanda M. Harabagiu and F.Lacatusu[130] proposed an abstractive summarizer using this approach, called *GISTEXTER*. It creates summaries with the use of information retrieval, that focuses on the identification of topic-related information in the text documents. Each topic is represented as a set of concepts that are implemented as a model and are stored in a database. The summarizer then chooses sentences from the database to create the summary. This summarizer is used for both extractive and abstractive summarization tasks;

3. **Lead and Body based approaches** - lead and body based approaches consist on inserting and changing syntactically similar sentences with the sentences from the lead or body. Lead sentences are sentences that begin an important part of the text, such as: a chapter, an article or a paragraph, while body sentences are sentences that belong to the middle of the text. Thus, the goal of this approach is to insert and rewrite the lead

sentences, by analysing the syntactical resemblance between these sentences and the body sentences.

An abstractive system for summarizing broadcast news was proposed by Tanaka et al. [131]. The system analysed the lead and body sentences and identified sentences that resemble these sentences. Then it inserted and switched phrases to generate the summary;

4. **Rule based approaches** - rule based approaches use terms of classes and lists of aspects to choose the documents that the system summarizes. To evaluate if a document is chosen, the document must answer one aspect of a certain group of extraction rules. In the end, Only the best candidates are chosen, i.e. the ones that answer the most number of rules. In the end, the system uses generation patterns for generating the summary sentences.

   Pierre-Etienne and G.Lapalme proposed a system that uses information extraction to identify semantically related words, such as nouns and verbs. After this related words are found, they are extracted and used to generate the summary [132].

   Another approach was proposed by Huong Thanh Le and T.M.Le[133], that consists in an abstractive summarizer that uses discourse rules, syntactical constraints and a word graph. The generation of a sentence is divided in two parts: the first is the finishing of the start of a sentence and the second is the finishing of the tip of a sentence. The sentences are then combined by observing and adhering to few syntactical cases;

5. **Graph based approaches** - abstractive graph-based approaches use a graph structure, to represent the text. Each of the graphs node represents a word unit, that in turn represents the structure of sentences [8].

   An opinosis-graph based approach was proposed by Kavita Ganesan et al. [134]. The graph does not require any knowledge of the domain, therefore it is highly flexible and is able to balance the order that words appear in the text and its inherent redundancies. By doing this, it is able summarize highly redundant content and be used with many languages. The difference between an opinosis-graph and the graphs used in extractive summarization approaches, such as LexRank and TextRank is that the graphs used in these approaches are often undirected and each node represent a single sentence and the edges represent the the similarity between sentences, while in a opinosis graph each node represents a word unit and the edges are directed, representing the structure of sentences. The graph is initially created to represent the structure of the text and then each one of the paths and sub-paths are analysed to generate candidate abstractive summaries [134].

   Similarly, a graph-based approach to summarize extremely redundant sentences have been proposed by Kavita Ganesan et al. [135]. It uses the opinosis graph to search for sub-graphs that encode valid sentences and that posses high redundancy scores. The system assigns scores to sentences and paths to choose valid paths, according to the score. The paths are then ranked in a descending order of the scores and the duplicated paths are eliminated. Finally, the paths are used to create the summary sentences.

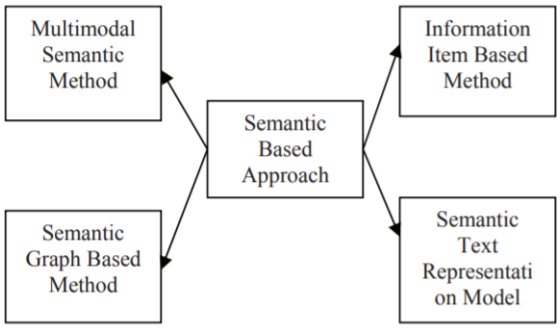

Figure 2.9: Abstractive semantic based approaches by [8].

Dingding Wang and T.Li proposed the creation of multi-document summarization systems that could use existing methods such as centroid-based method, graph-based method, among others in order to anaylse the outcomes and results of different baseline combination methods such as average score, to create a summarizer that could improve the performance of the summarization [136]

6. **Ontology based approaches** - Ontology based approaches evaluate the knowledge structure of a certain domain, that includes documents whose knowledge relates to a specific topic. Researchers have utilized the background knowledge (i.e., ontology) to improve their summarization results. Using ontology and combining it with domain–related information, a summarizer can determine the hidden semantic information of texts. This means that, textual information can be related with each other, using the shared and common understanding of a domain [137] [138].

   Ontology can be useful for domain specific documents where key concepts pertaining to the domain can be identified. This means that these approaches needs to have a pre-defined ontology, made by experts, in order to understand the documents domain.

   An Ontology Multi-document Summarizer was proposed by [139]. It is a summarizer that links the sentences from documents with a domain-specific ontology. Then queries the data from the ontology and extracts the summary from the sentences [137].

   Similarly, an Ontology Abstractive Summarizer was proposed by [138]. This system uses the world's largest ontology with an inference engine to create abstractive semantic summaries. It is an unsupervised and domain independent, which means it does not need to receive additional input to understand the text and it is only limited by ontology in itself.

2.3.15 Semantic Based Approaches

Semantic based approaches use linguistics, namely semantic, to analyse the noun and verb phrases in text and create abstractive summaries [8] [140].
In Figure 2.9 we indicate the four semantic-based approaches for abstractive summarization, by [8].

i **Multimodal semantic approaches** - these approaches use a semantic model to rep-

resent the text content that is used for multimodel documents, i.e. documents that have both text and images. The most important text concepts are rated according to some measure. The chosen concepts are used to create the summary, by helping in the selection of the sentences that represent these concepts.

Albert Gatt and E. Reiter [141] proposed an Natural Language Generation (NLG) system that is capable of summarizing great amount of numeric and symbolic data, by controlling how the phrases are built and combined into the summary.

ii **Information-item based approaches** - in this type of approaches, the abstractive summaries are not created with the sentences in the documents, but with an abstract representation of the documents. This abstract representation consists in a small, but very important part that contains the smallest and informative element in the text.

Pierre-Etienne Genest and G. Lapalme [142] proposed a system using this methodology, where the summarization process begins with an information item (*INIT*), that represents the smallest coherent element in a text. In the selection phase, the sentences that lead to a list of *INIT*, will be selected to be part of the summary. Instead of just using sentence selection, one can could use frequency based models to *INIT* selection. After selection, the summary generation phase begins. In this phase the generated sentences are ranked and a number of those that posses excessive size of the summary are the first ones selected.

iii **Semantic Text Representation Model** - in this technique, the semantics of the words are more important than the structure of the text.

Khan Atif and Y.J. Kumar [143] proposed a multi-document abstractive summarizer using this technique. The content is chosen by assigning ranks to the most significant predicate argument structures. Since the system assumes that the text is anaphora correct and does not have ambiguous parts, it can not process more detailed semantics to create the summary, that is generated with a specific tool.

Khan Atif et al. [144] also proposed an additional method, where the documents are divided into sentences, which in turn have their respective document and position numbers. By doing this, they were able to, like the previous system, extract predicate argument structures. They used a semantic graph to assign semantic similarity scores to the sentences. Finally, they used a graph based ranking algorithm to determine predicate structure, semantic similarity and document set relationship to create the summary.

iv **Semantic Graph Based Method** - in this method, the text is processed and represented as a linguistic graph, Rich Semantic Graph (RSG), that reduces the quantity of linguistic information and creates the final summary.

Lloret.E et al. [145] created a system that is able to construct concept-level summaries. Each input text receives lexical analysis[8] and is transformed into its respective syntactic representation. A generation tool, specific of the desired language, is used to create the

[8] Conversion of a sequence of characters into a sequence of tokens.

summary having the lexical units as input. Much like the system proposed by [143], this system does not have a semantic representation of the text and it assumes that all the sentences are anaphorally correct.

2.3.16 Issues of Extractive and Abstractive Summarizers

Even though automatic text summarization has had a huge research along the years, empowering humans with the ability to pass through the barrier of immense raw textual information, the tools that generate the summaries are still not fully efficient, because they possess problems in creating a coherent summary that represents the major content of the original text.
According to Sparck Jones, a summary is created from an input text, originating an output text [19], [1]. This output text much have a certain degree of textuality, which sometimes fails to be achieved in automatic text summarization , [146].
Textuality defines how communicative is a text, i.e. how is the connection between the author of the text and the reader/s. This connection is represented as trade of meaning from the author to the reader. It's easy to see that if a text is not communicative, it will not provide the reader with its essential information [146].
Textuality is divided in seven standards, such as: cohesion, coherence, intentionality, acceptability, informativity, situationality, and intertextuality. If any of these standards fails to be achieved, the text is not considered communicative, thus not an actual text (non-text) [146] [147] [148].
In the case of extractive summarizers, they might lose some of the essential information about the text, because they simply extract pieces of text.
As for the abstractive summarizers, they do not choose sentences from the text, instead they create their own interpretations, by compressing or re-generating new sentences. This is done through eight types of operations [9]:

i Reduction of sentences;

ii Combination of sentences;

iii Syntactic transformation;

iv Lexical paraphrasing;

v Generalization and specification;

vi Reordering.

Below are the main issues that extractive summarizers face [36] [149] [37]:

i **Redundancy** - The redundancy in the text is originated when the summarizer needs to process certain similar and salient pieces of text, that in the case of probabilistic approaches, possess the same score, which in the future will be added to the output. This also occurs with summarization of multiple documents that have the same topics;

ii **Incoherence** - The incoherence issue comes from some parts (or even the totality) of the text having incorrect anaphoric references. These references are used to define the theme of the text, and are related to words, such as pronouns, that reference subjects/entities of the text. If these pronouns are misrepresented, they do not reference the respective entity and the summary becomes messy. Incoherence can also come from sentences that are overlapped, leading to an incorrect text interpretation;

iii **Long sentences** - Extractive summarizers extract sentences from the text that are regularly too long. These sentences might include unnecessary information that will be added to the output summary, demanding computational space that might be needed for more relevant information.

Similarly, below are the abstractive summarization issues:

i **Over complex summaries** - According to [146], general users of automatic summarizers prefer extractive summaries because they are much more simple, since they constitute fragments of the original text. Hence, the users are much more capable of analysing the text and retaining its essential information [36];

ii **Incoherence** - Much like extractive summaries, abstractive summaries also possess incoherence. This problem is due to the fact that the sentences that are re-generated do not relate to each other.

In sum, both extractive and abstractive summarizers have their own problems. In the case of extractive summarizers, they can be applied to text with different lengths, from small to large documents and create simple summaries, that are more easily readable and understood. If the input consists of large quantities of text and some of them are slightly ungrammatical, the summarizer has no problem in extracting the sentences [9]. As for abstractive summarizers, they create summaries that are, sometimes much more coherent than extractive summaries and they add material that enriches the source text [36], despite being more costly in terms of computational resources. The ideal solution to these problems is to create a automatic summarizer that implements extractive and abstractive summarization approaches [7].

2.4 Summary Evaluations

Even though the state of the art in automatic text summarization had a lot of research, creating an ideal summary, for one or more documents, is still a very hard task to be done. According to the literature, the agreement between human summarizers is quite low, both for evaluating and generating summaries.
To evaluate a summary, one must evaluate its form and content. By form we mean how coherent it is, i.e. grammatically correct and if it is readable by a human, while by content we mean how well will it capture the source text [16] [150].

Besides these problems, another important one exists: the far-reaching evaluation metrics. A number of metrics exist and there is a disagreement in terms of which one is the best.

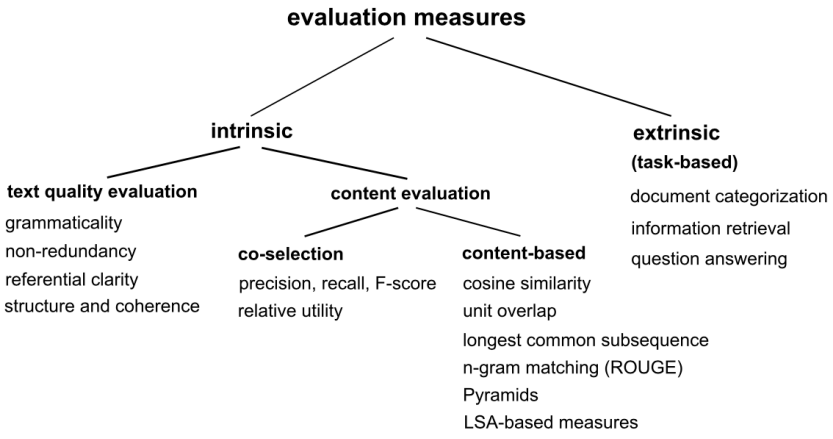

Figure 2.10: Evaluation measures by [9].

There is not a standard human or automatic evaluation metric to compare different systems and create an evaluation baseline.

Beyond that, manually evaluating summaries is a very expensive process: according to Lin [151], the evaluation of a big set of documents in the DUC, would require over 3000 hours of human efforts. So, the creation of an evaluation metric that could be universal to all and that allowed the evaluation of all types of summaries, would reduce the process of manual evaluation[9][16].
Summary evaluation measures are divided in two main groups: **intrinsic** and **extrinsic**. Intrinsic measures evaluate the internal structure of the summary, i.e. they judge the quality of the summarization process, while extrinsic measures evaluate how the quality of the summarization approach affects other tasks [127]. Intrinsic evaluation methods are usually the main approach of evaluation summaries, which is done by comparison with an ideal summary [9].
Intrinsic measures are divided in **text-quality evaluation** and **content evaluation**.
In Figure 2.10 we have an overview of all the existing evaluation measures.
We will begin by explaining the text-quality evaluation measures.

2.4.1 Text-Quality Evaluation Measures

The text quality of a summary can be decided in agreement with four aspects: grammatically, non-redundancy, reference clarity and finally, coherence and structure, however it can not be done automatically, it can only be done by annotators at DUC. They assign a score, from A (very good) to E (very bad) to each one of those aspects [9]

i **Grammaticality** - the summary must not contain non-textual items (i.e. markers), punctuation or incorrect words.

ii **Non-redundancy** - the text must not contain redundant information.

iii **Reference clarity** - every word that has a reference to something, such a noun or pronoun, must be clear in what it refers to.

[9]DUC represent events for disseminating scientific and technical papers written by people that participate in the DUC workshops.

iv **Coherence and structure** - the content of the summary must be coherent and have a good structure.

2.4.2 Content Evaluation Measures

Content evaluation measures evaluate the content of the summary text. They are divided in two sub-groups: **co-selection** and **content-based**.
Co-selection measures only work with sentences that are exactly the same, ignoring the fact that, two sentences might contain the same information, even though their structure is different. Co-selection encompass three very known evaluation metrics: *precision*, *recall* and *f-score* [9] [40] [2].

i **Precision** - consists of the number of sentences that occur both in the summary and in the reference summaries), divided by the number of sentences in the candidate summary (generated by the summarizer).

$$Precision = \frac{|Ref \cap Cand|}{|Cand|} \quad (2.1)$$

where *Ref* and *Cand* represent the reference and candidate summaries, respectively.

ii **Recall** - consists in the number of sentences that occur both in the summarizer output and reference summaries, divided by the number of sentences in the reference summary.

$$Recall = \frac{|Ref \cap Cand|}{|Ref|} \quad (2.2)$$

iii **F-score** - is the geometrical mean of *precision* and *recall*, that blends the precision and recall measures. One can use several variants of the F-score, where the simpler version (F1) is the most used:

$$F - score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \quad (2.3)$$

iv **Relative utility** - Relative Utility solves the main problem associated with using *precision* or *recall* measures to evaluate the quality of a summary - two equally good summaries can have different evaluation values, because the human judges might disagree about which sentences from the document are the most important.

Through the use of the Relative Utility measure, the final summary can represent all the sentences of the source text, with confidence values for their inclusion in the summary.

$$RU = \frac{\sum_{j=1}^{n} \delta_j \sum_{i=1}^{N} u_{ij}}{\sum_{j=1}^{n} \varepsilon_j \sum_{i=1}^{N} u_{ij}} \quad (2.4)$$

where u_{ij} is a utility score of sentence j from annotator i, ε_j is 1 for the top e sentences according to the sum of utility scores from all judges, otherwise its value is 0, and δ_j is equal to 1 for the top e sentences extracted by the system, otherwise its value is 0. A more detailed explanation about this measure is available in [152] [9].

Content-based measures analyse the text content, as opposed to Co-selection measures. In the case of two sentences with the same meaning, but different structure, we have a match because the content is the same.

i **Cosine similarity** - cosine similarity is a well known formula for calculating how similar two vectors are, in a vector space. One common use it to evaluate the similarity between vectors of sentences [100] [108] In this case, it is used for evaluating the content of summaries.

$$cos(X,Y) = \frac{\sum_i x_i \cdot y_i}{\sqrt{\sum_i (x_i)^2} \cdot \sqrt{\sum_i (y_i)^2}} \quad (2.5)$$

where X is a text summary and Y is the reference summary [153] [154]

ii **Unit Overlap** - unit overlap, much like the previous measure, calculates the similarity between two words: X and Y [155].

$$overlap(X,Y) = \frac{\|X \cap Y\|}{\|X\| + \|Y\| - \|X \cap Y\|} \quad (2.6)$$

iii **Longest Common Sub-sequence** - this measure calculates the length of the longest sub-sequence of words, between two word sequences X and Y [156] [157]

$$LCS(X,Y) = \frac{length(X) + length(Y) - edit_{di(X,Y)}}{2} \quad (2.7)$$

$edit_{di(X,Y)}$ stands for the *edit distance*, which consists in the number of edits needed to transform X into Y. These three first content based measures were proposed by Saggion et al. [154] [9] [155]

iv **ROUGE measure** - ROUGE consists in a group of measures, based on the similarity between n-grams[10]. This family of measure was first introduced in 2004 by Lin and was has became a standard for evaluating automatic text summaries [16] [151] .

Given a set of reference summaries created by a group of annotators RSS, the $ROUGE{-}n$ score of a candidate summary is calculated according to the following equation:

[10] An n-gram is a sub-sequence of n words from a certain text.

$$ROUGE - n = \frac{(\sum c \varepsilon RSS \sum gram_{n \varepsilon c} Count_{match}(gram_n))}{\sum c \varepsilon RSS \sum gram_{n \varepsilon c} Count(gram_n)} \quad (2.8)$$

The $Count_{match}$ is the number of n-grams that occur both in a candidate summary and a reference summary, while $Count(gram_n)$ is the number of n-grams in the reference summary [9].

There are other ROUGE scores, such as ROUGE-L, which evaluates the longest common sub-sequence (see the previous section) – and ROUGE-SU4, which is a bi-gram measure that enables at most 4 uni-grams inside bi-gram components to be skipped.

v **Pyramid evaluation method** - this method uses Summarization Content Units (SCU) that are important for comparing the information from summaries. They are created from the annotation of a corpus of summaries that do have a size not bigger than a certain clause. This annotation identifies sentences that are similar and identifies sub-parts that are related. If one of these units appears in more manual summaries, more weight it will have. Each unit is evaluated and according to their weights, the pyramid is created. At the higher positions of the pyramid we have the units that appear in the biggest number of summaries, while at the lower positions we have the units that appear in fewer summaries. The SCUs in peer summary are then compared against an existing pyramid to evaluate how much information agrees between the peer summary and manual summary [9] [158].

2.4.3 Task Based Evaluation Measures

As opposed to the previous evaluation measures, task-based evaluation methods do not focus on the summary sentences, instead they measure the utility of a summary for a certain task. The three most important task-based evaluation measures are the following ones: Document Categorization, Information Retrieval, and Question Answering. Task-based evaluation methods do not analyze sentences in the summary. They try to measure the prospect of using summaries for a certain task. Since there are are a big number of task-based evaluation measures in the literature, we will focus on the three most important ones: document categorization, information retrieval and question answering [9].

1. **Document categorization** - document categorization evaluates how effective the summary is in capturing the information in a text document in order to categorize it. To perform this task, we need a corpus of documents with their respective topics.

   Document categorization can be done manually or by a machine and the results of categorizing summaries are compared with the results of categorizing documents. If we choose an automatic categorization, we might see some inherent errors in the classifier. To deal with these errors, we must discover if the error is from the classifier or if it is from the summarizer and this is done by comparing the systems performance with upper and lower bounds, i.e., full documents or random sentence extracts respectively;

2. **Information Retrieval (IR)** - this task differs from the above one, because it uses an IR system, indexed the summaries, and evaluating how efficient the summary is in representing the main aspects of a document. The principle here is that we have good summaries whenever the retrieved results are equivalent or close to the one obtained from the source documents. Additionally, the difference between the efficiency of the summaries and the full documents represents a measure of the quality of the summaries;
3. **Question answering** - text summarization can be evaluated with regards to question answering. In most cases, the generated summaries are used as options to multiple choice problems. Human analysers have several possible choices from which they must choose one. Depending on the quality of the summaries, i.e. options, the easier it is for the human analysers to answer the questions.

Chapter 3

Implementation

In this chapter, we will list every decision we have taken in the implementation phase. We will start by talking about the programming language, frameworks and libraries used and then proceed to present the summarization approaches we adapted/created.
The obtained results from these experiments will be provided in Chapter 4.

3.1 Sources Used

When speaking about what tools were used, one can refer the programming language and the libraries/frameworks and algorithms used. **Python** was the chosen programming language for the code we wrote. It is an interpreted, high-level and general-purpose programming language, with important functionalities and frameworks, that made the creation of our system much easier. It is being widely used for all kinds of NLP.
The **NLTK**, **Pandas**, **Tensorflow/Keras**, **GloVe**, **Numpy** and **spaCy** were the main external libraries used.
NLTK is an open-source toolkit that provides libraries for classifying, tokenizing, stemming, tagging, parsing, and semantic reasoning text and also wrappers for industrial-strength NLP libraries. We used it especially for pre-processing the input texts, by removing stop-words, tokenizing and lemmatizing the text and for other calculations, in our deep neural network summarizer, presented in Section 3.4.
Pandas is an open-source library for data analysis [159]. We used it for reading and storing data, used in the deep neural network summarizer.
Tensorflow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in Machine Learning (ML) and developers easily build and deploy ML powered applications. **Keras** is a deep learning Application Programming Interface (API) built on top of **Tensorflow**, version 2.0. We used these tools to define every aspect of our neural network model, from the architecture: layers, neurons in each layer, activation and loss functions, to the training and inference phases as well evaluating the model performance[160] [161];
GloVe creates word vector representations from text. We used it to create word embeddings of our texts, in which each English word is assigned a 300 dimension vector with respective values. The word vectors come in a simple text file, so we loaded the file, separated the words from the vectors and created a python dictionary, where each key is a single word and the dictionary value has the embedding values. The word vectors dictionary was used to calculate similarity between sentences or words, in the **TextRank** summarizer (Section 3.4.1) or the deep neural network summarizer (Section 3.4).

SpaCy was used to create English language object representations from the text. With this object we have access to the vocabulary, syntax, entities and word-vectors and also a variety of methods for tokenization, POS tagging and dependency evaluation, named entities, word-embedding vectors, etc.

3.2 Term-Frequency Summarizer

A couple of months ago, when we began searching for solutions for this thesis, we thought it would be interesting to implement a simple term-frequency summarizer, to serve as a baseline for the other solutions we could find and implement in the future. The main idea was to use this summarizer and compare its results, to other more advanced and complex solutions, in order to see if they present better results. Term-frequency approaches are now considered classic approaches, since they have been proposed a few years ago, as we have seen in the Section 2.3.1. Our approach, however, is a bit more complex, but it has the same core logic.
This algorithm receives an input text and a compression rate. The compression rate measures how much shorter the summary is as of the original text. An alternative to a compression rate could be to use a retention rate, that measures how much information is retained in the summary [162]. The text is pre-processed, like in the other approaches and we evaluate the frequency of the words. Each word is assigned the respective frequency in which it appears in the text. The most frequent words are assigned higher scores. The sentence with these high frequency words are picked, according to the compression rate. In the end, the summary is created by combining the chosen sentences.

3.3 TextRank With Word Embeddings

In this algorithm, we began by creating the word embeddings for the whole text. Each word embedding is a key pair, in which each key is the word and the value is a vector of 300 values. In our approach, much like a typical **TextRank** algorithm, the text is represented as nodes from a graph, where each node is a single sentence and each edge is the relationship between two sentences. We began by creating a matrix that combines pairs of sentences. From each sentence pair, we get their word embeddings and calculate their cosine similarity. Having completed the similarity matrix, we convert it to a graph and rank the sentences according to their similarity scores: the highest scoring sentences come first, along with their respective values.
Finally, we filter and combine the ranked sentences, according to the compression rate (decimal percentile value) provided, into a full summary.
Below we have the pseudo-code of our algorithm and a brief explanation of the variables and methods:

1. **sumEmbeddings** - vector that stores the sum of the given sentence's words embeddings values;

2. **M** - matrix of N by N sentences;

3. **cosSim** - the cosine similarity between two vectors, where each one stores the sum of the respective sentence's words embeddings values;

4. **G** - graph created from the matrix M;

5. **S** - computed ranked nodes (sentences) from the graph;

6. **comp** - float value that represents the desired compression of the text;

7. **RS** - variable that stores the ranked sentences, according the compression rate;

8. **summary** - the variable that stores the summary text. initialized as an empty string and afterwards appends the ranked sentences.

Algorithm 1 *TextRank* algorithm.

```
for sent in Sents do
    if sent != NULL then
        words ← len(sent )
        sumEmbeddings ← ∑_{n=1}^{words} wordEmb
    else
        sumEmbeddings ← sumEmbeddings.append(sentEmb)
    end if
end for
M ← matrix(N,N)
for si in Sents do
    for sj in Sents do
        if si != sj then
            siSumEmbeddings ← sumEmbeddings(si)
            sjSumEmbeddings ← sumEmbeddings(sj)
            A(si, sj) ← cosSim(siSumEmbeddings, sjSumEmbeddings)
        end if
    end for
end for
G ← graph(M)
S ← scores(G)
RS ← sortSents()
RS ← RS[0 : |RS| * comp]
summary ← ' '
for rankedSent in RS do
    summary ← summary + rankedSent
end for
for sent in summary do
    summary ← summary + ' '.join(sent)
end for
    return summary
```

3.4 Deep Neural Network Summarizer

Due to the fact that Artificial Inteligence (AI) is very popular nowadays and we have a particular interest in the field, we thought it could be a good idea to implement an AI algorithm, capable of finding a solution for the summarization task. We began by researching possible existing implementations of AI algorithms for this specific task and we found some with potential results, namely Encoder-Decoder based approaches and text classification algorithms [163] [164] [34]. We decided to implement our own version of a feature-based summarizer and combine it with machine learning. The main idea consists in calculating a set of text features and use the results as input for our machine learning algorithm, while the output is used to decide whether a sentence must be in the summary or not.

3.4.1 Datasets

Like every machine learning model, ours needs data to train with, the more data is fed to the model, the more accurate it might be. So, we decided to look for big datasets for text summarization.
We found two large datasets, the **wikihowAll** [165] dataset and the **Newsroom** dataset [166]. The first represents a set of tutorials from Wikipedia, while the latter represents a set of news collected from several news websites across the Internet. They have a common aspect: for each document, a full text, summary and title are provided. This is an important aspect, since after we create our summaries, we want to evaluate how good they are and this can be done, because we have access to the reference summaries in the dataset.
In the case of the **Newsroom** dataset, it is represented in a JavaScript Object Notation (JSON) format, so we decided to convert it to a Comma-Separated Values (CSV) file, to keep every dataset in the same format. Table 3.1 displays a row of the **Newsroom** dataset.

Table 3.1: Newsroom dataset

Title	Text	Summary
Pro Sports Xchange notes	BANGALORE, India, June 4. The world's biggest computer services company could not have chosen a more appropriate setting to lay out its strategy for staying on top. A building housing I.B.M.'s software laboratory and application service teams on the company's corporate campus in Bangalore, India.	SAN DIEGO PADRES team notebook.

The whole datasets were loaded. Each row represents a single document, which has the full document text, the title and the summary. Some datasets do not include information about the title. Some ideas to fix this Automatic Title Generation (ATG) problem were proposed by the scientific community. Most of them use mechanisms such as keyword extraction, topic calculations, dependency trees, term-frequency and word embeddings to generate the title, making sure it represents the majority of the text's content, as one can read in this paper [167]. Our idea was to use our previously proposed summarizer, based on *TextRank*, to generate the title. As explained in the the algorithm's respective section, i.e. Section , one may have problems in generating the title, if the text is too big to handle.

3.4.2 Pre-Processing

The document text, title, and summary are pre-processed in order to end up with a less noisy set of features. In the case of our deep learning neural network, we did it as follows:

i **Convert the text to lowercase** - the first step was to convert all the text into lowercase. By having the text in this format, we assure that all the words are equally represented;

ii **Convert contractions to their normal forms** - some words in a document are not formally written as they should. Most of the times, they are abbreviated, e.g. "ain't". Even though we humans understand it, a computer in the contrary needs a way to make it easier to understand. So, we converted these words into their full representations, using a contractions dictionary. Each key of the dictionary is an abbreviated word and each value is the right representation of the word;

iii **Split the text and remove unwanted words** - so far we only converted the text to lower case and fetched the full representations of the words, therefore at this point we still have a few unwanted words in the text. To fix this issue, we tokenized the text (converted the document text into a list of word tokens) and we assessed if they represent stop words (words that do not convey important information) or special characters, such as: HTML tags, exclamation, interrogation marks, except dots because if we removed the dots we could not tokenize the document text into sentence tokens later on;

iv ***Lemmatize* words** - with the list of cleaned word tokens, we then applied lemmatization and converted each token to its most simplest form. This consists in an important step, because sometimes there are words that are very similar, but have opposite meanings;

v **Calculate sentence tokens** - after having the list of lemmas, we then *detokenized* it into simple text so that we could calculate the sentence tokens. With this list, the feature selection was much simpler.

3.4.3 Feature Selection

After loading each dataset, we proceeded in computing the features for each document. Each document's sentence is assigned a list of features with only numerical values.
We selected the following features for our system:

1. **Sentence position** - Depending on where a certain sentence is placed impacts the score it has. If it is present in the beginning or end of the text it will have a higher score. In our implementation, after the pre-processing phase we get a document as a list of sentences. For a given sentence, the index in the list, determines its position in the document;

2. **Number of title words in sentence** - The number of title words in a sentence impacts its score: the more it has, the more important it is. This feature is simply calculated by counting the common words, between the sentence and the title;

3. **Sentence similarity to title** - A sentence that is more similar to the title, both syntactically and semantically, has a greater relevance to the summary. Given a text sentence and the title we then calculate the cosine similarity of the sentences word embeddings, with the help of a previously built method from the *Spacy* library;

4. **Numerical data in the sentence** - Since numbers convey important information, we decided not to remove them in the pre-processing phase. In the calculation of this feature, we count the quantity of numbers in a sentence and divide it by number of word tokens (i.e. number of words) the sentence has. The word token calculation is done with the help of NLTK;

5. **Temporal expressions in the sentence** - Much like the previous feature, considering the temporal expressions in a sentence, means we have to divide the sentence into word tokens to check if they represent a temporal expression (date or date time);

6. **Relative sentence length** - This feature is calculated by first discovering the longest sentence in the document. Afterwards, we divide the length of our sentence by the length of the longest sentence;

7. **Content words in the sentence** - Content words are nouns, verbs, adjectives, and adverbs. They are the most important words in a sentence representing the real content presented in a text document. To calculate the number of content words in a given sentence, we used POS tagging from the NLTK that labels each word with its corresponding syntactical tag;

8. **Number of upper words** - Upper words are also an important part of a text, they usually represent acronyms, locations, or entities. We simply check if they are upper case, if so compute the percentage of such words in the sentence;

9. **Sentence polarity** - Sentence polarity was calculated through sentiment analysis of the text, i.e. we evaluated how positive or how negative is a given sentence. The higher the value, the more positive it is and vice-versa. Here we used the external library *TextBlob* that has all the necessary methods for evaluating text polarity;

10. **Sentence topics similarity** - This feature is a new one we are proposing. It represents how similar is a sentence to the topics of the document. A topic represents a theme/subject, and the more similar a sentence is to a topic, the more important it is in that topic and in the text. Sentences that have no sort of similarity to topics, are sentences that do not convey important information. With the processed text, we created a Latent Dirichlet Allocation (LDA) model, to get the topics and topic words of the text. The topics are sorted according to their relevance from the LDA model. From the sorted topics, we pick the top 3 topics and their respective top 10 words and store them in a dictionary, whose keys are the topics and values are the topic words. With the dictionary, we calculate the word embedding vectors for both sentence and topic words. Afterwards we just have to calculate the cosine similarity between the vectors.

The word vectors were computed with the same **GloVe** pre-trained word embeddings, used in the other functionalities;

11. **Sentence similarity to most frequent sentence** - This feature, as the previous one, is a new feature we are proposing. We thought it could be relevant to incorporate a feature that analyses the text with regards to the frequency of which the words and sentences appear. The main idea consists in compressing the original document's text in 30%, and retrieving the most scored sentence of the text. Then we evaluate if a given document sentence is indeed the most scored sentence, by calculating the similarity of the sentence to the most scoring sentence. The similarity value is in fact the feature's score. Even though the similarity is made between two pre-processed sentences, where irrelevant words, i.e. stop words, are automatically removed, the feature may induce errors, since most frequent words are in most cases, irrelevant;

12. **Sentence final score** - This feature is the sum of the values of all the features for a given sentence. It represents how relevant is the sentence, given the values from all the features. The higher the final value, the more probable it is added to the summary;

13. **Labels** - Declares if a sentence must be or not in the summary. If the similarity between the sentence and at least one other sentence from the summary is higher than 0.9 (this value, may change accordingly), means that the sentence is, in practice, represented in the summary. If so, the label has the value 1, otherwise has the value 0. This feature aids the model's learning capability, since it tells the model, in a supervised manner, that a sentence is indeed in the summary or not.

Since the previous feature lists had values with different scales, we had to normalize them in order to smooth the learning process of our neural network. In the case of the label's list, they were already represented as a set of zeros and ones. In each dataset, we calculated the documents features and saved the data in a complete CSV file, that was used to train the model.
Before training, we executed a simple step that analyses the importance of each feature in our classification problem. In this way, we can filter the less important features and train the model only with the most important. This way, our model will perform better.
In Figure 3.1 we can see how the different features contribute to the sentence classification. The three most important features are represented by the numbers 11, 2 and 5, which are the **Sentence final score, Sentence title similarity** and **Sentence length**, while the less important features are represented by the numbers 3,4,7 and 8, which are **Numerical data, Temporal data, Upper words** and **Sentence polarity**. The fact that the hierarchy of most important features is represented like this, means that features that evaluate certain rare aspects of the sentences/words have less importance for the sentence selection (in this case it is practically none).
In Figure 3.2 is an example of the content of the CSV file.

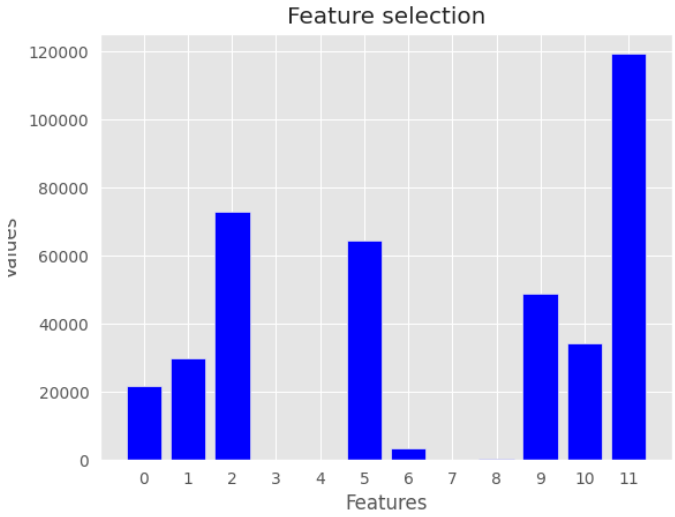

Figure 3.1: Feature selection.

Processed sentences	Processed headline	Sentence position	Title words	Sentence title sir	Numerical data	Temporal data	Sentence length
This fall, turn 20.	To celebrate 20th anniv	1	0.5	0.801936150611	0.666666666667	1	0.041666666667
Twenty year history, hosts		0.9736842105263	0.5	0.435657606676	0.181818181818	0	0.375
Twenty year life.		0.9473684210526	0.5	0.677178773501	1	0	0
Joe Kernen I work togethe		0.9210526315789	1	1	0.47619047619	0.42857142857	0.791666666667
When first face think 20 ye		0.8947368421053	1	0.946641141571	0.307692307692	0.46153846154	0.1875
But luckily, could worry ce		0.8684210526316	0	0.698399021439	0	0	0.270833333333
So begin 20th year moving		0.8421052631579	0.5	0.743705770368	0	0.75	0.083333333333
After year New Jersey stu		0.8157894736842	1	0.519493845869	0	0	0.625
At time, felt like leap faith		0.7894736842105	0	0.661443199421	0	0	0.166666666667
Now, nine month later, loo		0.7631578947368	0	0.846619107112	0.222222222222	0	0.291666666667
Our access guests, excite		0.7368421052632	0	0.662937508147	0	0	0.229166666667
This bring back original mi		0.7105263157895	0	0.701074412629	0	0	0.166666666667
We design whole thing trac		0.6842105263158	0	0.715446466019	0	0	0.416666666667
With mind, I go back origir		0.6578947368421	0	0.723053896198	0	0	0.166666666667
I start process say let us r		0.6315789473684	0	0.647074027802	0	0	0.1875
worry, totally resist.		0.6052631578947	0	0	0	0	0.020833333333

Content words	Upper words	Sentence headline similarity	Sentence polarity f	Sentence LDA scores	Sentence frequency score	Sentence final score	Labels
0.0808080808	0	0.619826313108187	0.94139534883721	0.556680442617552	0	0.574254309885545	0
0.5179063361	0	0.536722626275842	0.38883720930233	0.413813489849056	1	0.602378202867179	0
0.3636363636	0	0.558646384033968	0.94139534883721	0.488032030438819	0	0.545107223589673	0
0.7272727273	0.095238095	0.692162483076332	0.86046511627907	0.912381430049171	0	1	0
0.68997669	0.307692308	0.830128193692006	0.15860465116279	1	0	0.675533409607667	1
0.1140819964	0	0.82211088447106	0.32744186046512	0.804174889433709	0	0.356582806436035	1
0.5757575758	0	0.770418121160582	0.94139534883721	0.700962371913312	0	0.591047548036173	1
0.862745098	0.117647059	0.653782675317313	0.49348837209302	0.801344175134152	0	0.737021389788208	0
0.6464646465	0	0.745276140749419	0.94139534883721	0.752695530297359	0	0.507056816918966	0
0.4579124579	0.222222222	0.775066554866385	0.98744186046512	0.94453393091459	0	0.59221671201336	1
0.7595959596	0	0.702143235190923	1	0.66074588857017	1	0.635293951601316	0
0.6464646465	0	0.8071774195147	0.56151162790698	0.723592924100883	1	0.519884561110211	1
1	0	0.801058138128852	0.8493023255814	0.74195906402258	0	0.618745862787068	1
0.3636363636	0.333333333	0.745890051162505	0.59604651162791	0.727736548210234	0	0.368082700772602	0
0.4289044289	0.307692308	0.685072878322044	0.94139534883721	0.744802445710059	0	0.43218658204098	0
0.5333333333	0	0.321409357525011	0.94139534883721	0.276827027234956	0	0.204895128725017	0

Figure 3.2: Training, testing accuracy and loss.

3.4.4 Model's Architecture

Creating a neural network is not an easy task, since it requires a lot of tweaking in the architecture. It is hard to find the ideal architecture for the problem at hand, because every problem is different from the other and one does not have a standard that can be applied to all the variety of problems.
Initially, the model's architecture was very simple: it only consisted of one input, one hidden and one output layer. The hidden layer had only 20 neurons, with a Rectified Linear Unit (RELU) activation function to prevent the vanishing gradient problem. The output layer had only had one neuron, since it is a typical binary classification problem, where we only have two label classes (0 and 1). The chosen loss function was the **Binary Crossentropy** function, since it is known to be the best for these types of problems according to the literature. As for the optimizer, the preferred one was **Adam**, because it requires less memory and so it prevents memory overload when training.
When we successfully had more data to train the model, we improved the model's configuration, by adding three more hidden layers with RELU activation function. In overall the number of neurons in each dense layer increased: from the first to the fourth layer the number of neurons were 40, 30, 25 and 20. All the layers had regularization mechanisms to improve the model's learning and reduce errors, as it is shown in Figure 3.3. These mechanisms will be explained in the next section.

3.4.5 Model Training

Training the model was much easier than creating and parameterizing it. From the CSV features file, we loaded the features columns into a matrix and the labels column into a list. They were then used for creating the training and testing sets. The **sentence headline similarity** was not used in the training of the model, instead it was only used to create the **sentence final score** feature, as it would lead the model to bad judgements. In terms of percentages, 70% of the data was used for training, while the remaining 30% was used for validation.
The training was done on the features file, which had a size of 2.5 GB, during 45 epochs and a batch size of 28. During training, two possible Keras callbacks could be used: **ModelCheckpoint** and **EarlyStopping**. While the first automatically saves our model each epoch the validation accuracy increases, it may not be a good practice because the validation loss may also increase in some cases, so we preferred the latter because it monitors the validation loss in a given interval of epochs. If this loss increases, it means that our model is not improving, therefore it stops the training process. This process saves us time since we do not need to wait till the end to save the model and also prevents losses in the model's performance.
From epoch to epoch we analysed the model's performance in the validation set. After a certain number of epochs, the model decreased in performance and began overfitting, because the validation accuracy reached a peak from which it did not improve and the validation loss started increasing.
In Figure 3.4 we have two plots that display the evolution of the model over time, in the

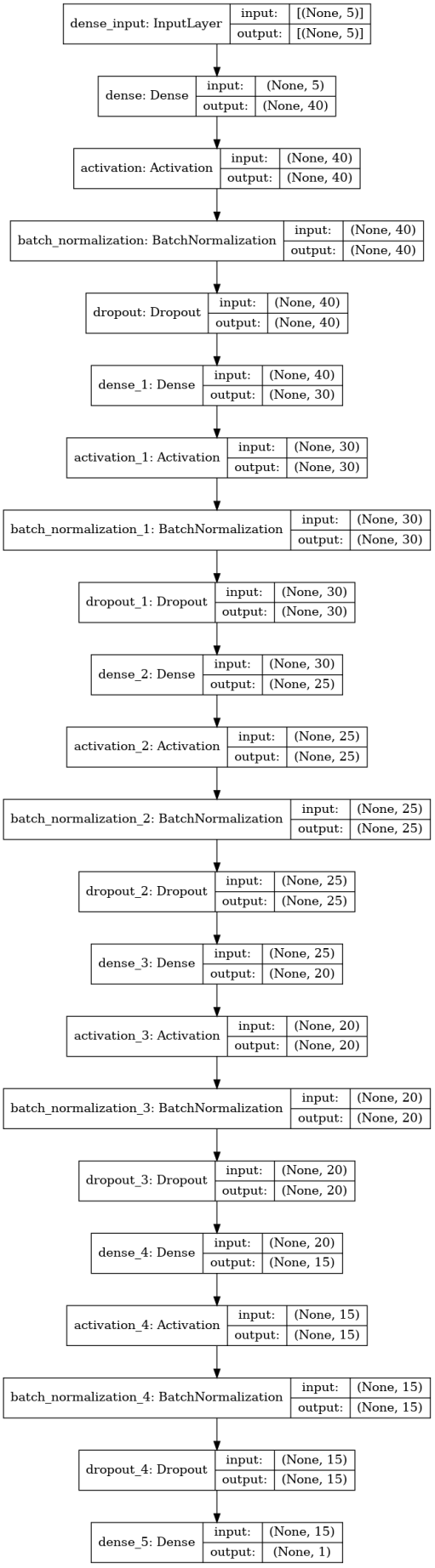

Figure 3.3: Neural network's architecture.

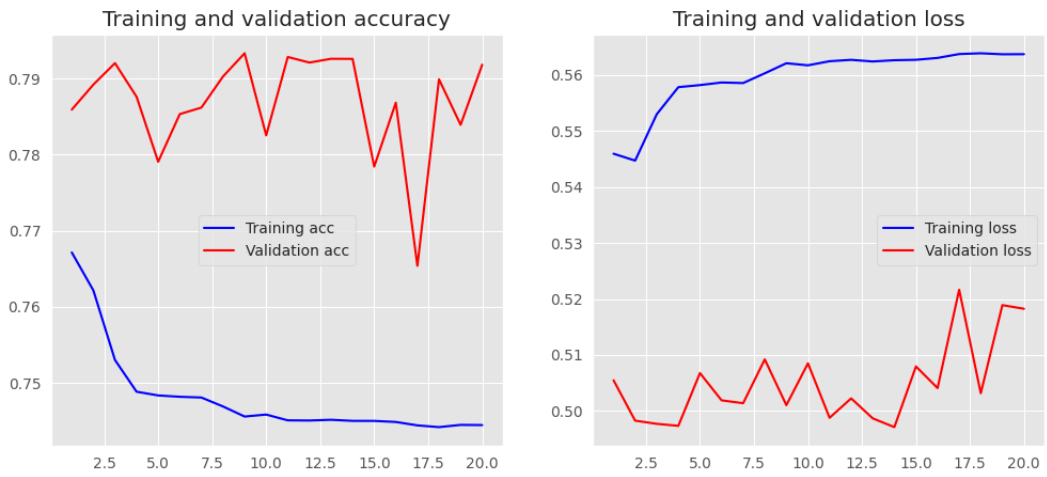

Figure 3.4: Training, testing accuracy and loss while over-fitting.

training and validation sets. From the Figure we see that around epoch 17.5 the model began having problems, since the validation accuracy decreased a lot and the validation loss also increased.

When overfitting the model lost its capability of predicting new data, meaning that if the model was used in the summary generation, it would not be able to pick the correct sentences for the summary because the predictions would not be correctly calculated.

According to the literature, several mechanisms can be used to prevent this problem, which are the following:

1. **Dropout layer** - the most common mechanism to prevent overfitting in deep learning models is to add a dropout layer. A dropout layer can be added directly when creating a specific input or hidden layer or instead append the dropout layer to the respective hidden layer. In either cases, one must define a percentage that represents the number of neurons that will be ignored when training. They are "dropped-out" randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass. The higher the dropout rate, the more neurons are ignored, hence the less probable is the model to overfit. In our model, we kept tweaking the dropout rates of each hidden layer, until we decided that the best rates were 70% in the input layer and 50% in each hidden layer;

2. **Keras regularizer** - another interesting mechanism for preventing overfitting is to add a weight regularizer to the dense layers. Regularizers apply penalties on layer parameters or layer activity during optimization. These penalties are summed into the loss function that the network optimizes. They can be of three different types: kernel, bias an activity, and they apply penalties on their respective aspects of the model. In our model, we only applied one regularizer, which was a kernel **L2** regularizer. The regularizer values were 30% for the input layer and 25% and 20% for the remaining layers;

3. **Batch normalization layer** - a batch normalization layer can be added to each input

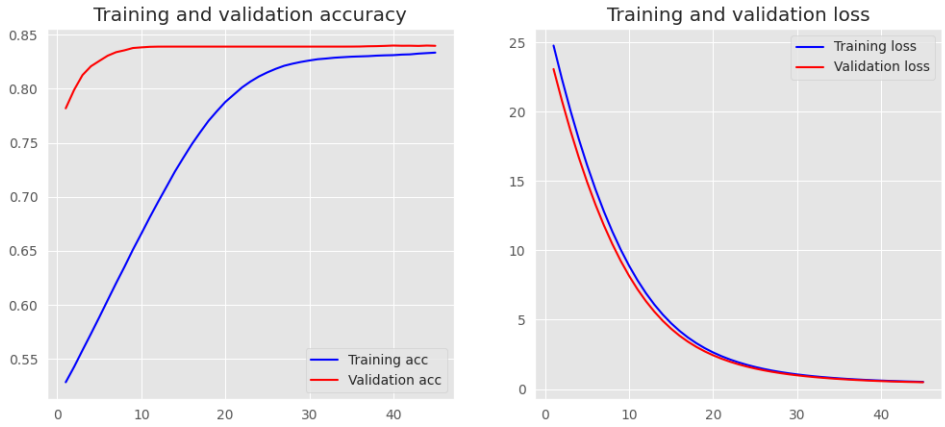

Figure 3.5: Neural network without over-fitting.

and hidden layers and it is used to normalize their inputs. It applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. This layer works similarly to every other normalization and standardization mechanism.

Besides these three types of regularization mechanisms, one can also add *LeakyRELU* layers to prevent the vanishing gradient problem[1]. Having a vanishing gradient problem in the model, means that the model prematurely converges to a poor solution.
As we can see in Figure 3.5, with this new improved architecture and more training data we were able to fix our overfitting problem.
In terms of performance, the model reached a peak of 85% in validation accuracy and 8% validation loss. These values could be further improved with more training epochs and data.

3.4.6 Summary Generation

Generating a summary with our features summarizer depends on the model's capability of inferring new data. The more data the model uses for training, the better the predictions it makes and therefore more accurate and coherent the summary will be. With the document text, title and one reference summary the algorithm calculates the text features. Each feature is normalized and filtered, so that only the most important features are chosen to compute the predictions. As we have seen in Figure 3.1, only some features are important for the summary creation, so we decided to pick the 5 most important features.
In our first version of the algorithm, we filtered the computed model prediction that had a higher final score than 0.5. By doing this we would end up with only the predictions that would represent possible sentences for the summary. This approach was not yet entirely correct, since some predictions would have values close to 1, i.e. representing possible summary sentences, while their respective labels were close to 0. Having such discrepancy would induce the algorithm in wrong judgements, because the model would choose some sentences

[1]the vanishing gradient represents the impossibility of propagating useful gradient information from the model's output layer back to the input layers of the model

that were not correct. In order to fix this problem, we decided to calculate the absolute value of the difference between each label and computed prediction. If this difference was lower than 0.2, it meant that in general the model would only pick the right sentences. Even though this was a good approach, if the model is not sufficiently trained in some cases it is not be able to compute the right predictions, so the previous discrepancy still exists thus no sentences are picked and added to the summary.
An alternative to this approach was instead to only pick the sentences whose predictions are above a given value, obtained with a compression rate. The computed predictions were sorted, from the least scoring to the highest scoring and filtered with a compression rate of 20%. So only the highest 20% scoring predictions are chosen to pick their respective sentences and combine them into the summary.
In the following text box, we present an example of a text document and the sentences, in bold, that this algorithm chose for the summary with 20% compression rate:

Original document and chosen sentences

US INSURERS expect to pay out an estimated Dollars 7.3bn (Pounds 3.7bn) in Florida as a result of Hurricane Andrew - by far the costliest disaster the industry has ever faced. The figure is the first official tally of the damage resulting from the hurricane, which ripped through southern Florida last week. In the battered region it is estimated that 275,000 people still have no electricity and at least 150,000 are either homeless or are living amid ruins. President George Bush yesterday made his second visit to the region since the hurricane hit. **He pledged the government would see through the clean-up 'until the job is done'.** Although there had already been some preliminary guesses at the level of insurance claims, yesterday's figure comes from the Property Claims Services division of the American Insurance Services Group, the property-casualty insurers' trade association. It follows an extensive survey of the area by the big insurance companies. Mr Gary Kerney, director of catastrophe services at the PCS, said the industry was expecting about 685,000 claims in Florida alone. However, the final cost of Hurricane Andrew will be higher still. Yesterday's estimate does not include any projection for claims in Louisiana, which was also affected by the storm, although less severely than Florida. The Oakland fire disaster, in California last year, cost Dollars 1.2bn. **By contrast, insurance claims resulting from the Los Angeles riots earlier this year - the most expensive civil disturbance in the US - totalled just Dollars 775m.**

3.5 Encoder-Decoder

During the research and implementation of our neural network summarizer, we found some other deep learning algorithms with great results. The most relevant one was a *Seq2Seq* algorithm that generates abstractive summaries[2]. A *Seq2Seq* is a deep-learning algorithm

[2]The source code is free of use and you can find it in the following website [168]

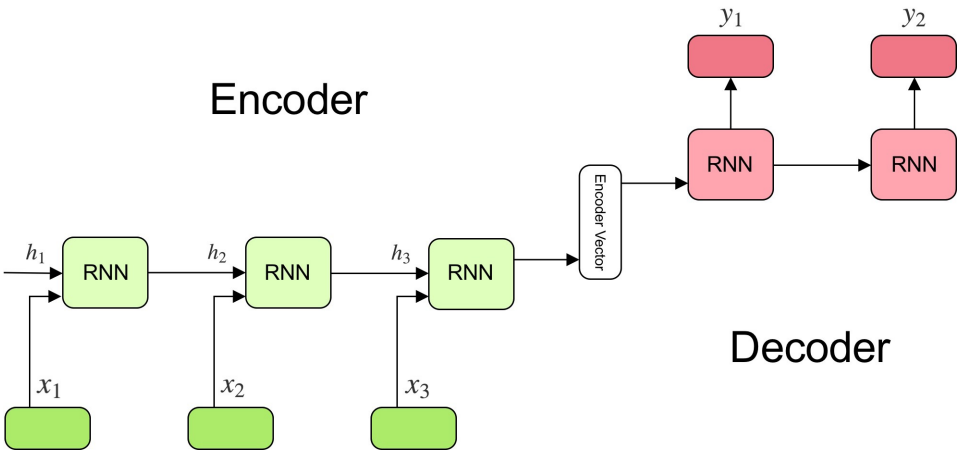

Figure 3.6: Sequence2Sequence model.

that solves problems where the main source of data is sequential. The source data can be text sequences, byte sequences or any kinds of sequences.
From the literature, we could envision a promising application of this method to our problem. It only needed some improvements in some aspects and to be better trained.

3.5.1 Overview of the Architecture

Usually, a *Seq2Seq* model is comprised of two parts: an encoder and a decoder Section 3.6. Each of them are neural networks, that usually combine one or more Long-Short Term Memory (LSTM) layers with embedding layers. In the training phase, the encoder receives the raw text data and at each time-step (the process of going from one layer to another), one word is fed - this allows the encoder to understand how the words are related. The encoder needs to have an initial state, which is a randomly generated vector of zeros. Each LSTM layer returns its own output states, that are used by the next layer. Intuitively, the final encoder's LSTM layer returns the state that is used as the initial state for the decoder.
On the other hand, the decoder uses the initial state to process the target sequence, i.e. a sequence of tokens that represent a piece of text of the generated summary. In our case, the target sequence is in general a sentence or a paragraph. Start and end tokens are added to the start and end of the sequence, to let the decoder know what are the limits of the sequence. In the testing phase, the encoder receives the input sequence and returns a similar decoder state. The decoder analyses the whole target sequence and at each time-step returns the probability of each next target word. The word with the highest probability is selected for the summary. This word is then used again, as the input for the decoder, which repeats the process until the end token is found. The chosen words are combined into a single text, which is the generated summary.
As we can see in Figure 3.6, from a text sequence, each word X_i moves through each LSTM layer of the encoder. The encoder's last layer returns the states that are used as input for the decoder. The decoder combines the sequence with the start and end tokens. Each sequence's words, Y_i are sent to the LSTM layers and in the end return the final sequence.

3.5.2 Dataset Loading/Pre-processing

Initially the chosen dataset was an Amazon reviews dataset, that spans a period of more than 10 years [169], having a total of 568454 data rows. Each row contains a single review and combines the original review with a summary created from that review. Even though it seems to be a big dataset, in fact it is not, because the average length of the reviews is around 40 characters. Having such a little amount of words in each review, makes the learning process simpler, but also less accurate. In order to fix this issue and improve the learning of our model, we decided to create a different dataset, from the original Newsroom dataset [166]. From the Newsroom dataset, the whole texts and summaries columns were used. With them, we created pairs of sentences, where the first sentence was a sentence from a news article, and the other was its most similar sentence from the article's summary. To compute the similarity values, we calculated the cosine similarity between the sum of the word embedding vectors from both sentences. By doing this, we were able to supply the model with a much more relevant dataset, since it helps in understanding how the sentence's words are related to the summary's words and what words and sentences need to be sampled and combined into the summary. The dataset has a total of 237139 rows.

Unlike every other approach we presented, we did not need to pre-process the texts and reference summaries. This is because we want our algorithm to learn every aspect of the text. Even the stop-words which are in general chopped, are kept here to be used by this algorithm so it knows how the most important words are connected, such as nouns, adverbs, adjectives, among others.

With the tokens sequence, a distribution is calculated in order to analyze the reviews and summaries length. The texts max length influence the amount of neurons needed in the encoder's input layer, as we will explain in Section 3.5.4. The initial value for the max texts length was 30 and for the max summaries lengths was 8, since the author was training with Amazon reviews dataset. In our case, the majority of sentences have a length of near 200 characters, while the summary sentences have around 40 characters. Therefore we decided that the best option was to use a max text length of 100 characters. As for the summaries max length, we use it to delimit the summaries that the model creates, as we will see in Section 3.5.7.

As illustrated in the Figure 3.7, the X axis represents the text lengths, while the Y axis represents the frequency of the correspondent length. For this specific example, we are presenting the whole dataset. As we can see, the majority texts lengths is around 200 characters while the majority summaries lengths is 40.

For the algorithm to understand where each text sequence begins and ends, we added two special tokens, **sostok** and **eostok**, that are used as the start and end of sequence tokens.

3.5.3 Vocabulary Creation

Building a vocabulary of the text means creating a set of unique words that appear in it. Our dataset was splitted into the training and testing sets, that we used to compute the full vocabulary size. Having created the vocabulary, we then fit the training and test sequences to the

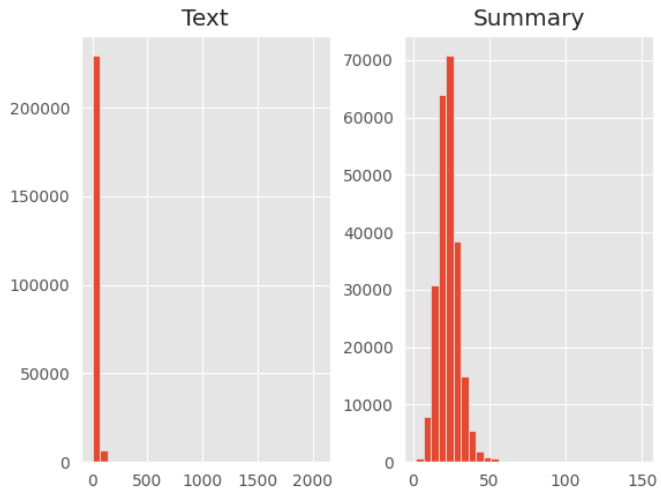

Figure 3.7: Sequence distribution.

words of the vocabulary and convert them to integer sequences. This step aids the model's learning capability, because it teaches the model how the words are related to each other. Since the sequences are of variable lengths, we had to pad them with sequences of zeros.
Instead of using tokenizers to convert the text sequences into integers, we could have used word embeddings. We chose not to do so because as we will see in the next section, the model already uses an embedding layer to compute the word embedding vectors from the input texts.
The sequences that only contained the two start and end tokens were completely removed, as they correspond to empty sentences.

3.5.4 Initial Model's Architecture

The author's implementation of this sequence-to-sequence model composed two parts: an initial model specifically for training and a final model created with the weights from the initial model and used for testing).
We shall begin by describing the initial model's architecture, by dividing it in two: the encoder and decoder's and presenting our changes to the architecture.
In the initial encoder's architecture, we had the following layers:

1. **Input layer** - Has a shape equal to the maximum text length;
2. **Embedding layer** - Has a shape equal to the text vocabulary size. It is used to convert the positive integers into dense vectors of 300 dimensions, i.e. word embeddings of 300 dimensional vectors;
3. **LSTM layers** - Three dense LSTM layers, all with a shape of 300. They were responsible for the learning process of the model and they had dropout layers attached as their main regularization mechanism.

As opposed to the encoder's configuration, the decoder was comprised of only three layers:

1. **Input layer** - Receives the encoder's LSTM states, i.e. the encoded sequence;
2. **Embedding layer** - Has a shape equal to the summary's vocabulary size, but convert's the integer sequences into dense vectors of only 100 dimension size;
3. **LSTM layer** - It is the decoder's final layer and it is the one that decodes the received sequence into the text summary.

Like we have mentioned above, the initial model was created to fit the Amazon reviews dataset. The max texts and summaries lengths were 30 and 8 respectively, which was enough to deal with that specific dataset, but that is a bad practice because the model will likely not perform well with bigger texts. We needed to fix this, since our goal was to create a summarizer that is adaptable to different kinds of documents with varying lengths. Different max text lengths and consecutively numbers of neurons can be used, but most of them do not yield optimal results. After a couple of tests with values spanning from 10 to 500, we thought that the best possible value would be 100, since it does not make our model extremely heavy and slow to train and at the same time it would probably represent good results when predicting new texts. One note worth mentioning is that using higher values than 500 made our model extremely heavy and running out of memory, although our computational system having a good level of resources.
After configuring the encoder and decoder layers two additional layers were added: an Attention Layer and a Time Distributed layer.
In general, in a *Seq2Seq* model, the encoder receives the input text and converts it into a fixed length vector, which is then sent to the decoder to proceed in predicting a sequence. This works well for small texts, but in the case of longer texts, the model can not memorize and encode them all at once. The Attention Layer aids the model in focusing on specific parts of the text and improves the performance. The attention mechanism can be of two types: global or local. In a global attention mechanism, all the text positions matter which consumes a lot of resources and time, while in a local attention mechanism, the model intelligently reduces the search space further and further by focusing only the most relevant parts. The Time Distributed layer uses the vocabulary size and divides the text sequences into slices that represent how many time-steps the encoder-decoder has - a time-step consists in moving from one LSTM layer to another.
In Figure 3.8 we have a diagram where we can see, in detail, the initial encoder configuration, in terms of layers and shapes.

3.5.5 Training Phase

As mentioned above, the initial encoder-decoder model was the one used in the training phase. In our initial implementation, the model was trained on the Amazon reviews [169] dataset for 50 epochs with a batch size of 128. Training on a dataset with such a big batch size has the advantage of taking less time and memory to train, but has the disadvantage of less accurate results. After switching to our custom dataset, we had to adapt the training process of the model. We tried training the initial model with our dataset and see how it would

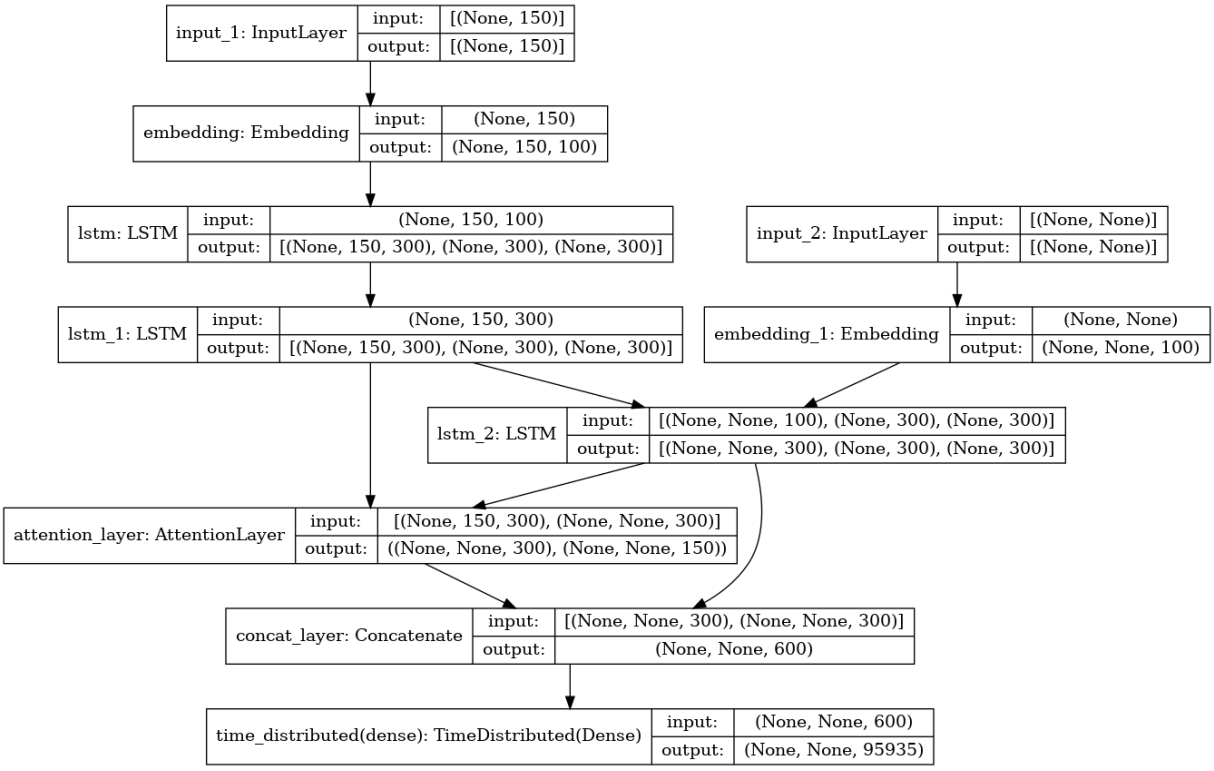

Figure 3.8: Initial Encoder-Decoder model configuration.

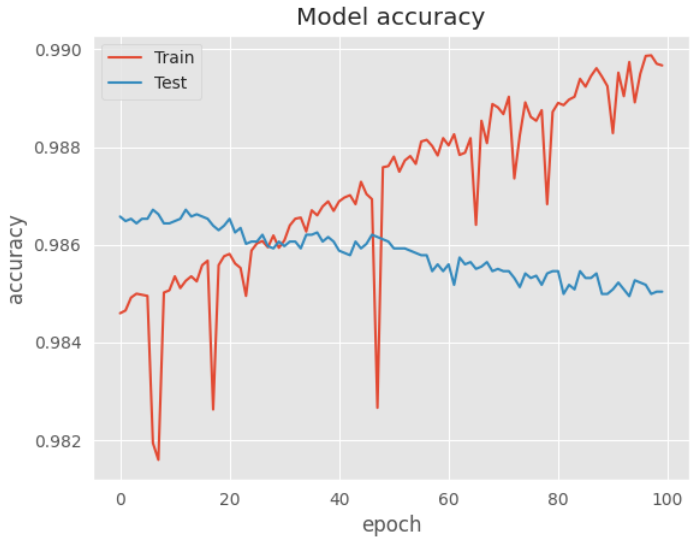

Figure 3.9: Initial encoder-decoder over-fitting.

perform: we used only 10000 rows, for 50 epochs with a batch size of 4. The model's overall performance was not good, since it began over-fitting: the validation accuracy fluctuated and then remained the same for several epochs, while the training accuracy kept on rising. This issue can be seen in Figure 3.9.

Fixing it was a difficult task, since there is no direct procedure that can be applied to all sorts of problems. We tried different hyper-parameters[3], such as the number of hidden LSTM

[3]A hyper-parameter is a parameter whose value is used to control the learning process

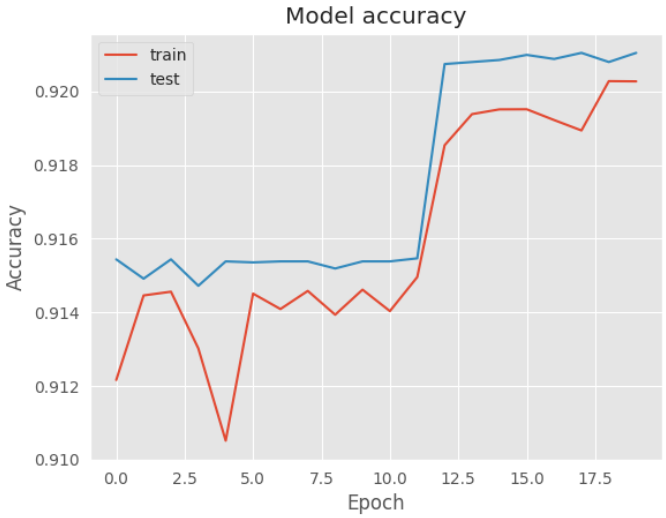

Figure 3.10: Initial encoder-decoder without over-fitting

layers and regularization mechanisms. We used the same amount of samples, epochs and batch size used in the first training phase to test the different possible architectures.
Our first approach consisted in creating a very small model comprised of only one hidden layer and no regularization mechanisms, so it makes the model less complex and easier to train, but with the cost of accuracy. This first architecture was good but not the ideal, since after a couple of epochs the model kept over-fitting.
This first architecture was improved by adding two regularization mechanisms. Each hidden LSTM layer was combined has 20% dropout and kernel L1 and L2 regularization mechanisms were added with a value of 0.001[4]. The model was retrained with the same parameters and the model did perform better with the addition of only over-fitting further in the training phase. We then decided to increase the regularization mechanisms, by using a higher percentage in the dropout layers, 40% in this case, and the L1 and L2 regularization mechanisms increased to 0.1. However this two mechanisms were not enough to stop the model from over-fitting, so we made one last change and increased the dropout to 80% and 60% in the input and hidden LSTM layers respectively, while the L1 and L2 regularizers increased to 0.2. According to the literature, using a higher dropout rate in the input layer and a slightly lower dropout rate in each hidden layer is the best approach to prevent over-fitting, as-well combining it with the L1 and L2 kernel regularizers.
Finally, since the model was no longer over-fitting, we opted for adding another hidden LSTM layer, hoping to improve the model's accuracy. As we can see in Figure 3.10, the model did perform better and fit well to the data after the epochs.
From this point we kept this last architecture and continued training the model with more data. The model's performance increases with the amount of data it receives for training. The batch size stayed equal to 4, since using a smaller value with the same number of instances saved memory that we needed for other tasks. Another aspect that made our model reach better performances was using the *EarlyStopping* callback to automatically stop the training

[4]L1 and L2 kernel regularizers apply a penalty on the layer's kernel

of the model whenever the validation loss increases, from K number of epochs. In the Keras API, we are supplied with several callbacks that can be used in the training phase. Another possible callback is the *ModelCheckpoint* callback, that allow us to automatically save the model in every epoch that the validation accuracy rises. Even though the purposes of these two callbacks is the same, in the literature we see that using the first callback represents better results.
In terms of results, with this new enhanced architecture, the model reached a validation accuracy of 92% and a validation loss of 14% after training with half of the dataset size.

3.5.6 Final Encoder-Decoder Architecture

The final encoder-decoder model was used in the inference phase and was created from the initial model, containing two separate parts, the encoder and decoder, named here as **finalEncoder** and **finalDecoder**.
The final Encoder is created by combining the initial encoder inputs with the encoder's outputs. In the final Encoder, the text sequences are encoded as we will see further on, in Section 3.5.7.
As for the final Decoder, we have a much more complex model. It has three input layers with a shape of 100, which are then connected to the initial decoder's embedding layer - the layer responsible for creating the word embedding vectors as shown in Section 3.5.4. Similarly to the initial model, used for training, in this model we also append the Attention layer to aid the model in the inference phase. The output layer, is a *Softmax dense* layer that generates a probability distribution over the target vocabulary. This probability distribution is very important in the summary generation phase, as we will explain in the next section.

3.5.7 Summary Generation

The summary generation phase begins after we have trained our encoder-decoder with a sufficient number of instances from our dataset. We have combined our encoder-decoder with a simple algorithm to generate the summary words. The model is mainly used to understand how the sentences from the text relate and how it can use this knowledge to compute new sentences.
Given the raw document text, the algorithm begins by using our final Encoder to compute the text predictions and return them as integer vectors. A target sequence is initialized with the **sostok** token index from the text tokenizer referred in Section 3.5.3, and it is used to store the indexes of the words generated by the algorithm. Until a stop condition is met we add tokens to our summary sequence. Each token is obtained from computing the predictions of the current target sequence with our final Decoder. The tokens are assigned to vectors of integers that represent the probabilistic distribution of the words in the text. The criteria by which the tokens are chosen is their probability in the text, generated by the final Decoder. The model uses the highest scoring tokens according to the probability distribution to aid the algorithm in generating similar words for the summary.
Each time a new token is sampled, the algorithm analyses it and asserts that it is not the

eostok token or the previously added token. This is done to prevent the model from continuously picking the previously added tokens, since they possessed the highest probability distribution values.
In some cases, the resulting summary is a combination of words and stop-words without any punctuation, which may be difficult to understand by the readers.

3.6 Neural network and encoder-decoder combined

In this section, we present an alternative approach that consists in combining our neural network with our encoder-decoder. Since the algorithms deal with specific and independent types of automatic text summarization, we can combine them and take advantage of both methodologies. By combining them, we end up with a more complete system that attempts to solve many problems associated with text summarization.
The way the summaries are created with this approach is simple. We begin by executing the neural network algorithm, in which we analyze the features of the text, create the data to feed the neural network and pick the best sentences. They are then compressed in 20% for an initial summary. This compressed text is used as input for the encoder-decoder, that generates the words for the summary according to their probability distributions and agglomerates them into the final summary.
In the next section we explain how each algorithm can be executed.

3.7 Program Execution

During the practical phase of this thesis, all the code was implemented without the need for a Graphical User Interface (GUI), so it can be executed with a simple command line or through an Integrated Development Environment (IDE). Before executing the program, all the necessary libraries must be installed correctly as well as the Python programming language. In the beginning of the program's execution, the user is prompted with some guides which give a simple introduction to the program and explains the two possible paths that the user may take: the first being to train the deep learning models with new data and the second to simply create the summaries.
In one hand, if the user picks the first path, it has the possibility of training either the neural network and/or the encoder-decoder, with data from CSV files. These files must have at least the document text. In the case of our neural network summarizer, the document title and one reference summary is needed to efficiently train the model. If no title or reference summary is provided, the algorithm will generate one for the user but it will most certainly decrease the model's performance. As for the encoder-decoder, it only needs the document text.
After loading the files, the user is asked which model he wants to train. If he desires to train the neural network then the document's features are calculated and immediately sent to the model. When the training ends, the models are saved in the program's model's folder. In both algorithms, the user may define the number of epochs of the training process.
In the other hand, if the user takes the second path, it is asked to paste the paths of the

document text, title and one reference summary. If no reference summary is provided, the neural network summarizer will not be able to calculate the features related to the reference summary and will perform worse. Additionally, all the generated summaries, from each algorithm, are not be evaluated and the results are not be printed in the screen.
Next, the user is asked which of the summarizers he wants to execute. The user must choose at least one algorithm to execute. If the chooses all of them, he has the possibility of combining them all in the end. If the user decides to execute the neural network summarizer, then after computing the text features the user is prompted with a compression rate and then proceed in calculating both features and predictions. With the computed predictions the model picks the sentence summaries and displays it in the screen. Following is the *Term-Frequency* summarizer. For this summarizer to work, the user must fill in the compression rate for the desired number of sentences he wants. Next is the *TextRank* summarizer. Similarly to the previous algorithms, the user is prompted with the percentage of sentences he wants the summary to have. In the case of the encoder-decoder, it simply calculates the summary and displays it in the screen. If he chooses to combine the summarizers, the program re-executes all of them. It begins by re-computing the neural network's summary and send it to the term-frequency summarizer, that in turn returns its summary to the text-rank summarizer and finally it sends the created summary to the encoder-decoder to create its abstractive summary.
After each algorithm's execution, the evaluation results are provided in the screen if the user has inserted a reference summary.

3.7.1 Example Execution

Below we present one of DUC 2001's edition document of 400 characters, including a part of the document text, the title and one of three reference summaries provided.
Immediately below, we also present each summary created from this document with our summarizers. In the summaries that require a compression rate, we used the value of 20%, since it is the standard value according to the literature.

Document title
Hurricane insurers expect record claims.

Piece of the document text

US INSURERS expect to pay out an estimated Dollars 7.3bn (Pounds 3.7bn) in Florida as a result of Hurricane Andrew - by far the costliest disaster the industry has ever faced. The figure is the first official tally of the damage resulting from the hurricane, which ripped through southern Florida last week. In the battered region it is estimated that 275,000 people still have no electricity and at least 150,000 are either homeless or are living amid ruins. President George Bush yesterday made his second visit to the region since the hurricane hit. He pledged the government would see through the clean-up 'until the job is done'. Although there had already been some preliminary guesses at the level of insurance claims, yesterday's figure comes from the Property Claims Services division of the American Insurance Services Group, the property-casualty insurers' trade association. It follows an extensive survey of the area by the big insurance companies. Mr Gary Kerney, director of catastrophe services at the PCS, said the industry was expecting about 685,000 claims in Florida alone. It is reckoned the bulk of the damage - over Dollars 6bn in insured claims - is in Dade County, a rural region to the south of Miami. However, the final cost of Hurricane Andrew will be higher still. Yesterday's estimate does not include any projection for claims in Louisiana, which was also affected by the storm, although less severely than Florida. An estimate of the insured losses in this second state will be released later this week.

Reference summary

Florida's losses from Hurricane Andrew for which US insurers expect to pay 7.3B make it the most costly insured catastrophe in the US, even before the Andrew claims from Louisiana are tallied. In southern Florida at least 150,000 residents are homeless or living amid ruins, and nearly twice that number still have no electricity. President Bush made his second trip to the devastated area and pledged the government would stay with the clean-up until completed. Andrew has exceeded the insurance costs of Hurricane Hugo, the Oakland fire and tornado and hail damages. The insurance industry should have adequate reserves to cover the losses. Wall Street has remained calm.

Each generated summary is presented below:

Neural network summary with 20% compression rate

US INSURERS expect to pay out an estimated Dollars 7.3bn (Pounds 3.7bn) in Florida as a result of Hurricane Andrew - by far the costliest disaster the industry has ever faced. He pledged the government would see through the clean-up 'until the job is done'. By contrast, insurance claims resulting from the Los Angeles riots earlier this year - the most expensive civil disturbance in the US - totalled just Dollars 775m.

Term-Frequency summary with 20% compression rate

This easily exceeds the record Dollars 7.6bn of catastrophe losses seen in 1989, when the industry paid out on both Hurricane Hugo and the Loma Prieta earthquake in California. By contrast, insurance claims resulting from the Los Angeles riots earlier this year - the most expensive civil disturbance in the US - totalled just Dollars 775m. US INSURERS expect to pay out an estimated Dollars 7.3bn (Pounds 3.7bn) in Florida as a result of Hurricane Andrew - by far the costliest disaster the industry has ever faced. Mr Gary Kerney, director of catastrophe services at the PCS, said the industry was expecting about 685,000 claims in Florida alone.

Text-Rank summary with 20% compression rate

US INSURERS expect to pay out an estimated Dollars 7.3bn (Pounds 3.7bn) in Florida as a result of Hurricane Andrew - by far the costliest disaster the industry has ever faced. This easily exceeds the record Dollars 7.6bn of catastrophe losses seen in 1989, when the industry paid out on both Hurricane Hugo and the Loma Prieta earthquake in California. But on the Florida losses alone, Hurricane Andrew becomes the most costly insured catastrophe in the US. Yesterday's estimate does not include any projection for claims in Louisiana, which was also affected by the storm, although less severely than Florida.

Encoder-Decoder summary

coast it the such faced hail severely second 685 region later paid visit louisiana resulting residents their disaster record both becomes rises although totalled resulting bulk with association completed ruins mr 'until pledged tornadoes cover extensive ever job leaves also number week recently was projection recently than official over than 'until most 9bn does firming both losses thought added association 3b loma insurers series battered florida's pcs california been 6bn a week tornadoes pcs paid 775m civil government remained since official result visit facing expect street than costliest government expected riots los had rural living series later hit insurers riots people follows made kerney adequate us 'until either damages figure paid remained hurricane or bush figure number through.

Combined summary from the neural network and encoder-decoder summarizers

job us see clean which at losses the up up claims his done' make expect no faced totalled industry would fire claims catastrophe result cover no an no nearly it it up costs costliest by disturbance trip 150 riots disaster adequate even expect expect area it area costs or bush have through 7 a this with 7 catastrophe 3.

Chapter 4

Evaluation Measures and Results

The task of automatically generating summaries does not only comprise creating text summaries, but also evaluating how good they are. It is a demanding process, which requires a lot of conditions to be met, such as readability, domain knowledge, compression rate, quantity and quality of the information, and the existence of a human jury to evaluate the summary's overall quality. So far, there is still not an exact definition of what is the ideal summary for a given text.
As we previously saw in Section 2.4, several evaluation measures are available nowadays, some are considered more important than others. Evaluating an automatic summarization system means analysing the summary created by the system with regards to one or more reference summaries. In order to accomplish this task, one must have access to the reference summaries. Much like the summarization datasets presented before (Newsroom, WikihowAll and Billsum) there are others with the same purpose, that also provide the original documents and respective reference summaries. Some of these datasets are from conferences, such as DUC [170], others are from individual academic projects, such as *TeMário* [171] (for Portuguese).
Since DUC is specifically used for creating and evaluating automatic text summaries in conferences, we decided to use it for testing our system. The main language present in the DUC texts is English, which is not a problem for our summarizers.
In this dataset, we have documents from editions that span from 2001 to 2007. We used the entire 2001's edition, due to the fact that it is structured in the most simple way: in each folder, we have one original document text and three reference summaries, made by three different individuals. From this year edition, we tested our algorithms with all the documents of 400 characters. In the next section, we will explain how we evaluated the summarizers.

4.1 Evaluation Measures

As we have seen in Section 2.4, there are plenty of strategies and measures to be used, but some are considered more important than others in the scientific community. Groups of measures such as text-quality or task-based measures, are better used by human evaluators, since they analyse specific details in the text, for example the gramaticality, redundancy, reference clarity, coherence and structure of the text.
We decided to evaluate the summarizers with different measures and compare the results. **ROUGE** and **Bidirectional Encoder Representations from Transformer (BERT)** scores had the most important roles, as they are widely used in the field and represent the most accurate results. Additionally **BERT** has been showing great results in evaluating abstractive summaries [52], which is an important aspect for evaluating the summaries of our

Encoder-Decoder. Besides these two measures, we also used some standard similarity measures such as **Jaccard** and **Cosine** similarities [172].
With ROUGE one can evaluate a text summary in three different levels: ROUGE-1, ROUGE-2 and ROUGE-L. In ROUGE-1 we evaluate the overlap of uni-grams between our summary and the reference summary, while ROUGE-2 evaluates the overlap of bi-grams between the system and reference summaries. As for ROUGE-L, it evaluates the longest matching sequence of words. It does not require consecutive matches, instead it requires in-sequence matches that reflect sentence level word order. As opposed to ROUGE-1 and ROUGE-2, ROUGE-L does not require a predefined n-gram length.
With the BERT score we evaluate a generated summary with regards to a reference summary in three related measurements: precision, recall and f-score. In the BERT score, we use pre-trained BERT contextual embeddings to compute the similarity of two summaries as a sum of the cosine similarities between their tokens embeddings. It solves some problems faced by other N-gram-based metrics (ROUGE or Bilingual Evaluation Understudy (BLEU)) such as the penalization of semantically similar phrases differing only on the surface from the reference sentences. Also, n-gram models fail to capture distant dependencies and penalize semantically-critical ordering change.
The *Jaccard* similarity, however, is classified as token-based measure, so the goal is to find the similar tokens in both candidate and reference summaries. The more common tokens, the more similar the summaries would be.
Besides these measures, two other string similarity measures were initially idealized:

1. **BLEU** - evaluates automatic generated translations by comparing the matching n-grams from both original and computed translations. Having a score of 0, means that the candidate and references have zero matching n-grams, while having a score of 1, means that all the n-grams match. It conveys four important advantages: fast, inexpensive, easy to understand and language independent. In order to have a score of 1 means that both generated and reference summaries would have to be entirely equal[173];

2. **Levenstein similarity** - counts the minimum number of transformations necessary to convert one string into the other, namely insertions, deletions and substitutions of words. The lower the value, the closer are two strings (summaries in our case).

Using these measures for our problem is impractical, because when we create summaries by hand or with the help of computer algorithms there is a wide variety of words that can be used to create the summary. Multiple different summaries can be created for the same document, which means that when using these measures to analyse the summaries with the reference summaries, the results would be very low, even in the cases where we have very semantically similar summaries.

4.2 Results

In order to evaluate the results of our summarizers, we could have used the Newsroom dataset, since we have access to the entire news articles and one reference sum-

mary per document, as well the information available in the official repository `http://lil.nlp.cornell.edu/newsroom/evaluate/index.html`, that could be used to compare our results with the results obtained by other summarizers in this dataset.
Even though this dataset had these advantages, we decided to test the summarizers with the DUC summarization dataset. The reason was the fact that DUC is an official dataset which has data from text summarization conferences and that it supplies the users with the entire documents and three different reference summaries per document, instead of just one, as seen in Section 3.7.1. In our work, we tested the summarizers with a portion of DUC's 2001 edition, more specifically the 400 words documents, totaling ... documents.
With regards to the summarizers, except the Encoder-Decoder, the chosen compression rate used was 20%. There is not a standard value for the compression rate, any value can be used as long as it is not zero or equal to the original text length. Depending on the value chosen, the summary differs in terms of size and also the amount of information retained.

4.2.1 Results for the Term-Frequency summarizer

In this first summarizer, as presented in Section 3.2, one needs to define a compression rate that is used to filter the text sentences.
In Table 4.1 presented below, we have the evaluation results from this summarizer, with all the documents and respective reference summaries. We decided to calculate the average values of each measure.

Table 4.1: Results for the Term-Frequency summarizer with 20% compression rate

ROUGE results			
Measures	ROUGE-1	ROUGE-2	ROUGE-L
Precision	0.30	0.05	0.21
Recall	0.23	0.04	0.17
F-score	0.24	0.04	0.17

BERT results		
Precision	Recall	F-score
0.83	0.83	0.83

Similarity results	
Jaccard	Cosine
0.68	0.96

From Table 4.1, we can see that we needed to improve our summarizer since the results were not outstanding when comparing them to other extractive methods proposed in the scientific community [174] [175] [176].
Term-frequency approaches only analyse how frequent is each text element and do not analyse how they are related. More frequent elements are assigned higher scores while less frequent elements are assigned lower scores. This means that important elements such as acronyms, numbers and temporal expressions, etc are ignored and not added to the summary, while less important but more frequent elements are added to the summary.

The summarizer could have got better results if it had a mechanism of evaluating how the words are related.
In the next section we will present the results obtained with the TextRank algorithm.

4.2.2 Results with TextRank and Word Embeddings

A typical Text-Rank algorithm is in general a more complex algorithm than a term frequency-based algorithm. By representing the text's elements as nodes from a graph, one can understand how they relate. Representing a potential solution for text summarization, it still has its limitations and can be improved. So, our goal was to adapt the original algorithm with word embeddings and see if the results improved. The results with the chosen compression rate were as follows:

Table 4.2: Results for the TextRank summarizer with 20% compression rate

ROUGE results			
Measures	ROUGE-1	ROUGE-2	ROUGE-L
Precision	0.30	0.04	0.21
Recall	0.23	0.03	0.16
F-score	0.24	0.04	0.17

BERT results		
Precision	Recall	F-score
0.83	0.82	0.83

Similarity results	
Jaccard	Cosine
0.66	0.96

From Table 4.2 the results of our summarizer were similar to the ones from the previous summarizer, but they could have been better. The fact that we used pre-trained word embedding vectors could have reduced the model performance. Instead we could have trained our own word embedding vectors but that would take a lot of time to be done.
In terms of comparison to other extractive approaches, we see that our summarizer needed improvements. Most extractive approaches had their ROUGE results higher than ours, being 0.43 the minimum for ROUGE-1 and 0.16 for ROUGE-2 [174]. As for the original TextRank algorithm, it only provided results for DUC 2002 dataset, therefore there is no possible comparison between the two summarizers [103].
Even though we have solved some of the issues from the previous Term-Frequency summarizer, the summaries could have been better. In this approach, we only evaluated the relationships between the text elements and not the degree of importance they have for the subject.
The following section will present the results obtained with our neural network features summarizer that attempts to obtain better results.
Next section will present and explain the results from Deep Neural Network summarizer.

4.2.3 Deep Neural Network Summarizer Results

In our neural network summarizer, we attempt to improve the results obtained with the previous algorithms. By analysing the text's most important features, the algorithm is capable of choosing which sentences and words have more weight than others.
In each DUC document, title and respective reference summaries, the features scores were calculated and sent to the model. In an initial phase, we evaluated how the model would perform with each test document. In average, the validation accuracy and validation loss were 89% and 10% respectively for the 10 documents. These values were fairly good, since we only trained our model with 2 gigabyte of data.
Having finished evaluating the model's performance with the test data, the model began computing the predictions and picking the correct sentences for the summary.
Table 4.3 presents the results obtained with our neural network algorithm.

Table 4.3: Results for the Neural network summarizer with 20% compression rate

ROUGE results			
Measures	ROUGE-1	ROUGE-2	ROUGE-L
Precision	0.28	0.05	0.21
Recall	0.16	0.03	0.13
F-score	0.20	0.04	0.15

BERT results		
Precision	Recall	F-score
0.71	0.82	0.71

Similarity results	
Jaccard	Cosine
0.56	0.82

As we can see from the results, the algorithm did not choose the most correct sentences from the documents for creating the summaries, so the results, specially ROUGE, were inferior to many approaches from the scientific community [174]. One plausible explanation is the lack of training of the model. Deep learning models need data to be trained with, the more data they receive, the more efficient they are. Perhaps our model was not trained enough, making it pick the wrong sentences for the summaries.
One interesting aspect to mention is that in our first approach as referred in Section 3.4.6, when we filtered the predictions above 0.5, some results obtained with ROUGE were in fact higher than some of ones presented with this second approach. The problem with picking only the sentences whose predictions are above 0.5 was the fact that in some cases, since the model was not trained enough, it failed to pick at least one sentence from the documents and generate summaries.
Another reason for the lower performance of our algorithm may be the existence of errors in the training data. Having errors, makes it more complicated for the model to train and learn the features, hence resulting in a lower performance. Also when computing the sentences features, like the previous *TextRank* algorithm, we had to use pre-trained trained word em-

beddings, since the other more powerful ones had problems with some vectors. By using these less trained vectors, most of the similarity related features were miscalculated.
Additionally, in some documents that we used for training, had Arabic characters or other elements that had to be completely removed. Even though the number of these cases is reduced, it still may impact the model's performance. Some of them may be important and by removing them we are removing essential information from the important sentences for the summary and induce the model with errors.
In overall, some potential improvements could be made in the pre-processing and the features calculations. Features such as sentence polarity may mislead the model in choosing wrong sentences. When a sentence has a very high or low sentence polarity value means that the sentence has extreme sentiments, very positive or very negative, but that may not mean that a given sentence is in fact important.
In the next section, we will present the results obtained with our Encoder-Decoder algorithm.

4.2.4 Encoder-Decoder Summarizer's Results

Ultimately we have the encoder-decoder's results. It was by far our hardest algorithm to understand, adapt and use for creating summaries. In the initial implementation, the algorithm was only capable of creating short review summaries with three words at most. With some improvements, we were capable of adapting it to more complex scenarios, where the documents had lengths between 100 to 400 or more words, however, most summaries created still had structural and coherence problems, since they were merely combinations of words without any punctuation.
Below are the results obtained with this algorithm:

Table 4.4: Results for the Encoder-Decoder summarizer

ROUGE results			
Measures	ROUGE-1	ROUGE-2	ROUGE-L
Precision	0.59	0.007	0.49
Recall	0.31	0.005	0.24
F-score	0.36	0.006	0.30

BERT results		
Precision	Recall	F-score
0.77	0.79	0.78

Similarity results	
Jaccard	Cosine
0.59	0.95

From Table 4.5, we see that our Encoder-Decoder got fairly good results with ROUGE, specially ROUGE-1 and ROUGE-L, but lower results in ROUGE-2. In ROUGE-1, we evaluate sequences of uni-grams, while in ROUGE-L we evaluate the longest matching sequences. These results may mean that in terms of uni-grams, the summarizer is able to create the right individual words for the summaries but when analysed as bi-grams, we see that in most of the

cases the words are not related. The BERT score is very useful in this case, since it can aid the evaluation process by using the word embeddings and understand how the words are related. So, given the previous BERT results we see that we got interesting results in the BERT score metrics, which implies that both generated and reference summaries have similar meanings. In order to compare our results with the ones from the scientific community, we could only analyse the results obtained by extractive approaches, because for this DUC 2001 dataset there was only one abstractive approach created and according to the literature, the evaluation was not done with ROUGE [177]. So, according to the state of the art extractive approaches, we see that we similar results in ROUGE-1, since theirs were between 0.43 and 0.48. In terms of ROUGE-2, they presented much higher values because by calculating extractive summaries, the elements are directly extracted from the original documents, hence the bi-grams are more easily related [174] [175].

4.2.5 Results from combining the neural network and the encoder-decoder

As previously explained in Section 3.6, we attempted to improve our results, mainly in ROUGE and BERT, by providing a solution that incorporates our most complex extractive summarizer with our abstractive summarizer. As illustrated in the next table, we can see the results obtained by combining the summarizers into a single one:

Table 4.5: Results for the combination of the neural network and the encoder-decoder

ROUGE results			
Measures	ROUGE-1	ROUGE-2	ROUGE-L
Precision	0.60	0.009	0.53
Recall	0.30	0.005	0.23
F-score	0.36	0.006	0.29

BERT results		
Precision	Recall	F-score
0.78	0.79	0.78

Similarity results	
Jaccard	Cosine
0.57	0.95

In a quantitative evaluation, we can see that the results were pretty similar to the ones obtained with our Encoder-Decoder. In general, there was only a slight increase in ROUGE-1. This may be related to the fact that the last summarizer to be executed was our Encoder-Decoder. In terms of a qualitative evaluation, since the output summaries come from the Encoder-Decoder, they also have a lack in punctuation, coherence and cohesion.
As for the comparison with other state of the art approaches, there is no difference to what was presented in the previous section [174] [175] [176].

4.2.6 Results of all the summarizers

In this final section, we present a simple table with all the main metric results from our summarizers.
From analysing all the values, we see that our Encoder-Decoder and our combined summarizer were our best approaches, having reached our best results with ROUGE.

Table 4.6: Results from all the summarizers

MeasureSummarizer	Term-Frequency	TextRank	Neural Network	Encoder-Decoder	Neural network with Encoder-Decoder
ROUGE-1 (F-score)	0.24	0.24	0.20	0.36	0.36
ROUGE-2 (F-score)	0.04	0.04	0.04	0.006	0.006
ROUGE-L (F-score)	0.17	0.17	0.15	0.30	0.29
BERT (F-score)	0.83	0.83	0.71	0.78	0.78
Jaccard Similarity	0.68	0.66	0.56	0.59	0.57
Cosine Similarity	0.96	0.96	0.82	0.95	0.95

Chapter 5

Conclusions and Future Work

In this final chapter, we will present the main conclusions regarding the work done in this thesis, as well as presenting some ideas to improve the results obtained.
The core objective was to find new and interesting ideas to solve the problem of excessive and unnecessary data in text documents, independent of the type of the documents. Having unnecessary data in the text hinders our learning curve and there is a need to filter the text data and keep only the essential parts so the documents are easier to process.
Text summarization has been in development for years, having some pauses in the middle, which made the research slow down. In the beginning, there was not sufficient computational power to achieve good results. Today, we reached a state in which we have bigger processing power, but newer and more complex approaches, like those inspired by machine/deep learning, require way more memory and faster Central Processing Unit (CPU)s/Graphic's Processing Unit (GPU)s to process data. Having to continuously feed the algorithms with more data, makes them highly costly and limited in terms of the text domain level. This is why some authors return to older and less expensive approaches, with the goal of improving them, but with the cost of creating lower textual quality summaries.
We researched and implemented some of the most powerful techniques, with the goal of combining them into a fully capable summarizing system. Our first technique computes relevancy based on term-frequency. The text was compressed according to a given rate, that allowed us to filter the most frequent sentences and combine them into a summary.
In a different approach, a graph was created to evaluate how words relate to each other and word embedding vectors were used. Similarly to the first approach, we also filtered the text sentences with a compression rate, meaning that only the most scored sentences were picked for the summary.
Besides these two approaches, we created a more complex summarizer that combined feature analysis with deep learning. In this system, we analysed the document's text in order to understand how it is structured: what are the most important elements, what sentiments the sentences convey, what topics are present and determine which sentences need to be added to the summary. Having analysed the text according to the different features, we then created and trained a deep learning model to predict which document's sentences had to be combined into a full summary.
Next, we proceeded into a new strategy, which was attempting to create an abstractive summarizer. We used and adapted the code from an open-source repository that creates a sequence-to-sequence algorithm for generating small reviews summaries. Our job was to improve it and use it with real world scenarios, where the documents are much bigger. This last summarizer was by far our hardest summarizer to implement. We did improve the way it creates summaries and adapt it to any kind of document we want, however the resulting summaries had a lack of structure and punctuation.

In the end, we combined all the summarizers into one, to see if the results improved.
The evaluation of each summarizer was an automatic procedure, in which we analysed the generated summaries and compared them with the reference summaries, with the help of state of the art measures, such as ROUGE. The measures however, only evaluated the structure and content of the summaries, so we performed an additional evaluation of how coherent and grammatically correct the summaries were.
The results provided were obtained by computing the average values of each summarizer, which means that there were some cases whose results were very interesting while others were less interesting. In the case of our extractive summarizers, when analysing the same document the resulting summary is always the same independently of any circumstance, however in the cases of our Encoder-Decoder and the combined summarizer, the summary is different each time we execute the algorithms. This small detail made a big impact in the results we presented, because depending on the summaries generated, the results are better or worse.
When comparing our summarizers with others from the scientific community, we saw that ours were in general less efficient. Our extractive summarizers were capable of creating fully coherent and grammatically correct summaries, but they were a little different from the reference summaries, both in a quantitative and qualitative way. This issue may be related to two different scenarios: one being the lack of efficiency of the summarizers and the other being the inaccuracy of the evaluation measures. Our Encoder-Decoder and the combined summarizer could also possessed better results. When comparing them with the abstractive approaches present in the scientific community, we saw that these approaches only presented results with more DUC dataset versions than ours. There was only one that was tested with the same dataset as ours, but it did not present any ROUGE results, so there was not way of evaluating our abstractive summarizers but to compare their results with the ones from extractive approaches.
Even though most of our approaches calculated the summaries in a matter of seconds, the results could have been better. Our best approaches, the Encoder-Decoder and the combined summarizer, got results relatively close to some extractive approaches used with the same DUC 2001 dataset.
Both neural network summarizer and the Encoder-Decoder could be further improved by changing the architecture, adding more regularization mechanisms, training with more and better data and plot the training process in order to detect possible existing issues.
The Encoder-Decoders implementation was changed a couple of times, until we finally figured out the behavior of the algorithm. The architecture and the summary generation mechanism were very hard to understand. When the problem was surpassed, we were able to begin training the model with our new dataset and generate some interesting summaries. In some cases they were pretty similar to most reference summaries and despite having no punctuation, they easy to understand.
In terms of general internal improvements to be done, we think that in the case of the neural network algorithm, if no title is provided at some point, we can create our own more powerful title generator and aid the model when it needs to compute the title's related features. Some

features can also have more weight than others, since as we have seen before some of them are useless in certain cases. The algorithms that use word embeddings, such as the TextRank and neural network summaries, may need more powerful word embedding vectors or we need to train our own..

In terms of external improvements, we think that the field of automatic text summarization still needs a lot of research and development to demonstrate outstanding results, even though some extractive approaches already present good results from a quality point of view.

The scientific community can also research and provide better ways of generating more training data for deep learning summarizers. In most cases, like ours for instance, there is a need to generate the training data for the summarizers, which is in fact a time consuming task.

I have learned a lot throughout the hard work developed in this thesis. It was difficult most of the times. The state of the art in text summarization is very extensive and there are many complex aspects and theory to learn about. The field itself is under research and development and it still needs to be improved to a certain degree. In terms of the state of the art research, we think we covered the majority of the state of the art with regards to text summarization. I think I could have done a better job in finding new efficient ways of solving the problem of text summarization. Time was an important factor and in most cases there was none to work in this thesis. Due to the fact that I kept most of time on the computer during the work days, I got very tired and the little free time I had, was used to recover both physically and mentally. If I had managed my free time a bit better, both in the work days and also the weekends, I could have finished much sooner and with better results.

Bibliography

[1] K. S. Jones, "Automatic summarising: The state of the art," *Information Processing & Management*, vol. 43, no. 6, pp. 1449–1481, 2007.

[2] O. Tas and F. Kiyani, "A survey automatic text summarization," *PressAcademia Procedia*, vol. 5, no. 1, pp. 205–213, 2007.

[3] K. Kaikhah, "Automatic text summarization with neural networks," in *2004 2nd International IEEE Conference on'Intelligent Systems'. Proceedings (IEEE Cat. No. 04EX791)*, vol. 1. IEEE, 2004, pp. 40–44.

[4] J. Steinberger and K. Jezek, "Using latent semantic analysis in text summarization and summary evaluation," *Proc. ISIM*, vol. 4, pp. 93–100, 2004.

[5] P. D. Patil and N. Kulkarni, "Text summarization using fuzzy logic," *International Journal of Innovative Research in Advanced Engineering (IJIRAE)*, vol. 1, no. 3, 2014.

[6] G. Erkan and D. R. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization," *Journal of artificial intelligence research*, vol. 22, pp. 457–479, 2004.

[7] G. Carenini, J. C. K. Cheung, and A. Pauls, "Multi-document summarization of evaluative text," *Computational Intelligence*, vol. 29, no. 4, pp. 545–576, 2013.

[8] N. Moratanch and S. Chitrakala, "A survey on abstractive text summarization," in *2016 International Conference on Circuit, power and computing technologies (ICCPCT)*. IEEE, 2016, pp. 1–7.

[9] J. Steinberger and K. Ježek, "Evaluation measures for text summarization," *Computing and Informatics*, vol. 28, no. 2, pp. 251–275, 2012.

[10] D. R. Radev, E. Hovy, and K. McKeown, "Introduction to the special issue on summarization," *Computational linguistics*, vol. 28, no. 4, pp. 399–408, 2002.

[11] P. B. Baxendale, "Machine-made index for technical literature—an experiment," *IBM Journal of research and development*, vol. 2, no. 4, pp. 354–361, 1958.

[12] H. P. Edmundson, "New methods in automatic extracting," *Journal of the ACM (JACM)*, vol. 16, no. 2, pp. 264–285, 1969.

[13] E. Hovy, "Text summarization - chapter 32," pp. 583 – 593, 1998.

[14] M. Jill Salahub, Steven Reid, "Types of summaries," 2019. [Online]. Available: https://writing.colostate.edu/guides/teaching/summaryresponse/summary.cfm

[15] W. Fan, L. Wallace, S. Rich, and Z. Zhang, "Tapping the power of text mining," *Communications of the ACM*, vol. 49, no. 9, pp. 76–82, 2006.

[16] D. Das and A. Martins, "A survey on automatic text summarization," 12 2007.

[17] S. Ruder, "Types of summaries." [Online]. Available: http://nlpprogress.com/

[18] A. Vashisht, "Edmundson heuristic method for text summarization." [Online]. Available: https://iq.opengenus.org/edmundson-heuristic-method-for-text-summarization/

[19] K. S. Jones *et al.*, "Automatic summarizing: factors and directions," in *Advances in automatic text summarization*. MIT press Cambridge, Mass, USA, 1999, no. 1, pp. 1–12.

[20] M. Maybury, *Advances in automatic text summarization*. MIT press, 1999.

[21] H. Eduard and C.-Y. Lin, "Automated text summarization and the summarist system," in *Proceedings of a workshop on held at Baltimore*, 1998.

[22] C.-Y. Lin and E. Hovy, "The automated acquisition of topic signatures for text summarization," in *Proceedings of the 18th conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, 2000, pp. 495–501.

[23] T. Fukushima, T. Ehara, and K. Shirai, "Partitioning long sentences for text summarization," *Journal of Natural Language Processing*, vol. 6, pp. 131–147, 01 1999.

[24] T. Strzalkowski, J. Wang, and B. Wise, "A robust practical text summarization," in *Proceedings of the AAAI Symposium on Intelligent Text Summarization*, 1998, pp. 26–33.

[25] C.-Y. Lin, "Automated text summarization in summarist," 05 2001.

[26] K. Knight and D. Marcu, "Statistics-based summarization-step one: Sentence compression," *AAAI/IAAI*, vol. 2000, pp. 703–710, 2000.

[27] H. Jing and K. 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*, 1999, pp. 129–136.

[28] M. J. Witbrock and V. O. Mittal, "Ultra-summarization (poster abstract) a statistical approach to generating highly condensed non-extractive summaries," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, 1999, pp. 315–316.

[29] E. Lloret and M. Sanz, "Text summarisation in progress: A literature review," *Artif. Intell. Rev.*, vol. 37, pp. 1–41, 04 2012.

[30] K. S. Thakkar, R. V. Dharaskar, and M. Chandak, "Graph-based algorithms for text summarization," in *2010 3rd International Conference on Emerging Trends in Engineering and Technology*. IEEE, 2010, pp. 516–519.

[31] D. Radev, A. Winkel, and M. Topper, "Multi document centroid-based text summarization," in *ACL 2002*. Citeseer, 2002.

[32] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American society for information science*, vol. 41, no. 6, pp. 391–407, 1990.

[33] M. G. Ozsoy, F. N. Alpaslan, and I. Cicekli, "Text summarization using latent semantic analysis," *Journal of Information Science*, vol. 37, no. 4, pp. 405–417, 2011.

[34] J. L. Neto, A. A. Freitas, and C. A. Kaestner, "Automatic text summarization using a machine learning approach," in *Brazilian symposium on artificial intelligence*. Springer, 2002, pp. 205–215.

[35] P.-E. Genest and G. Lapalme, "Absum: a knowledge-based abstractive summarizer," *Génération de résumés par abstraction*, vol. 25, p. 26, 2013.

[36] U. Hahn and I. Mani, "The challenges of automatic summarization," *Computer*, vol. 33, no. 11, pp. 29–36, 2000.

[37] V. Gupta and G. S. Lehal, "A survey of text summarization extractive techniques," *Journal of emerging technologies in web intelligence*, vol. 2, no. 3, pp. 258–268, 2010.

[38] L. Antiqueira, O. N. Oliveira Jr, L. da Fontoura Costa, and M. d. G. V. Nunes, "A complex network approach to text summarization," *Information Sciences*, vol. 179, no. 5, pp. 584–599, 2009.

[39] N. Moratanch and S. Chitrakala, "A survey on extractive text summarization," in *2017 international conference on computer, communication and signal processing (ICCCSP)*. IEEE, 2017, pp. 1–6.

[40] J. P. Verma and A. Patel, "Evaluation of unsupervised learning based extractive text summarization technique for large scale review and feedback data," *Indian Journal of Science and Technology*, vol. 10, no. 17, pp. 1–6, 2017.

[41] J. Peralta, "Text preprocessing in python | set – 1." [Online]. Available: https://www.geeksforgeeks.org/text-preprocessing-in-python-set-1/

[42] ——, "Text preprocessing in python | set 2." [Online]. Available: https://www.geeksforgeeks.org/text-preprocessing-in-python-set-2/?ref=rp

[43] V. T. Chou, L. Kent, J. A. Góngora, S. Ballerini, and C. D. Hoover, "Towards automatic extractive text summarization of a-133 single audit reports with machine learning," *arXiv preprint arXiv:1911.06197*, 2019.

[44] D. Monsters, "Text preprocessing in python: Steps, tools, and examples." [Online]. Available: https://medium.com/@datamonsters/text-preprocessing-in-python-steps-tools-and-examples-bf025f872908

[45] E. Loper and S. Bird, "Nltk: the natural language toolkit," *arXiv preprint cs/0205028*, 2002.

[46] J. Brownlee, "What are word embeddings for text?" [Online]. Available: https://machinelearningmastery.com/what-are-word-embeddings/

[47] ——, "What are word embeddings for text?" October 11, 2017. [Online]. Available: https://machinelearningmastery.com/what-are-word-embeddings/

[48] G. Rossiello, P. Basile, and G. Semeraro, "Centroid-based text summarization through compositionality of word embeddings," in *Proceedings of the MultiLing 2017 Workshop on Summarization and Summary Evaluation Across Source Types and Genres*, 2017, pp. 12–21.

[49] V. Phung and L. De Vine, "A study on the use of word embeddings and pagerank for vietnamese text summarization," in *Proceedings of the 20th Australasian Document Computing Symposium*, ser. ADCS '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2838931.2838935

[50] X. Rong, "word2vec parameter learning explained," *arXiv preprint arXiv:1411.2738*, 2014.

[51] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[52] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[53] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *Advances in neural information processing systems*, 2019, pp. 5754–5764.

[54] A. J. T. M. Piotr Bojanowski, Edouard Grave, "fasttext," August 18, 2016. [Online]. Available: https://research.fb.com/blog/2016/08/fasttext/

[55] C. C. Aggarwal and C. Zhai, *Mining text data*. Springer Science & Business Media, 2012.

[56] H. P. Luhn, "The automatic creation of literature abstracts," *IBM Journal of research and development*, vol. 2, no. 2, pp. 159–165, 1958.

[57] M. F. Porter *et al.*, "An algorithm for suffix stripping." *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[58] Snowball, "The lovins stemming algorithm." [Online]. Available: http://snowball.tartarus.org/algorithms/lovins/stemmer.html

[59] C. Moral, A. de Antonio, R. Imbert, and J. Ramírez, "A survey of stemming algorithms in information retrieval." *Information Research: An International Electronic Journal*, vol. 19, no. 1, p. n1, 2014.

[60] S. University, "Stemming and lemmatization." [Online]. Available: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

[61] L. Galambos, "Lemmatizer for document information retrieval systems in java," in *International Conference on Current Trends in Theory and Practice of Computer Science*. Springer, 2001, pp. 243–252.

[62] H. van Halteren and M. Rem, "Dealing with orthographic variation in a tagger-lemmatizer for fourteenth century dutch charters," *Language resources and evaluation*, vol. 47, no. 4, pp. 1233–1259, 2013.

[63] H. Paulussen and W. Martin, "Dilemma-2: a lemmatizer-tagger for medical abstracts," in *Proceedings of the third conference on Applied natural language processing*. Association for Computational Linguistics, 1992, pp. 141–146.

[64] R. S. Prasad, N. M. Uplavikar, S. S. Wakhare, V. Jain, T. Avinash *et al.*, "Feature based text summarization," *International journal of advances in computing and information researches*, vol. 1, 2012.

[65] J. L. Neto, A. D. Santos, C. A. Kaestner, N. Alexandre, D. Santos *et al.*, "Document clustering and text summarization," 2000.

[66] C. N. Silla, G. L. Pappa, A. A. Freitas, and C. A. Kaestner, "Automatic text summarization with genetic algorithm-based attribute selection," in *Ibero-American Conference on Artificial Intelligence*. Springer, 2004, pp. 305–314.

[67] A. Aristoteles, W. Widarti, and E. D. Wibowo, "Text feature weighting for summarization of documents bahasa indonesia by using binary logistic regression algorithm," *International Journal of Computer Science and Telecommunications*, vol. 5, no. 7, pp. 29–33, 2014.

[68] Y. J. Kumar, F. J. Kang, O. S. Goh, and A. Khan, "Text summarization based on classification using anfis," in *Asian Conference on Intelligent Information and Database Systems*. Springer, 2017, pp. 405–417.

[69] L. Suanmali, M. S. Binwahlan, and N. Salim, "Sentence features fusion for text summarization using fuzzy logic," in *2009 Ninth International Conference on Hybrid Intelligent Systems*, vol. 1. IEEE, 2009, pp. 142–146.

[70] D. Shen, J.-T. Sun, H. Li, Q. Yang, and Z. Chen, "Document summarization using conditional random fields." in *IJCAI*, vol. 7, 2007, pp. 2862–2867.

[71] K.-F. Wong, M. Wu, and W. Li, "Extractive summarization using supervised and semi-supervised learning," in *Proceedings of the 22nd international conference on computational linguistics (Coling 2008)*, 2008, pp. 985–992.

[72] D. Tsarev, M. Petrovskiy, and I. Mashechkin, "Supervised and unsupervised text classification via generic summarization," *International Journal of Computer Information Systems and Industrial Management Applications. MIR Labs*, vol. 5, pp. 509–515, 2013.

[73] T. Nomoto and Y. Matsumoto, "A new approach to unsupervised text summarization," in *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, 2001, pp. 26–34.

[74] M.-R. Amini and P. Gallinari, "The use of unlabeled data to improve supervised learning for text summarization," in *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, 2002, pp. 105–112.

[75] ——, "Automatic text summarization using unsupervised and semi-supervised learning," in *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 2001, pp. 16–28.

[76] P.-y. Zhang and C.-h. Li, "Automatic text summarization based on sentences clustering and extraction," in *2009 2nd IEEE international conference on computer science and information technology*. IEEE, 2009, pp. 167–170.

[77] F. Kyoomarsi, H. Khosravi, E. Eslami, P. K. Dehkordy, and A. Tajoddin, "Optimizing text summarization based on fuzzy logic," in *Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008)*. IEEE, 2008, pp. 347–352.

[78] J. L. Neto, A. D. Santos, C. A. Kaestner, N. Alexandre, D. Santos *et al.*, "Document clustering and text summarization," 2000.

[79] T. Yiu, "Understanding neural networks," Jun 2. [Online]. Available: https://towardsdatascience.com/understanding-neural-networks-19020b758230

[80] R. Singh, "Pruning deep neural networks." [Online]. Available: https://towardsdatascience.com/pruning-deep-neural-network-56cae1ec5505

[81] K. Svore, L. Vanderwende, and C. Burges, "Enhancing single-document summarization by combining ranknet and third-party sources," in *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, 2007, pp. 448–457.

[82] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics and intelligent laboratory systems*, vol. 39, no. 1, pp. 43–62, 1997.

[83] C. Sutton, A. McCallum *et al.*, "An introduction to conditional random fields," *Foundations and Trends® in Machine Learning*, vol. 4, no. 4, pp. 267–373, 2012.

[84] A. Prasad, "Conditional random fields explained." [Online]. Available: https://towardsdatascience.com/conditional-random-fields-explained-e5b8256da776

[85] D. A. S.Santhana Megala, Dr. A. Kavitha, "Text summarization system using fuzzy logic and conditional random field algorithm," 2015.

[86] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American society for information science*, vol. 41, no. 6, pp. 391–407, 1990.

[87] K. Ježek and J. Steinberger, "Automatic text summarization (the state of the art 2007 and new challenges)," in *Proceedings of Znalosti*, 2008, pp. 1–12.

[88] K. R. Patil *et al.*, "Automatic text summarization using pathfinder network scaling," 2007.

[89] Y. Gong and X. 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*, 2001, pp. 19–25.

[90] L. Sellberg and A. Jönsson, "Using random indexing to improve singular value decomposition for latent semantic analysis." in *LREC*. Citeseer, 2008, pp. 2335–2338.

[91] K. Bellare, A. D. Sarma, A. D. Sarma, N. Loiwal, V. Mehta, G. Ramakrishnan, and P. Bhattacharyya, "Generic text summarization using wordnet." in *LREC*, 2004.

[92] P. University, "About wordnet." [Online]. Available: https://wordnet.princeton.edu/

[93] Shodhganga, "Text summarization using fuzzy logic." pp. 84 – 87.

[94] T. Point, "Fuzzy logic - membership function." [Online]. Available: https://www.tutorialspoint.com/fuzzy_logic/fuzzy_logic_membership_function.htm

[95] L. Suanmali, N. Salim, and M. S. Binwahlan, "Fuzzy logic based method for improving text summarization," *arXiv preprint arXiv:0906.4690*, 2009.

[96] R. Ferreira, F. Freitas, L. de Souza Cabral, R. D. Lins, R. Lima, G. França, S. J. Simskez, and L. Favaro, "A four dimension graph model for automatic text summarization," in *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, vol. 1. IEEE, 2013, pp. 389–396.

[97] G. Salton, A. Singhal, M. Mitra, and C. Buckley, "Automatic text structuring and summarization," *Information processing & management*, vol. 33, no. 2, pp. 193–207, 1997.

[98] G. Erkan and D. Radev, "Lexpagerank: Prestige in multi-document text summarization," in *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, 2004, pp. 365–371.

[99] G. Erkan and D. R. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization," *Journal of artificial intelligence research*, vol. 22, pp. 457–479, 2004.

[100] J. Zhang, Y. Sun, H. Wang, and Y. He, "Calculating statistical similarity between sentences," *Journal of Convergence Information Technology*, vol. 6, no. 2, 2011.

[101] I. Mani, E. Bloedorn, and B. Gates, "Using cohesion and coherence models for text summarization," in *Intelligent Text Summarization Symposium*, 1998, pp. 69–76.

[102] R. Mihalcea, "Graph-based ranking algorithms for sentence extraction, applied to text summarization," in *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, 2004, pp. 170–173.

[103] R. Mihalcea and P. Tarau, "Textrank: Bringing order into text," in *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004, pp. 404–411.

[104] M. Litvak and M. Last, "Graph-based keyword extraction for single-document summarization," in *Proceedings of the workshop on Multi-source Multilingual Information Extraction and Summarization*. Association for Computational Linguistics, 2008, pp. 17–24.

[105] A. Dode and S. Hasani, "Pagerank algorithm."

[106] P. J.-J. Herings, G. Van Der Laan, and D. Talman, "The positional power of nodes in digraphs," *Social Choice and Welfare*, vol. 24, no. 3, pp. 439–454, 2005.

[107] K. Ganapathiraju, J. Carbonell, and Y. Yang, "Relevance of cluster size in mmr based summarizer: A report 11-742: Self-paced lab in information retrieval," 2002.

[108] A. Vashisht, "Using cosine-similarity to build a python text summarization tool." [Online]. Available: https://medium.com/@krause60/using-cosine-similarity-to-build-a-python-text-summarization-tool-d3c8228549bf

[109] D. Radev, A. Winkel, and M. Topper, "Multi document centroid-based text summarization," in *ACL 2002*. Citeseer, 2002.

[110] J. Carbonell and J. Goldstein, "The use of mmr, diversity-based reranking for reordering documents and producing summaries," in *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, 1998, pp. 335–336.

[111] D. Marcu, "From discourse structures to text summaries," in *Intelligent Scalable Text Summarization*, 1997.

[112] D. Radev, "A common theory of information fusion from multiple text sources step one: cross-document structure," in *1st SIGdial workshop on Discourse and dialogue*, 2000, pp. 74–83.

[113] Sciforce, "Towards automatic text summarization: Extractive methods." [Online]. Available: https://medium.com/sciforce/towards-automatic-text-summarization-extractive-methods-e8439cd54715

[114] ——, "Towards automatic text summarization - extractive methods." [Online]. Available: https://medium.com/sciforce/towards-automatic-text-summarization-extractive-methods-e8439cd54715

[115] T. Nomoto, "Bayesian learning in text summarization," in *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, 2005, pp. 249–256.

[116] D. Wang, S. Zhu, T. Li, and Y. Gong, "Multi-document summarization using sentence-based topic models," in *Proceedings of the ACL-IJCNLP 2009 conference short papers*. Association for Computational Linguistics, 2009, pp. 297–300.

[117] T. Amit, "Introduction to hidden markov models." [Online]. Available: https://towardsdatascience.com/introduction-to-hidden-markov-models-cd2c93e6b781

[118] D. Jurafsky and J. H. Martin, "Speech and language processing (draft)," *Chapter A: Hidden Markov Models (Draft of September 11, 2018). Retrieved March*, vol. 19, p. 2019, 2018.

[119] Y. Sun, H. Deng, and J. Han, "Probabilistic models for text mining," in *Mining text data*. Springer, 2012, pp. 259–295.

[120] J. M. Conroy and D. P. O'leary, "Text summarization via hidden markov models," in *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, 2001, pp. 406–407.

[121] P. Fung, G. Ngai, and C.-S. Cheung, "Combining optimal clustering and hidden markov models for extractive summarization," in *Proceedings of the ACL 2003 workshop on Multilingual summarization and question answering-Volume 12*. Association for Computational Linguistics, 2003, pp. 21–28.

[122] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[123] J. Morris and G. Hirst, "Lexical cohesion computed by thesaural relations as an indicator of the structure of text," *Computational linguistics*, vol. 17, no. 1, pp. 21–48, 1991.

[124] M. A. K. Halliday and R. Hasan, *Cohesion in english*. Routledge, 2014.

[125] R. Barzilay and M. Elhadad, "Using lexical chains for text summarization," *Advances in automatic text summarization*, pp. 111–121, 1999.

[126] P. M. Roget, "Roget's thesaurus of english words and phrases." [Online]. Available: https://www.gutenberg.org/cache/epub/10681/pg10681.txt

[127] Y. Chen, X. Wang, and Y. Guan, "Automatic text summarization based on lexical chains," in *International Conference on Natural Computation*. Springer, 2005, pp. 947–951.

[128] H. G. Silber and K. F. McCoy, "An efficient text summarizer using lexical chains," in *Proceedings of the first international conference on Natural language generation-Volume 14*. Association for Computational Linguistics, 2000, pp. 268–271.

[129] R. Barzilay and K. R. McKeown, "Sentence fusion for multidocument news summarization," *Computational Linguistics*, vol. 31, no. 3, pp. 297–328, 2005.

[130] S. M. Harabagiu and F. Lacatusu, "Generating single and multi-document summaries with gistexter," in *Document Understanding Conferences*, 2002, pp. 11–12.

[131] H. Tanaka, A. Kinoshita, T. Kobayakawa, T. Kumano, and N. Kato, "Syntax-driven sentence revision for broadcast news summarization," in *Proceedings of the 2009 Workshop on Language Generation and Summarisation*. Association for Computational Linguistics, 2009, pp. 39–47.

[132] P.-E. Genest and G. Lapalme, "Fully abstractive approach to guided summarization," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2012, pp. 354–358.

[133] H. T. Le and T. M. Le, "An approach to abstractive text summarization," in *2013 International Conference on Soft Computing and Pattern Recognition (SoCPaR)*. IEEE, 2013, pp. 371–376.

[134] K. Ganesan, C. Zhai, and J. Han, "Opinosis: A graph based approach to abstractive summarization of highly redundant opinions," in *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*. Beijing, China: Coling 2010 Organizing Committee, Aug. 2010, pp. 340–348. [Online]. Available: https://www.aclweb.org/anthology/C10-1039

[135] Kavita, C. Zhai, Ganesan, and J. Han, "Opinosis: A graph based approach to abstractive summarization of highly redundant opinions." Coling 2010, 2010.

[136] D. Wang and T. Li, "Weighted consensus multi-document summarization," *Information Processing & Management*, vol. 48, no. 3, pp. 513–523, 2012.

[137] Y. J. Kumar and N. Salim, "Automatic multi document summarization approaches," in *KS Gayathri, Received BE degree in CSE from Madras University in 2001 and ME degree from Anna University, Chennai. She is doing Ph. D. in the area of Reasoning in Smart*. Citeseer, 2012.

[138] A. T. B. Choi, "Knowledge based automatic summarization," 2017.

[139] L. Li, D. Wang, C. Shen, and T. Li, "Ontology-enriched multi-document summarization in disaster management," in *Proceedings of the 33rd international ACM SIGIR*

conference on Research and development in information retrieval, 2010, pp. 819–820.

[140] H. Saggion and G. Lapalme, "Generating indicative-informative summaries with sumum," *Computational linguistics*, vol. 28, no. 4, pp. 497–526, 2002.

[141] A. Gatt and E. Reiter, "Simplenlg: A realisation engine for practical applications," in *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009)*, 2009, pp. 90–93.

[142] P.-E. Genest and G. Lapalme, "Framework for abstractive summarization using text-to-text generation," in *Proceedings of the workshop on monolingual text-to-text generation*, 2011, pp. 64–73.

[143] A. Khan, N. Salim, and Y. J. Kumar, "A framework for multi-document abstractive summarization based on semantic role labelling," *Applied Soft Computing*, vol. 30, pp. 737–747, 2015.

[144] ——, "Genetic semantic graph approach for multi-document abstractive summarization," in *2015 Fifth International Conference on Digital Information Processing and Communications (ICDIPC)*. IEEE, 2015, pp. 173–181.

[145] E. Lloret, E. Boldrini, T. Vodolazova, P. Martínez-Barco, R. Muñoz, and M. Palomar, "A novel concept-level approach for ultra-concise opinion summarization," *Expert Systems with Applications*, vol. 42, no. 20, pp. 7148–7156, 2015.

[146] B. Endres-Niggemeyer and B. Endres, "Human-style www summarization," *Report. Hannover: University of Applied Sciences and Arts*, 2000.

[147] H. H. Mikhchi, "Standards of textuality: Rendering english and persian texts based on a textual model," *Journal of Universal Language*, vol. 12, no. 1, pp. 47–74, 2011.

[148] Beaugrand and Dressler, "Seven standards of textuality?" [Online]. Available: http://web.letras.up.pt/icrowcli/textual.html

[149] A. B. Tikarya, K. Mayur, and P. H. Patel, "Pre-processing phase of text summarization based on gujarati language," *Int. J. Innovative Res. Comput. Sci. Technol.(IJIRCST)*, vol. 2, no. 4, pp. 1–5, 2014.

[150] S. Singhal and A. Bhattacharya, "Abstractive text summarization," 2015.

[151] L. CY, "Rouge: a package for automatic evaluation of summaries," in *Proceedings of the Workshop on Text Summarization Branches Out. Barcelona, Spain*, 2004, pp. 56–60.

[152] D. R. Radev, H. Jing, M. Styś, and D. Tam, "Centroid-based summarization of multiple documents," *Information Processing & Management*, vol. 40, no. 6, pp. 919–938, 2004.

[153] G. Salton, "Automatic text processing. addison-wesley publishing company," 1988.

[154] H. Saggion, S. Teufel, D. Radev, and W. Lam, "Meta-evaluation of summaries in a cross-lingual environment using content-based metrics," in *Proceedings of the 19th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, 2002, pp. 1–7.

[155] C.-Y. Lin, "Looking for a few good metrics: Automatic summarization evaluation-how many samples are enough?" in *NTCIR*, 2004.

[156] GeeksForGeeks, "Longest common subsequence | dp-4." [Online]. Available: https://www.geeksforgeeks.org/longest-common-subsequence-dp-4/

[157] D. R. Radev, S. Teufel, H. Saggion, W. Lam, J. Blitzer, H. Qi, A. Celebi, D. Liu, and E. Drabek, "Evaluation challenges in large-scale document summarization," in *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*. Association for Computational Linguistics, 2003, pp. 375–382.

[158] A. Nenkova and R. J. Passonneau, "Evaluating content selection in summarization: The pyramid method," in *Proceedings of the human language technology conference of the north american chapter of the association for computational linguistics: Hlt-naacl 2004*, 2004, pp. 145–152.

[159] W. McKinney *et al.*, "pandas: a foundational python library for data analysis and statistics," *Python for High Performance and Scientific Computing*, vol. 14, no. 9, 2011.

[160] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[161] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

[162] F. M. Kundi, M. Z. Asghar, S. R. Zahra, S. Ahmad, and A. Khan, "A review of text summarization," *language*, vol. 6, no. 7, p. 8, 2014.

[163] V. Saravanan, "Text summarization from scratch using encoder-decoder network with attention in keras," 2020. [Online]. Available: https://towardsdatascience.com/text-summarization-from-scratch-using-encoder-decoder-network-with-attention-in-keras-5fa80d12710e

[164] P. Dwivedi, "Text summarization using deep learning," 2019. [Online]. Available: https://towardsdatascience.com/text-summarization-using-deep-learning-6e379ed2e89c

[165] M. Koupaee and W. Y. Wang, "Wikihow: A large scale text summarization dataset," *arXiv preprint arXiv:1810.09305*, 2018.

[166] M. Grusky, M. Naaman, and Y. Artzi, "Newsroom: A dataset of 1.3 million summaries with diverse extractive strategies," in *Proceedings of the 2018 Conference of the*

North American Chapter of the Association for Computational Linguistics: Human Language Technologies. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 708–719. [Online]. Available: http://aclweb.org/anthology/N18-1065

[167] L. Shao and J. Wang, "Dtatg: an automatic title generator based on dependency trees," *arXiv preprint arXiv:1710.00286*, 2017.

[168] A. Pai, "Comprehensive guide to text summarization using deep learning in python," 2019. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/

[169] J. J. McAuley and J. Leskovec, "From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews," in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 897–908.

[170] NIST, "Document understanding conferences." [Online]. Available: https://duc.nist.gov/

[171] T. A. S. Pardo and L. H. M. Rino, "Temário: Um corpus para sumarização automática de textos," *São Carlos: Universidade de São Carlos, Relatório Técnico*, 2003.

[172] M. Mayank, "String similarity — the basic know your algorithms guide!" [Online]. Available: https://itnext.io/string-similarity-the-basic-know-your-algorithms-guide-3de3d7346227

[173] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[174] M. Mendoza, S. Bonilla, C. Noguera, C. Cobos, and E. Leon, "Extractive single-document summarization based on genetic operators and guided local search," *Expert Systems with Applications*, vol. 41, p. 4158–4169, 07 2014.

[175] K. Hong, M. Marcus, and A. Nenkova, "System combination for multi-document summarization," in *Proceedings of the 2015 conference on empirical methods in natural language processing*, 2015, pp. 107–117.

[176] M. Gambhir and V. Gupta, "Recent automatic text summarization techniques: a survey," *Artificial Intelligence Review*, vol. 47, no. 1, pp. 1–66, 2017.

[177] D. Marcu, "Discourse-based summarization in duc-2001," in *Proceedings of the Document Understanding Conference (DUC01)*, 2001.

[178] A. Kornilova and V. Eidelman, "Billsum: A corpus for automatic summarization of us legislation," *arXiv preprint arXiv:1910.00523*, 2019.

[179] S. Narayan, S. B. Cohen, and M. Lapata, "Don't give me the details, just the summary! Topic-aware convolutional neural networks for extreme summarization," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, 2018.

Appendix A

Attachments

A.1 Datasets Used

1. **Amazon Fine Food Reviews** - Dataset with reviews of fine foods from Amazon. The data span a period of more than 10 years, including all 500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories [169];

2. **Newsroom: Dataset of 1.3 Million Summaries with Diverse Extractive Strategies** - *Newsroom* is a free summarization dataset of 1.3 million articles and summaries written by authors and editors in newsrooms of 38 major news publications. Extracted from search and social media metadata between 1998 and 2017, these high-quality summaries demonstrate high diversity of summarization styles. In particular, the summaries combine abstractive and extractive strategies, borrowing words and phrases from articles at varying rates. We analyze the extraction strategies used in NEWSROOM summaries against other datasets to quantify the diversity and difficulty of our new data, and train existing methods on the data to evaluate its utility and challenges. [166];

3. **WikiHowAll** - WikiHowAll is a new large-scale dataset using the online *WikiHow* knowledge base. Each article consists of multiple paragraphs and each paragraph starts with a sentence summarizing it. By merging the paragraphs to form the article and the paragraph outlines to form the summary, the resulting version of the dataset contains more than 200,000 long-sequence pairs. [165];

4. **Billsum: A Corpus for Automatic Summarization of US Legislation** - Dataset that contains information about US Congressional and California state bills [178];

5. **Xsum** - Dataset that contains information, such as text and summary about several document texts, from different areas [179].

6. **DUC** -The DUC dataset contains conference documents for text summarization, from 2001 to 2007. Each edition has data has text data about all sorts of subjects. One can have access to the full texts and three reference summaries, provided by three different human writers.

Glossary

Deep-learning	In artificial inteligence there is a field called machine learning, that combines several algorithms that improve over time through experience. Deep-learning is a field of machine-learning, where some of these algorithms contain several layers where they can process data and gain experience in their problems.
N-gram	A n-gram is defined as a sequence of *n elements*, that can be of any type: words, numbers, special characters. For example, a uni-gram and bi-gram, are sequences of one and two elements, respectively.
Stop-words	Stop-words are in most cases, words that are widely displayed in the text. Even though frequency in the text is high, they do not convey important information about it. Examples of stop-words are the words 'our', 'me', 'what', etc.
Corpus	In linguistics, a corpus consists in a set of documents that give respect to the same subject or discipline. For example, a set of documents that have data about economics can be represented as a corpus of economics.