POLITECNICO DI TORINO

MASTER's Degree in COMPUTER ENGINEERING

MASTER's Degree Thesis

MULTI-DOCUMENT SUMMARIZATION DRIVEN BY DOMAIN-SPECIFIC EMBEDDING

Supervisors

prof. LUCA CAGLIERO

prof. PAOLO GARZA

Candidate

MATTIA CARA

MARCH 2020

Abstract

Word embeddings are nowadays widely deployed in a large number of Natural Language Processing tasks. A word embedding is used to map each word belonging to a corpus, into a vector space, keeping semantic and syntactic properties. They are used in different implementations such as sentiment analysis, topic extraction, Part-Of-Speech tagging and of course document summarization. The focus of this thesis is towards this last job: the object is to extrapolate, given a collection of articles, the most relevant sentences to provide the reader only a limited set of information but hopefully the most meaningful.
Particularly, the scope of this work is to empirically show that a domain-specific word embedding is able to extract a better summary with respect to a general-purpose one. A general-purpose word embedding is obtained after a training phase in which the input corpus is made of texts of various nature. On the other side, a domain-specific word embedding is trained using only documents that are treating that particular domain. The idea behind is that, training a word embedding with documents belonging to the same domain will produce a better representation of all the words related to that argument, because, with respect to a non specific text, they are present more often and used in a more specific context. Other than that, a domain-specific embedding is capable to handle better words having multiple meanings: instead of treating each meaning with the same weight, the one linked to the precise domain will receive more relevance. This thesis is split mainly in two parts: the first one is about producing a domain-specific word embedding and judging its quality; the second one is about applying the previous result to a downstream task, the multi-document summarization indeed.

Acknowledgements

I would like to thank Prof. Luca Cagliero and Paolo Garza and the whole DataBase and Data Mining group for allowing me to have the possibility to work on this thesis, providing me all the materials and support required, my PhD advisor Moreno La Quatra, for his patience in teaching me all the knowledge required to solve all the problems encountered, and all the people that have worked previously on related topics, which have shared with everyone their discoveries and results.
I would also like to thank my parents, who have supported me emotionally and financially all these years, letting me choose my own path, and all my relatives and friends, who always gave me the chance to have a good laugh even in the most though periods.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the uprising relevance that internet has assumed these last few years, people has the possibility to access to a very large set of information by simply browsing the net using devices that are now familiar with everyone, such as smartphones or personal computers. The web may be considered as a huge library; indeed there is the chance to obtain documents related to every possible topic, but also produced by writers having different levels of preparation: the same subject may be treated using various approaches, targeting people having a different sex, age or also opinion.
Natural language processing (NLP) is the branch of computer science devolved to address and solve problems that are related to the human languages, implementing models and algorithms that are able to analyze a text. Some of these tasks may be solved easily by a human being, while instead machines are requested to implement complicated procedures: a classic example is the speech recognition problem, in which the machine has to understand what the speaker is saying.
Among all the tasks, the one chosen for the work of this thesis is the document or text summarization, which has as objective to provide a summary given a collection of texts. The goal is to extract as many meaningful information as possible, avoiding to repeat them. This attempt to eliminate overlapping concepts is also due the fact that each summary does not have an arbitrary number of words: a threshold is usually defined (the output cannot exceed a given number of terms) and this is also pushing this field of research to find always more accurate algorithms. Summarization is a concept that is very well known to everyone, for example kids at school are taught to underline with their highlighter all the periods they believe are containing the key concept of the topic. Nonetheless, this job will become extremely hard and extremely time consuming if the input data consists in thousands of articles and this is where computer science intervenes.

Document summarization, in reality, is just a generic word to indicate a large number of similar but at the same time different problems, because, as said previously, there is a large variety in documents and each one of those kind has to be handled in a proper manner. Throughout this work most of these aspects will be covered, even if the focus is put on the multi-document summarization driven by domain-specific embeddings, which becomes the most important actor while processing this task. A word embedding is a model used to map words into real number vectors which can be used while

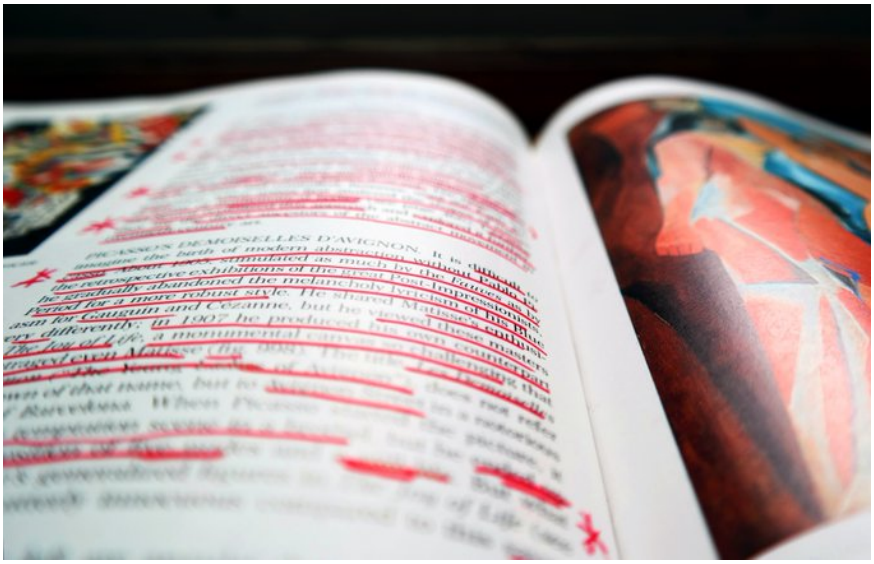

Figure 1.1: Skimming, is a technique widely adopted by students in order to discern which sentences are the most important; this helps further reading and memorizing faster concepts contained. Image taken from the August/September 2018 publication of the newspaper https://professionalartistmag.com/.

selecting which sentences are the most relevant. It is a very useful tool, because it does not only provide a way to compare periods but the distance retrieved represents semantic and syntactic relationships among words. Thanks to this aspect, it is possible to overcome one of the major issues of the document summarization, that is dealing with synonyms. Basic sentence selection algorithms, not based on word embeddings, are only taking into account matching words in order to understand if two sentence are similar and so if they are treating the same argument. On the other side, introducing a vector representation of a word or of a sentence will provide more degrees of similarity and more easiness to compute how much information are shared among two periods.
As just introduced, a word embedding is able to map a dictionary into a vector space, generally according to their meaning. This vocabulary is built gathering word coming from the input corpus used for training purposes: this means that, using different collections, we will obtain, first of all, different dictionaries but also different vector representations. Starting from this specific of word embeddings it is possible to formulate the objective of this thesis: is it true that a domain-specific word embedding is able to recognize with less difficulties terms belonging to a specific lexicon, with respect to a general-purpose word embedding? Moreover, due the fact that a word embedding is just a tool and it does not provide any meaningful results by itself, is the summary produced by a model exploiting a domain-specific embedding more refined with respect to a general-purpose one, while analyzing documents that are treating arguments related to that specific domain?
In order to answer these questions multiple aspect should be covered, to understand which actors are playing an important role in this task. The first step performed is about evaluating the goodness of the representations provided by a word embedding, focusing on those ones which are related to the domain chosen. It is important to have a metric that, given two word embeddings, is able to return a score which indicates

which one is the best in addressing those terms. Usually this test is done considering couples of synonyms or analogies; unfortunately they are not suited for our purpose and so we have chosen to introduce a new evaluation model, based on a glossary containing definitions related to a specific domain. While dealing with the actual document summarization task, in order to have a compete view of the scenario, five word embeddings have been tested (four domain-specific and one general-purpose) using a handful set of sentence selection algorithms on three different test datasets.
Results gathered are showing that domain-specific word embeddings are able to, while dealing with homonyms (words written in the same way but having multiple meanings), assert more relevance to the meaning most linked to the domain. However, the glossary evaluation attempt did not return positive results; there is, before discarding it, still the possibility to refine it and improve it. Talking about, instead, the quality of the summaries produced, the difference of scores while using algorithms using word embeddings or less is very small. Finally a very good result has been obtained in comparing domain-specific with general-purpose embeddings: in one of the dataset implemented, summaries produced by domain-specific models are significantly better. While considering these results, we must remind that the ROUGE metric have been used, which rewards matching words and not synonyms: a summary with a lower score could also signify that, to express the same concept, it has used different terms.
This thesis is presenting other seven chapters after this introduction. In Chapter 2 there is a brief explanation of the preprocessing techniques, used to elaborate and refine the input text that, and of which datasets will be used. Chapter 3 is instead devolved to introduce various models of word embeddings, presenting their evolution to solve issues affecting previous implementations. Tests to validate them are introduced in the chapter 4, which portrays intrinsic evaluation tools, including the summary approach proposed. In the next two chapters, after a brief excursus of the actual state of extrinsic evaluation models, the focus is put on document summarization, describing many ways to implement it (chapter 6 will treat only word-embedding based sentence selection algorithms). Lastly, chapter 7 and 8 contain all the results gathered throughout this work and the relative conclusions.

Workflow

In figure 1.2 there is the flowchart followed to reach the final result of this work.

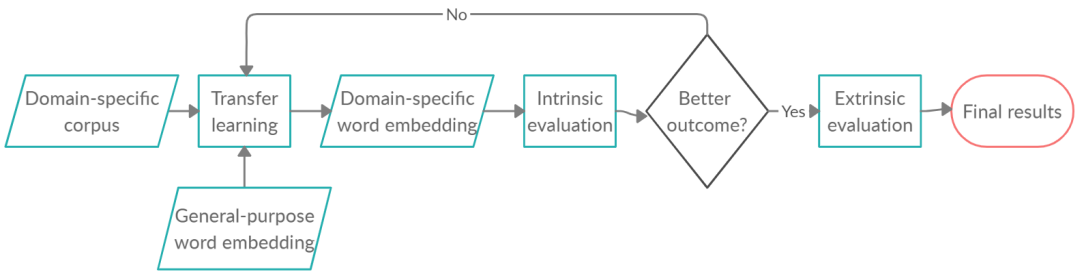

Figure 1.2: Flowchart of the work made throughout this thesis.

This path may be implemented to deal with NLP problems regarding domain-specific

scenarios, even if the downstream task chosen is not the document summarization. The starting point is represented by a collection of texts, all imprinted around the same topic (domain-specific corpus in the figure), and by a word embedding that is suited to address general-purpose tasks. Performing a new training phase on the above mentioned model it is possible to obtain a new representation, which goodness will be tested to understand if the result desidered has been achieved (intrinsic evaluation). It is possible to loop this first phase whenever the outcome is not satisfactory enough. Once this has been completed, the first result will be implemented in a downstream job (extrinsic evaluation) in order to understand which benefits it will carry with itself.

Chapter 2

General Background

In this chapter the most important data preprocessing techniques are covered. NLP implementations are usually solved using a cascade of intermediate tasks: their purpose is to remove all the words that are not useful for the analysis and provide them to the model in a standard form, so that it is easier for the machine to recognize them. It will follow a brief introduction of the datasets used in this work.

2.1 Data preprocessing

Before explaining what a word embedding is, how it is obtained and how it can be contextualized with regard of a specific argument, in this chapter different techniques will be covered. They are implemented in a pipeline fashion and they are fundamental in order to obtain a result that has any meaning. Indeed each text that is used has to be first of all preprocessed in order to be ready to be handled. The following tasks will be described keeping in consideration the fact that all the articles involved in this thesis are written in English. Using a different language means that you have to deal with very different grammatical aspects: different punctuation marks, the presence or not of blank spaces to understand where a word ends and another begins or other differences may vary by a lot the complexity of the following tasks. All the functions described are taken from the NLTK python library(https://www.nltk.org/) that is dedicated to solve natural language processing problems.

2.1.1 Sentence splitting

The first step to be performed is splitting the body of the text into sentences. Indeed each word has a correlation only with other words within the same sentence; we can consider each sentence as a different environment in which all terms co-operate to produce a meaning and so they are the basic units composing the text. The summary obtained is in fact made of the most meaningful sentences, until reaching a given number of words, produced by the algorithm chosen. In English the full stop is the punctuation mark we are looking for to achieve this goal. Unfortunately it is used also for other purposes such as an indication of omitted characters, an acronym or abbreviations. Because of this, the function *sent_tokenize()* and all the ones similar to it are based on top of a model trained to recognize all the previous scenarios.

```
import nltk

sentence = "B. Obama is a member of the Democratic Party.
He is the 44th president of the U.S.A."
sent_token = nltk.sent_tokenize(sentence)
print(sent_token)
```

```
['B. Obama is a member of the Democratic Party.',
'He is the 44th president of the U.S.A.']
```

2.1.2 Tokenization

The following step is the so-called *tokenization* that will produce as a result a list of terms. Once each sentence has been separated by the others it becomes important to analyze which words are found inside them and how many times they appear. This step is also fundamental when using tools, like the document-term matrix, required during the computation phase performed by NLP algorithms. In English is enough to split the input sentence each time a blank space is found and that will produce a list of words. This job is performed by the method *word_tokenize()* present in the NLTK library.

```
word_token = []

for sent in sent_token:
    word_token.append(nltk.word_tokenize(sent))

print(word_token)
```

```
[['B.', 'Obama', 'is', 'a', 'member', 'of', 'the', 'Democratic', 'Party', '.'],
['He', 'is', 'the', '44th', 'president', 'of', 'the', 'U.S.A', '.']]
```

2.1.3 Stop words

After reaching this point it is required to remove stop words. This particular type of vocables do not provide any specific meaning to the context of the sentence and because of that they need to be filtered before the start of the computation. They usually represent the majority of words in a sentence; among the most used we can find words like *the*, *or*, *at*, *which* and many other. There is not an universal list of stop words and each library will use a slightly different version of it but it is also possible to add or remove some of them depending on your needs. NLTK offers the possibility to load pre-made stop words lists, according to the language required.

```
stop_words = set(stopwords.words('english'))
        word_clean_token = []

        for sent in word_token:
            clean_sent = []
            for w in sent:
                if w not in stop_words:
```

```
                    clean_sent.append(w)
            word_clean_token.append(clean_sent)

        print(word_clean_token)
```

```
[['B.', 'Obama', 'member', 'Democratic', 'Party', '.']
['He', '44th', 'president', 'U.S.A', '.']]
```

2.1.4 Stemming

Stemming is the last step to perform in order to complete the pre-processing phase. This procedure is used to reduce the word to its basic form, the *root* form. This process is fundamental for NLP problems because it makes converge many terms to the same one: to understand the context of a sentence, for example, is not important to know the tense of a verb or the fact that a noun is singular or plural because all the possible forms that can be hold by a term are useless with regards of their actual meaning. The stem does not need to be a word; if a word is instead required as an output then the process is called *lemmatization* and it will substitute each word with their lemma. Even if this looks like an easy step to implement, there are actually many algorithms that can be chosen, depending on your needs. Indeed a trade-off between the accuracy and the performance has to be evaluated when selecting one of them. The library NLTK provides the *Porter* version that is de facto the standard algorithm for the Engish language.

```
stemmer = PorterStemmer()

        for sent in word_clean_token:
            stemm_toke = [stemmer.stem(token) for token in sent]
            print(stemm_toke)
```

```
['B.', 'obama', 'member', 'democrat', 'parti', '.']
['He', '44th', 'presid', 'u.s.a', '.']
```

2.2 Datasets

In this section a briefly explanation of the datasets used is provided. These datasets have been used mainly for two purposes: to train the model that will be later on used to produce summaries and to test their performance.

2.2.1 Reuters

Reuters https://www.reuters.com/ is a news agency founded in October 1851. Various collections of articles written by their journalist are available and they will be used in this work to train and to contextualize embeddings. The choice to utilize their documents is not casual, indeed, due to the fact that they have been produced by experts, each text will contain a lexicon that is suitable to be printed on a newspaper. The aim

is to obtain a model that is more capable to recognize this kind of terms and periods. Two different collections have been chosen:

- *Reuters27000*, 27000 english news articles download from the main web page, taking in account the argument they belong to. These categories are: health, art, politics, sports, science, technology, economy and business;
- *RCV1* (Reuters Corpus Volume 1), an archive containing more than 800000 English news articles collected from 1996-08-20 to 1997-08-19, available for research purposes.

2.2.2 DUC-2002

The *National Institute of Standards and Technology*, NIST in shorthand, is an organization which has its interests in promoting and in supporting technology innovations. They set up many competition in which team of researchers may compete. It is organized in laboratory programs dealing with different topics; among these we can find contest regarding the area of text summarization, called the **Document Understanding Conference** (DUC). They propose challenges both in English and from foreigner languages to English (cross-language summarization). DUC-2002, particularly, is made of 567 news articles coming from well-known American newspapers. For each document a human-written reference summary has been provided. The final result has a constraint of 100 words to be respected.

2.2.3 Opinosis

Opinosis is a dataset containing Amazon product reviews instead of new articles. For this reason the result coming from it are not so relevant with respect to this work, because the goal is to demonstrate that a word embedding trained using newspaper article works better than a general-purpose word embedding. Nonetheless it is interesting to compare the results coming from it in order to understand the goodness of the model presented later on. Opinosis is made of 51 groups of reviews: each cluster contains documents related to the same product. It is a rather small dataset so it is very likely that it doesn't hold any statistical relevance.

2.2.4 BBC

The British Broadcasting Corporation (BBC) is a society created to broadcast news all around the world, using multiple channel of communications, such as radio, internet or television. This dataset (`http://mlg.ucd.ie/datasets/bbc.html`) is made of 2225 articles, presenting a label according to their area (business, entertainment, politics, sport, technologies), gathered from 2004 to 2005. BBC have created this datasets for benchmark purposes, to be used during machine learning experiments, such as [9].

Chapter 3

Word Embeddings

Word embeddings is the name to identify a collection of models used in the NLP environment to map words or sentences into a vector space. There are many ways to obtain them, spacing from neural networks to probabilistic implementations, but the final goal is always the same: mapping each term of the vocabulary into an array of real numbers, in such a way there is the possibility to allow machines to analyze and comprehends documents.
To produce a word embedding, a corpus, that is a collection of texts, is required; each word presents an enough number of times will be mapped into the vector space. This mapping depends on how the word is used: while analyzing each document, the frequency of the word is taken into account, together with which terms are usually rather close to it. The best quality of this model is that arrays of numbers are holding semantic and syntactic properties among words, evaluated as distances between their vectors. This characteristic allows to a handful set of NLP tasks to understand better documents they are analyzing. Indeed, in this vector representation, synonyms should be located in the same area and relationships are underlined by a distance metric. For example, this is very useful in document summarization, because the algorithm, chosen to extract which sentences are the most relevant, is able to understand that two sentences are introducing the same concept even if they contain very different terms. Due the fact the final result is depending on the input corpus, it is possible to obtain word embeddings which are most focused on different lexicons. Using a domain-specific corpus should provide a better representation of every words strictly linked to that argument, with respect to a corpus that is instead more general.
One of the most hard issue to solve is to correctly address terms that are holding various meanings (homonyms), because this implementation will collapse all of them into the same vector (each sequence of characters is mapped into the same representation, the one-to-one correspondence is between points in the vector space and words and not their meaning). In this chapter the most well-known methods to produce them are explored, focusing on how to solve this recently introduced problem, emphasizing the meaning that is more related to a specific topic.

3.1 Word2Vec

Word2Vec (Mikolov et al., 2013a [22]) has been developed by Google and it is a technique that implements a neural network that is able to map words into a vector space, such that each word will be translated into a real number vector. Within this model synonymous words are expected to be near one to each other: this property is really useful for many NLP tasks because algorithms implemented are relying on this metric to evaluate their results. The best characteristic of this representation is given by the fact that many types of similarities between words may be obtained through linear translations (Mikolov et al., 2013c [23]). This is an incredible advantage because it allows to literally process numbers, procedure that machines are excellent in, in order to manipulate texts and words that do not hold any particular meaning to machines. An example (fig 3.1), to clarify what just said, is the following: given three vectors, corresponding to the words *king*, *man* and *woman* it is possible to obtain a vector close to one that represents *queen* just doing this operation:

$$queen = king - man + woman \tag{3.1}$$

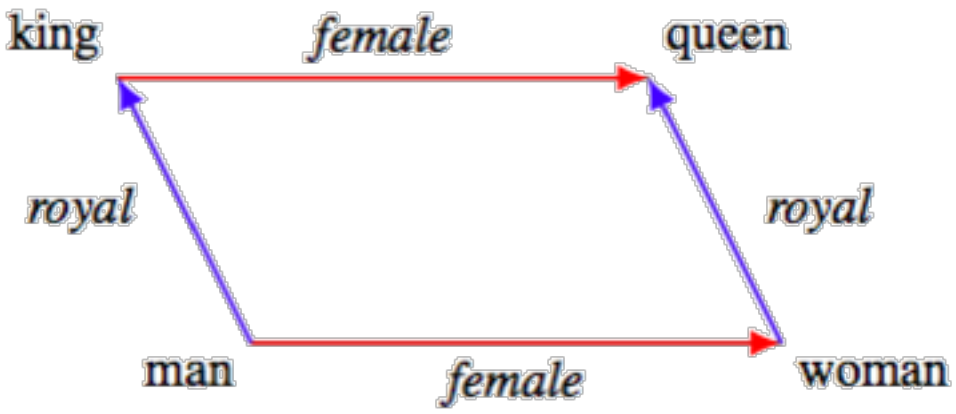

Figure 3.1: This is a 2-D (the actual model has 300 dimensions instead) graphic representation of how vectors associated to words can be manipulated. Both the couples man-king and woman-queen are linked by the concept of *royalty*; couples king-queen and man-woman by the concept of *female*. Image taken from (Kawin Ethayarajh), Why does King - Man + Woman = Queen? Understanding Word Analogies https://kawine.github.io/blog/nlp/2019/06/21/word-analogies.html.

Two different models exists for learning word representations *continuous bag-of-words* (CBOW in short) and *skip-gram*.
CBOW architecture bases its algorithm on the neighborhood of each word. The starting point of this procedure is to define a window, in the figure 3.2 starting from w(t-2) until *w(t+2)*, so that each word contained in this interval will be taken into consideration while computing the result. To guess the term located in the middle *w(t)*, the model will combine all the other representations; this evaluation is based on the fact that it is possible to understand the missing word just looking at the context in which it is found. The order in which terms are appearing is not taken into account: the goal is straightforwardly to make a prediction based on the presence of words regardless

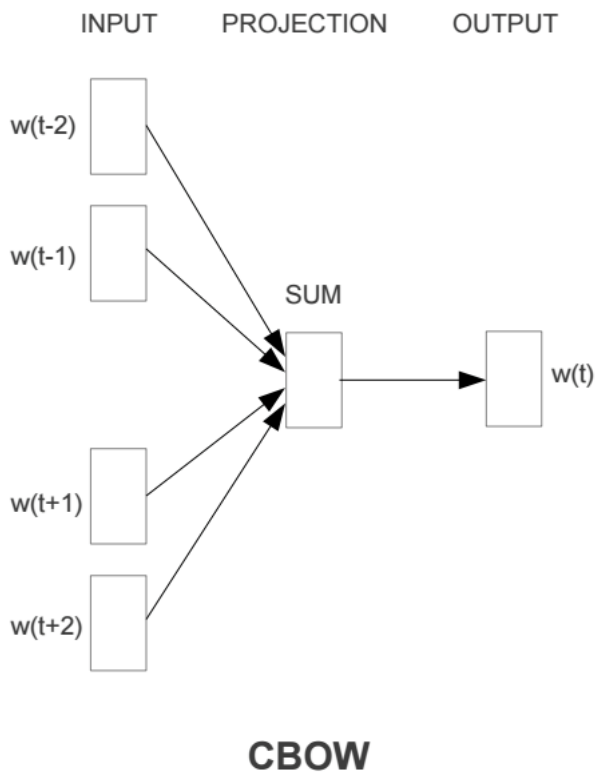

Figure 3.2: CBOW graphical representation; the surrounding words are combined in order to predict the word in the middle. Image adopted from the original paper [22].

their location.
The Skip-gram is another variant of the Word2Vec model that works in an opposite fashion with respect to the previous model.

As we can see from the figure 3.3 in this case the input is the word in the middle and the model has the job to correctly address which words belong to its context.
Both these approaches may be performed starting from very large corpora because they are characterized by a low computational complexity. Training the same corpus with the two models will produce two results that are pretty similar; nonetheless there are some minor drawback or advantages: Skip-gram usually returns a result that is slightly more accurate; on the other hand CBOW is able to process more data in a shorter time and so it is suited to deal with larger dataset.

3.2 GloVe

GloVe, or Global Vectors in full, is an example of word embeddings introduced by Pennington et al. 2014 [28]. The name, Global Vectors, has derived by the fact that with their model they are able to understand and reproduce global information coming from the input collection of documents. In order to improve the previous model, in their work they are giving a great relevance to the co-occurrence probabilities of words, together used in the same context. This means that to obtain the final vector

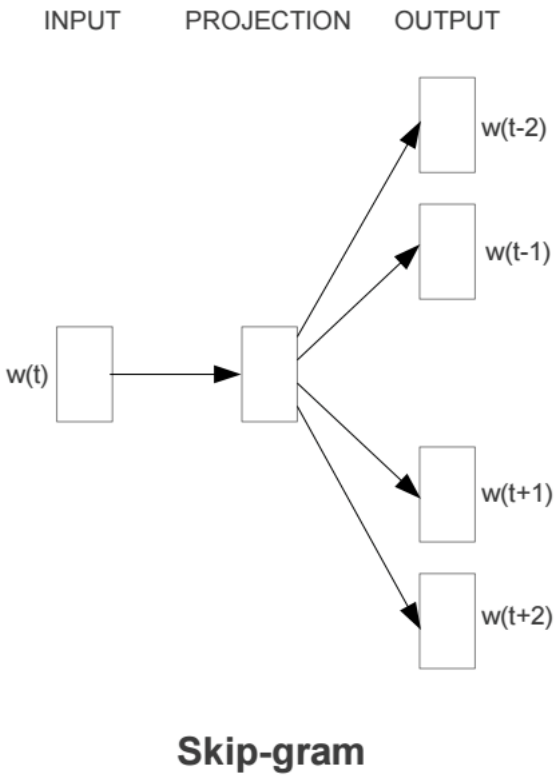

Figure 3.3: Skip-gram graphical representation; the distributed representation of the input words is used to predict the context. Image adopted from the original paper [22].

representation they are considering which words are often appearing in the same sentence, but in a smarter way with respect to previous attempts. Indeed, this time, not all the words in the context are considered equally, but just the ones which are strongly characterizing. This strategy is used to give less relevance to words which are used more times and so they are present in a large number of contexts. To understand if two terms are correlated there is the need to introduce some *probe* words that will be used to make a comparison. The probability of a word j to be present in the context of the term i is given by the formula

$$P_{\mathrm{ij}} = P(j|i) = \frac{X_{\mathrm{ij}}}{X_i} \tag{3.2}$$

where X_{ij} is the number of times the word j can be found in the context of i and X_i is the count of words in its own context. An example, provided by Pennington et al. 2014 [28], is about two words, *ice* and *steam*. The goal is to search which words are the most able to discern these two targets terms. To obtain them it is requested to evaluate the probability, introduced in 3.2, with probe k terms, such as *solid*, *gas*, *water* and *fashion*. We are interested in those words which will return as result a proportion P_{ik}/P_{jk} of co-occurrence statistic that is larger than one, avoiding words which instead are given as result a score smaller than one. Whenever this type of score is achieved it signifies that the probe words chosen is correlated to just one of the two words and not both;

on the other hand, when the probe words are *water* and *fashion*, the ratio is very close to the unit because, in the first case, the word is correlated to both, in the second case to none of them. This concept just introduced is at the base of the functioning of their model: the word vectors used in the GloVe model are therefore acquired from this co-occurrence score index rather than using the probability of having a certain word in a document.

3.3 Sentence embeddings

It is possible to change the focus of the embedding from words to sentences. This need to introduce a different implementation is provoked by the fact that almost never the same word will be used to express a concept and there are many terms which are considered ambiguous, they can hold a different meaning depending on the context and because of this they need to be treated differently. It is indeed very hard to understand the content of a period just by analyzing separately word by word, leading most of the time to a rather imprecise result. The straightest approach to solve this problem is to perform operations (such as addition or multiplication) among terms (Mithcell and Lapata, 2010 [24]) in order to obtain a representation of the sentence itself. Unfortunately this type of approach does not keep in consideration all the multiples relationships between words that are present in every language: usually when a person is producing a sentence, he does not just juxtapose words one next to each other but they are put following a well-known order in such a way the final result is more complex and more expressive (the easiest example can be found in common sayings, in which the meaning of the sentence is far away from the literal meaning). This is why *sentence embeddings* have been explored, so that is possible to process a document on a sentence-level. Different models have been proposed; among these:

- *Paragraph Vector* [17] is a method that tries to overcome the drawbacks introduced by the bag-of-words models, in which words are considered out of order and the lack of information about words semantics. Paragraph vector is an unsupervised model based on the forecasting of terms included into fixed-length part of the text (for example a paragraph or a sentence);
- *auto-encoder* is a type of neural network (figure 3.4) that is usually used to learn a representation (an encoding indeed) about your dataset. Its working its divided in two main phases: an encoder to transform the input and then a decoder that is able to reproduce an output, which should be as close as possible to the input provided. It is possible to see then, how it can be used while dealing with NLP problems. The goal of this architecture is to be able to understand how to encode and summarize all the semantic and syntactic properties; the final encoding will be used in future analysis to recognize with a higher precision patterns of words. Slightly different models have been proposed, each one implementing a diverse kind of neural network, such as a recurrent neural network [10] that is able to keep a state evolving during time (very useful considering that each part of speech is not randomly inserted into a sentence, but instead they are placed in a fixed schema), an unfolding recursive auto-encoder [30] or the Skip Thought model [14];
- *Smooth inverse frequency* proposed by Arora et al. (2017) [1] is the one that will

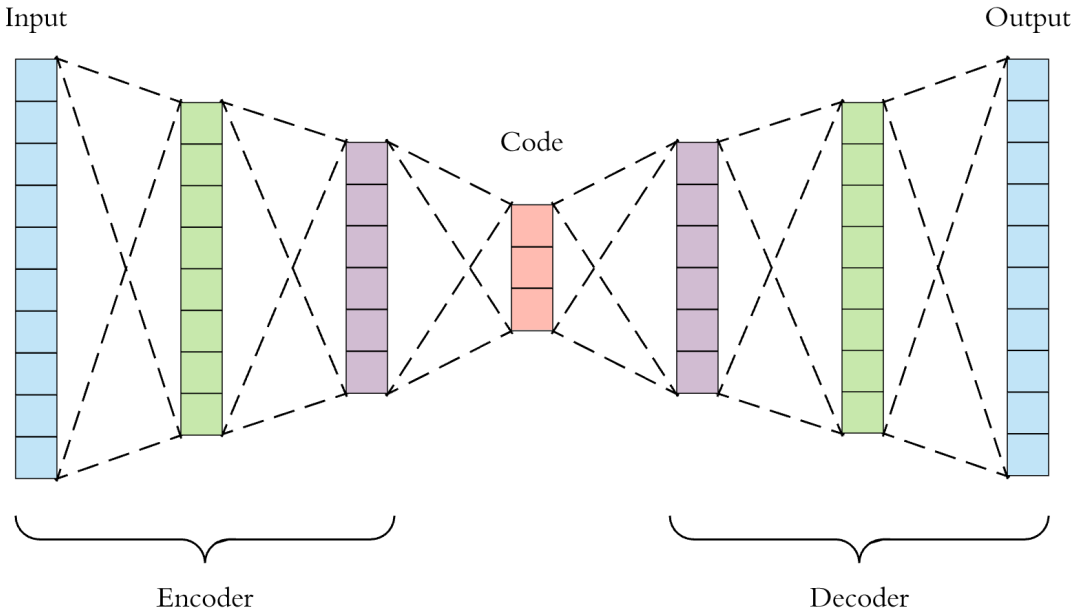

Figure 3.4: Basic schema of an auto-encoder structure. The input is processed in order to create a code that is going to be used to produce the output, the latter will be confronted with the source in order to understand if the encoding and decoding procedures are working correctly. Image taken from Arden Dertat, Applied Deep Learning - Part 3: Autoencoders https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798.

be used in this work and because of this it will be explained with a little more details. The algorithm that has been used to produce this new embedding is the following:

Algorithm 1 Sentence Embedding algorithm from Arora et al. (2017) [1].

Input: Word Word embeddings $v_w : w \in V$, a set of sentences S, parameter a and estimated probabilities $p(w) : w \in V$ of the words.
Output: Sentence embeddings $v_s : s \in S$

1: **for all** sentence s in S **do**
2: $v_s \leftarrow \frac{1}{|s|} \sum_{w \in S} \frac{a}{a+p(w)} v_w$
3: **end for**
4: Form a matrix X whose columns are $v_s : s \in S$, and let u be its first singular vector
5: **for all** sentence s in S **do**
6: $v_s \leftarrow v_s u u^T v_s$
7: **end for**

This approach starts computing the vector average of every terms contained in each period, introducing a weight factor that indicates the estimated probability to have that precise word in that context. Once this has been computed, the first principal component is obtained and the projection of each sentence onto it is removed. This is done to remove syntactic information in the sentence embedding in order to amplify the semantic one instead.

3.4 Text similarities

As explained before, word embeddings are used to retrieve a numeric representation of word belonging to a dictionary. These numbers have been used to perform many operations, but one of the most relevant statistic that could be evaluated is the distance or similarity. The similarity among two periods is used very often to understand if they are dealing the same argument. It is enough to think about search engines and the work that they are implementing. Search engines are requested to, given an input made of a group of words, find as many web pages containing a sentence that is, in the best case scenario, equals to the target one or otherwise the closest one. This task is not strictly linked to the world of retrieval information but its importance may be found also in legals situations. Indeed, the *Case Law principle* guarantees that identical scenarios should be treated in a similar way. A judge should be able to read all the papers related to two cases and to understand if their content is similar enough to reserve them the same treatment.
Identifying two similar periods for a human is a rather easy action to do; it becomes a little more complicated if it is done by a computer. The most basic approach is to check the presence of the same words in both sentences, but this is sometimes misleading. Just think about the following sentence:
A dog is eating a piece of meat.
It is possible to change completely the meaning of it without introducing any new words, but just changing their order. This is because, for example, it is possible to switch the subject of the sentence with the object complement. In this case the period would become:
A piece of meat is eating a dog.
Another important aspect is about recognizing that two sentences are explaining the same concept even if the words contained inside them are different (for example, in the previous period the meaning stays the same if instead of the word *eat* there would have been the term *consume*).
This is why is very fundamental to pick the correct implementation of distance. Now it will follow a list of similarities used while dealing with NLP problems and a short description about their goodness.

- *Jaccard similarity*; the Jaccard similarity is defined as the number of matching words divided by the total number of words. As told before this is a really bad solution because it does not take into account synonyms or the functionality of each word in the sentence;
- *Cosine similarity*; thanks to the mapping obtained through word embeddings it is possible to evaluate the cosine between two words

$$cos(\theta) = \frac{\sum_i A_i B_i}{\sqrt{\sum_i A_i}\sqrt{\sum_i B_i}} \tag{3.3}$$

  It represents an improvement with respect to the Euclidian metric because, if the embedding has been correctly trained, similar concepts are mapped into arrays with a rather similar angle;

- *Word Mover's distance*; to apply this metric it is first required to have evaluated a word embedding. Word Mover's distance (fig. 3.5) considers the shortest distance that a sentence has to "move" in order to reach the other one. This movement should be thought as the vector required in the embedding space to reach a word *B* starting from a word *A*.

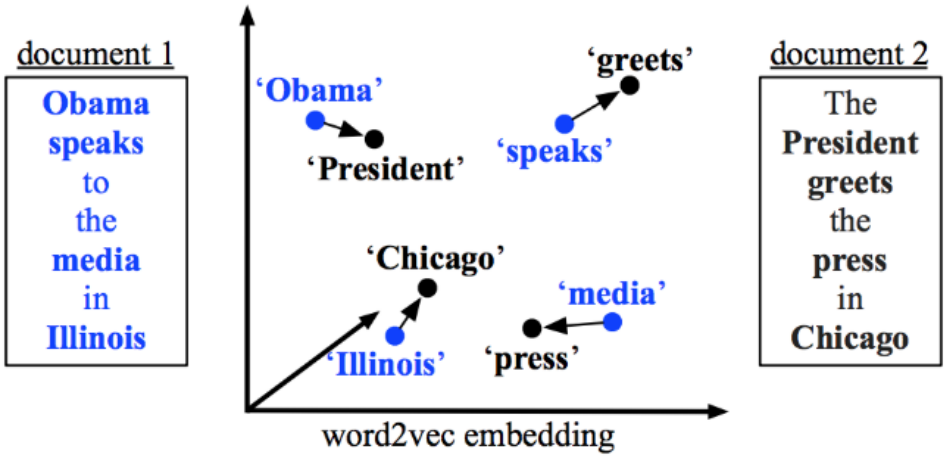

Figure 3.5: In the image the distance to reach the words belonging to the second sentence are represented by the black arrows. Couples considered are not randomly generated: each term is paired with the word of the other period that is the closest. Image adopted from the original paper [16].

  This way to evaluate a distance could be seen as a *transportation problem*, a problem that is solved by the path to be followed in order to spend the lowest quantity of resources (for example, choosing the correct sequence of routes to distribute packages by a courier service);

- *Latent Dirichlet Allocation* (LDA); a generative approach that is able to identify arguments inside texts and how much they have been treated, in order to assign a different degree of relevance. Indeed most of the time a single text is covering multiple arguments, from the start to the end, and because of this there is the chance to compare two articles depending on which topic have been touched and the relevance they assume inside the document. LDA, besides learning which topics are covered, is also linking to each topic a group of terms that are characteristic of it. Once these sets have been evaluated, LDA implements the *Jensen-Shannon* divergence to assign a score of analogy between two bodies of text.

3.5 Domain adaptation techniques

As previously introduced, word embeddings are providing a mapping of words to a high dimensional vector space, maintaining semantic and syntactical information. While building this model a vocabulary is built, gathering all the words contained in the corpora used for the training phase. Because of this, only the terms contained in these documents are actually mapped (only if there is a big enough number of them) and the final representation obtained is strongly dependent on the corpora itself. Let

us consider an embedding obtained after a training phase using a dataset coming from Twitter. This model will be very precise in addressing all the words that are used commonly by the people that are using the platform. As soon as we decide to apply this model to a downstream task, for example, like in this work, about document summarization of newspapers articles, we could encounter a few problems, such as the absence of words belonging to a specific lexicon or an imprecise mapping of words that are rarely used. Another issue that can be encountered is that really often, specific datasets are too small: to correctly train a word embedding a big quantity of input text is required, otherwise the results will be affected by a large noise, affecting the final result.
To solve this lack of information, there is the need to improve word embeddings model, contextualizing their dictionary and representation to deal with a more specific domain. This technique is called *Domain adaptation* and it is about, using as starting point a generic word embedding, performing an ulterior training phase to add more information.
This procedure, known as *transfer learning*, has to be implemented carefully because it is possible to obtain a better representation for the specific lexicon, but at the same time we could ruin all the previous results produced by the first phase of training, with the generic corpus. We have to ensure, while retraining word embeddings, to provide a dataset that contains most of the terms inside a normal dictionary. Word embeddings models are implemented to map semantically synonyms words into similar region of the vector space. Whenever we are retraining word vectors, we are shifting every word and it is possible that the model "forgets" that two words are synonyms if they are not present a sufficient number of times in the new corpus. This aspect introduces a problem about the number of samples to be contained inside a dataset: trying to contextualize a word embedding using a small dataset could actually decrease the performance instead of increasing them.

Canonical Correlation Analysis

Canonical Correlation Analysis (CCA) [13] is a method to infer information exploiting cross-covariance matrices. In this scenario, the aim is to add any relevant information from the vocabulary obtained starting from a domain specific collection into a generic one. Given two arrays X and Y of random variables, CCA algorithm is able to find a linear combination of them, having the highest correlation, if the two starting arrays are correlated. This procedure is performed on a single dimension, but it can be expanded to a case having a number $d > 1$ dimensions. It is possible to introduce a kernel function to improve the previous approach. Indeed, CCA works in a linear fashion and it may happen that it will not provide the best result possible in the starting vector space. This is where kernel function are enters in action: a kernel function (fig. 3.6 is able to map all the data into a feature space, enabling the possibility to reach a better representation. This solution is instead called KCCA;

Retrofitting with semantic lexicons

Retrofitting with semantic lexicons is the attempt performed by Faruqui et al. (2014) [8] to refine and improve word vectors, trying to provide closer representations to couple of words that are strictly linked among them. The procedure is based on the concept

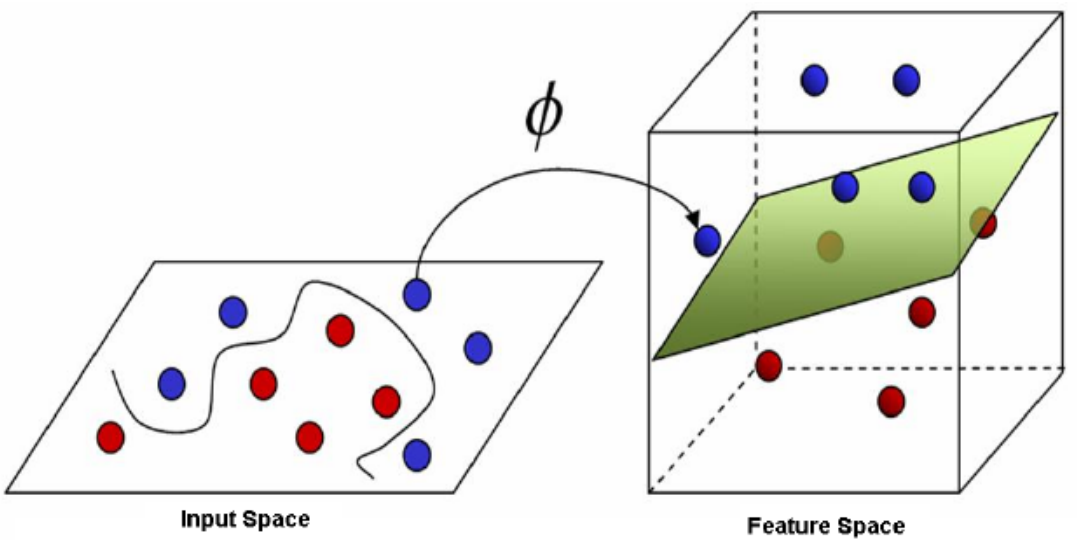

Figure 3.6: Simple example of why kernel functions are useful. The issue of finding a function that separates the two classes (red and blue) can be easily solved in the feature space, where this problem is linear. Image taken from [25].

of an *ontology*, which contains relationships among words contained in the dictionary, built as a collection of every words contained in the word embedding. This kind of relationships are dependent on the lexicon chosen to retrofit: using different lexicons will imply that different environments will be considered and so the meaning of each word will be slightly shifted. These connections among words are represented by an edge in a graph (figure 3.7), in which each vertex is a term of the vocabulary. Given a matrix Q containing all the word representations vectors as columns, the goal of this technique is to find a matrix R, such that each column will contain a vector that is, not only similar to the same word vector in the matrix Q, but also to each other term sharing an edge with it in the graph representation.

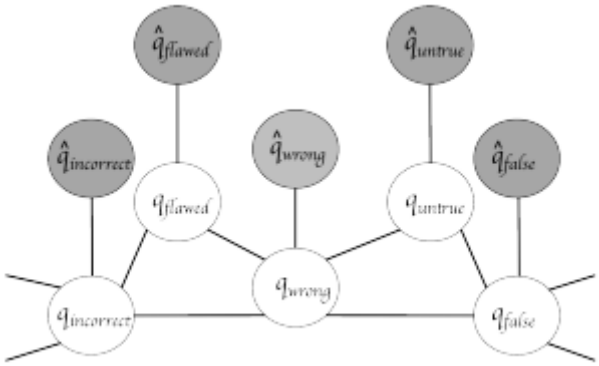

Figure 3.7: Vertex colored in grey are the original word representation while white ones are the inferred ones. Image adopted from the original paper [8].

The distance used in this implementation is the Euclidean distance and the objective

is to find the minimum of the formula

$$\Psi(R) = \sum_{i=1}^{n} \left[\alpha_i \left\| q_i - r_i \right\|^2 + \sum_{(i,j)\in E} \beta_{ij} \left\| q_i - q_j \right\|^2 \right] \tag{3.4}$$

where α and β are parameters indicating how significant is the relationship among two words. ψ is a convex equation and so its solution may be found implementing a system of linear equations; its first derivative will be used to settle all the vectors to a new value and this procedure may be repeated multiple times to obtain an outcome with an higher degree of precision at each iteration.

Subword information

Subword information [3] is an approach which has a background idea different from the previous ones; indeed here the key concept is to infer new information into the word embedding using the morphology of each word. Due to this feature, this implementation may vary its performance depending on which vocabulary it is applied: each language presents its own number of words (the more terms there are, the more there is the chance to find similar words, sharing chains of characters) and there are different rules to obtain their variations, such as singular/plural form or verbs conjugations. This model is exploiting the *skipgram* model that has been previously explained (figure 3.3) but this time it will be applied to each single term obtaining as results groups of characters, instead of having group of words coming from a sentence. To provide an example we can analyze the word *where*, using $n = 3$; the *3-grams* are going to be:

<wh, whe, her, ere, re>.

Particular attention has to be put on the sequence *her* because it will be treated differently from the actual English word her, furthermore groups composed by a single character are discarded because they don't hold any relevant meaning. From this previous example it is possible to understand that there is the chance to divide and capture all the prefixes and suffixes, morphemes that are placed before or after the stem of the word. Given a word, its representation will be obtained this time by gathering all the *n-grams* and adding their representations. Implementing this model, terms containing sub-sequences of the same characters will share the same partial representation, leading to have a final vector more similar with respects to words that are completely different.

Contextualized word vectors

Contextualized word vectors (CoVe in shorthand) is a method to improve the goodness of a word embedding, which has its root in computer vision techniques. In image recognition problems, an improvement has been registered while using weights that have been trained on previous sets (i.e. ImageNet). This behaviour is justified by the fact that models which share the same common goal may benefit from partial results coming from *synergistic tasks*. Whenever different architectures are made of analogous elements there is the possibility to inherit some information to speed-up or boost the training phase. Even if image recognition and NLP tasks are very different one from

each other, both of them are implementing a neural networks: this is the reason why this approach has been explored and implemented.
Word embeddings have proven multiple times to be a very solid starting point to deal with a large set of NLP tasks; nonetheless there is a major concern about their implementation. Indeed the output of a word embedding is a vector representation of each word without any context: this represents an issue because, each time a document is analyzed, words are not appearing by themselves and the general meaning is created by a chain of terms placed in the correct order. The key point of this model is to learn a representation of words within a specific context, such that it will be used in other NLP related tasks. In this particular scenario, instead of learning weights like in the computer vision field, the main goal is to learn an encoder starting from a large NLP problem, which will be used later on other implementations.
In the original paper McCann et al. (2017) [20] the task chosen to test this new method is the Machine translation: given a portion of text to the machine, the algorithm implemented needs to return as an output the best possible translation possible. It is rather easy to see how context is fundamental in this type of operations, because each term will be replaced in the new language by its correspondent but this is not always a 1-on-1 correspondence. For example the word "water" may be translated to Italian with two words, "acqua" or "annaffiare", depending on if it is used as a noun or as a verb. This information, like many other, is possible to be understood only if the target word is not isolated but it is considered inside a context.

Sense embeddings

Sense embeddings is an attempt of transfer learning made by Comacho-Collados et al. (2018) [4] to produce a representation that is based on the meaning of each word, rather than a representation of each "sequence" of characters. As explained before, one of the major restriction of word embeddings is that if a term holds multiple meanings, they will all collapse into the same vector.

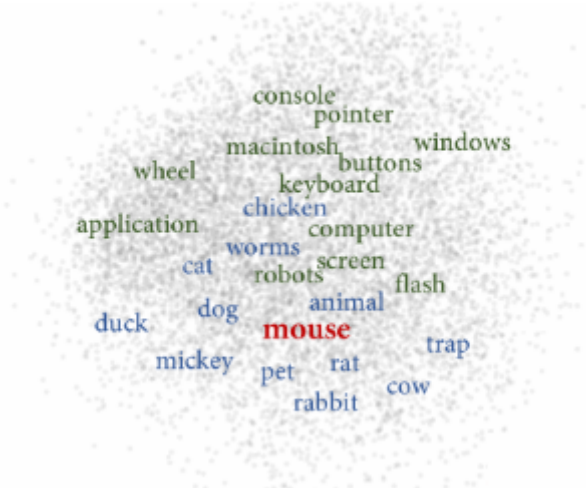

Figure 3.8: 2D representation of the neighbours of the ambiguous word *mouse*. It is possible to notice how there are words related to two main groups, animals or technological devices. Image adopted from the original paper [4].

As shown in figure 3.8 in each language there are multiple words that are defined ambiguous, or homonyms. In a *sense embedding* each one of these term will have in the final vector space multiple independent representation, one for each meaning and all different among them.
Word Sense Disambiguation (WSD in short) is a NLP job that is trying to solve the meaning conflation deficiency. Every time an homonym appears in a period the goal is to retrieve, among all the different meaning, the one that is the most related to its environment. Even if it may looks like a rather simple issue to be solved by a human being, it is instead very complex to be solved by a machine and it is still today a very open problem. Different solution have been proposed in recent years and they may be mainly split into unsupervised or supervised. In the first case, all the details required are learnt from the input corpus. To learn these representations there is the possibility to use, as input, a collection of documents written in multiple languages. This solution is based on the fact that, if a term is a case of homonymy in a certain language, it does not necessarily happens in another language too. This gives the possibility to learn the correct context of an ambiguous word from the vector of a word embedding of a foreign language. In the supervised environment, on the other hand, there is the aid of an external source which is providing more information to understand which is the correct context and direct the final outcome towards the correct representation.

3.5.1 Observations on models

The main goal of this work is to obtain a word embedding that it is able to recognize with more accuracy words belonging to a certain topic with respect to a non-contextualized word embedding. Once this has been obtained it will be used with a downstream tasks that benefits from this: for example, while extracting sentences from a set of articles during the text summarization job, it is more convenient to have an embedding trained with specialised lexicon. Indeed the training phase of these models makes use of texts related to the topic chosen and because of this the embedding will learn to map with more precision all these words that belongs to a jargon that it is rarely adopted in other contexts. In order to obtain this new contextualized vector space, continual learning has been applied to an English Word2Vec model generated using a Wikipedia corpus, a collection of every English page present in Wikipedia. The new input consists of articles coming from the Reuters 27000 collection; they have not been used all together, but they were split previously into eight different categories (art, business, economy, health, politics, science, sport and technology), obtaining so eight new different word embeddings.
A first hint to understand if the process has returned what we expected is to look at the most similar words and see what happened. A good example could be found in observing the behaviour of your model addressing words that are spelled at the same way but having different meanings depending on their context. There are two main classes to distinguish this kind of terms:

- *homographs*, words that are formed by the same sequence of characters; whenever they are pronounced the same way they are called *homophones*, otherwise *heteronyms*;

- *polysems*, words equally written having various meaning but all of them are connected. This last characteristic is the main distinction between polysems and homographs, it is very subtle and the usage of a dictionary may become in handy to distinguish them: if multiple meanings are found in the same entry they are polysems, otherwise homographs are represented by multiple entries, one for each meaning;
- *capitonyms*, words spelled the same way but they holding a different meaning when the starting character is capitalized; usually it does not represent an issue but most of the time corpora are saved with no capital letters inside and so the misunderstanding may happen.

To provide an example, here, the word *volume* will be used as a test. Volume may indicate a book, a certain amount, the level of sound or the amount of space occupied by something. The goal of contextualization is to reward those meanings that are inherent to the specific argument. For each of the following cases the twenty most similar words are shown.

```
('volumes', 0.7157266139984131)
('vol', 0.6243071556091309)
('pages', 0.5965839624404907)
('fascicle', 0.5570683479309082)
('edition', 0.5501592755317688)
('reprint', 0.527431845664978)
('hardcover', 0.5236551761627197)
('compendium', 0.5160764455795288)
('cxxxvi', 0.5112342834472656)
('paperback', 0.5095018148422241)
('folio', 0.5066821575164795)
('lxxi', 0.5035179853439331)
('compilation', 0.4998970627784729)
('miscellanea', 0.4998505711555481)
('duodecimo', 0.4992506206035614)
('mythologiques', 0.4967345595359802)
('tankoubon', 0.49598732590675354)
('book', 0.4950214624404907)
('multivolume', 0.4940647482872009)
('addenda', 0.4892631471157074)
```

Figure 3.9: Wikipedia embedding.

Besides the first word that is *volumes* every time, it is possible to observe that in figure 3.9 most of the words are correlated to the book meaning. So what happens when we are analyzing a contextualized embedding instead?

From figure 3.10 we can see what does it mean to "shift" the word embedding mapping towards a contextualized environment. Indeed on the left science-related words appear, such as *density* or *measure*, because in this embedding the target word is located in a scientific background, denoting a *volume* of a liquid or of a substance for example; on the right economy related words appear, such as *profits* or *bulk*, because in this case the target word is used to indicate the market volume. In this previous example the two embeddings have been obtained after a training phase using Reuters articles.

```
('volumes', 0.5373799800872803)
('quality', 0.5358219742774963)
('density', 0.5278875827789307)
('exceed', 0.5185091495513916)
('quantity', 0.5161540508270264)
('excess', 0.515495240688324)
('depth', 0.5132719278335571)
('performance', 0.5093227624893188)
('maximum', 0.5071604251861572)
('listening', 0.507012665271759)
('measure', 0.5020551085472107)
('content', 0.49871987104415894)
('accuracy', 0.49826517701148987)
('reducing', 0.49629268050193787)
('salinity', 0.4942089021205902)
('complexity', 0.4940725862979889)
('increase', 0.4937550723552704)
('amounts', 0.4895678758621216)
('measured', 0.48950839042663574)
('availability', 0.48896655440330505)
```

```
('volumes', 0.6766924858093262)
('size', 0.6605004072189331)
('fraction', 0.6252366304397583)
('amounts', 0.6231648325920105)
('value', 0.6174380779266357)
('overseas', 0.6056556701660156)
('commercial', 0.6024958491325378)
('profits', 0.595534086227417)
('derivatives', 0.5949854850769043)
('inventories', 0.5947299003601074)
('financing', 0.5930225849151611)
('wholesale', 0.5929096341133118)
('loan', 0.5880571603775024)
('net', 0.5857816338539124)
('packages', 0.5849630236625671)
('cds', 0.5801906585693359)
('accounted', 0.5756608247756958)
('contracts', 0.5731828808784485)
('bulk', 0.5703249573707581)
('share', 0.5701884627342224)
```

Figure 3.10: Science and Economy Reuters embedding.

What does happen if, instead, we try to use science pages coming from Wikipedia?

```
('cheilostomatous', 0.635068416595459)
('pj', 0.6325166821479797)
('polychaetes', 0.6254833936691284)
('js', 0.6048337817192078)
('bryozoans', 0.6020302772521973)
('bryozoa', 0.599591076374054)
('minéraux', 0.5815457701683044)
('molluscs', 0.5777745246887207)
('mollusca', 0.5656664371490479)
('copepods', 0.5605159997940063)
('xxxvii', 0.5476316213607788)
('poissons', 0.5457835793495178)
('oiseaux', 0.5413264632225037)
('caenogastropoda', 0.5407536625862122)
('hayward', 0.5375729203224182)
('xxxii', 0.5316126346588135)
('quadrupèdes', 0.5210169553756714)
('hydroids', 0.5202498435974121)
('ryland', 0.5199435949325562)
('leptostracan', 0.5175133943557739)
```

Figure 3.11: Science Wikipedia embedding.

In figure 3.11 is true that every words is science-related but at the same time none of them is actually linked to the scientific meaning of the word *volume*. In fact, in this scenario the target word is still correlated to the book meaning, but at this time to scientific books titles. This is a proof that the corpus you are using to contextualize your

embedding is strongly dependent on the source: a Reuters dataset will contains surely articles written by people more expert in the particular topic and this will benefit the mapping. Moreover each text is thought to be read by different public and so the lingo used will suit the level of knowledge of the reader.
Another way to perform a similar analysis, not related to a language property, is to check what happens to words that are specific of a certain argument. Indeed we should expect that specific terms are pretty infrequent in general texts and because of this usually word embeddings struggle to locate them with precision into the vector space: the mapping is based also on the frequency of a word and so if it appears just a few time the result will be highly inaccurate.
As previously, in the following figures, the most similar words are displayed. This time *syloxane*, a word related to the scientific lexicon, has been chosen as a test.

```
('acrylate', 0.7973843812942505)
('resorcinol', 0.7950678467750549)
('silane', 0.7901585102081299)
('sulfonate', 0.782994270324707)
('polyether', 0.7813056707382202)
('polyamides', 0.7769700884819031)
('polyimides', 0.7769001722335815)
('nitrile', 0.7768754959106445)
('polymeric', 0.7751662731170654)
('glycols', 0.774511456489563)
('copolymer', 0.772000789642334)
('sulfonated', 0.771863579750061)
('perfluoro', 0.7677739858627319)
('acetal', 0.7668299674987793)
('homopolymer', 0.7668240070343018)
('polyol', 0.7664928436279297)
('phenylene', 0.7663983702659607)
('pyrrolidone', 0.7646535634994507)
('polypyrrole', 0.764228880405426)
('diamines', 0.762488842010498)
```

Figure 3.12: The 20 most similar words to syloxane according to the Wikipedia embedding.

From figure 3.12 and 3.13 we can notice that all the three different embeddings are presenting as most similar most of the time the same words. This means that the Wikipedia embedding, even if it has not been trained using a domain-specific dataset, is actually able to map in a nice way terms belonging to a specific lexicon. It could be interesting to repeat the same test, using a dataset that is even more specialized with respect to ones used in this work.
The last consideration that can be done is about the distance number next to each word. Its usage depends strongly on the downstream task implemented: if the goal is to find clusters then the higher this value is the better is because it means that clusters which will be found are going to be well separated. On the other hand, speaking about the document summarization problem, the most relevant factor is the rank and not the similarity; the same word, represented in two different word embeddings, will be accounted with the same weight if its rank is 1, nonetheless the distance number.

```
('acrylate', 0.7973843812942505)      ('dimethyl', 0.834279477596283)
('resorcinol', 0.7950678467750549)    ('acrylate', 0.8224309682846069)
('silane', 0.7901585102081299)        ('resorcinol', 0.8189539909362793)
('sulfonate', 0.782994270324707)      ('esters', 0.8159806728363037)
('polyether', 0.7813056707382202)     ('pyrrolidone', 0.8136961460113525)
('polyamides', 0.7769700884819031)    ('perfluoro', 0.8095525503158569)
('polyimides', 0.7769001722335815)    ('heptyl', 0.8078374266624451)
('nitrile', 0.7768754959106445)       ('sulfonate', 0.8077452182769775)
('polymeric', 0.7751662731170654)     ('divinyl', 0.8050148487091064)
('glycols', 0.774511456489563)        ('polyoxyethylene', 0.8044319152832031)
('copolymer', 0.772000789642334)      ('tetramethyl', 0.8025466203689575)
('sulfonated', 0.771863579750061)     ('benzophenone', 0.7990791201591492)
('perfluoro', 0.7677739858627319)     ('diamines', 0.7966668605804443)
('acetal', 0.7668299674987793)        ('diphenyl', 0.7959851026535034)
('homopolymer', 0.7668240070343018)   ('difluoro', 0.7943211793899536)
('polyol', 0.7664928436279297)        ('polyether', 0.7941441535949707)
('phenylene', 0.7663983702659607)     ('benzonitrile', 0.793381929397583)
('pyrrolidone', 0.7646535634994507)   ('sulfoxide', 0.7929235696792603)
('polypyrrole', 0.764228880405426)    ('isobutyl', 0.7926014065742493)
('diamines', 0.762488842010498)       ('silane', 0.792535662651062)
```

Figure 3.13: The 20 most similar words to syloxane according to the Reuters science embedding (left) and Wikipedia science embedding (right).

Chapter 4

Intrinsic Evaluation

Word embeddings are used for a wide range of NLP related tasks, such as recognizing the argument of a document, producing summaries, question answering and many others. Most of the times these downstream tasks are hard to elaborate and they require an high computation time. Because of this, usually whenever a new word embedding is produced, first its goodness is tested using intermediate easier sub-tasks that are considerably faster; the results they will return is used to confront embeddings and understand which one is the best one. There are multiple implementations of intrinsic evaluation to be used, even though it must be taken in consideration the fact that a requirement is that the intrinsic evaluation has a positive correlation with the final task; among these we can find:

- word vector analogies; it is one of the most implemented and it consists of completing a word analogy of the form

  $$a : b = c : ? \tag{4.1}$$

  Among all the vocables existing in the vocabulary the one chosen is the one which maximizes the cosine similarity:

  $$d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||} \tag{4.2}$$

  It is possible to use semantic [1] or syntactic analogies [2], depending on which downstream will be later on applied;

- word semantic similarity; this sub-task evaluates the embedding through a set of couples of words that are considered synonymous. This test assigns a score to each couple based on the distance among them, assuming that synonyms should be located pretty close in the vector space. It is also possible to define different grades of similarities, considering how much the words holds the same meaning;

- word clustering; this method evaluates the possibility to identify clusters in the embedding space. In order to achieve this result, a dataset containing words belonging to a specified number of different topic is defined. The score used to

[1] For example: $Capital\ city_a : Country_a = Capital\ city_b : Country_b$

[2] For example: $Verb_a - ingform : Verb_a\ past\ participle = Verb_b - ingform : Verb_b\ past\ participle$

evaluate the embedding is obtained through any metric for clustering evaluation;

- synonym selection; this test consists of retrieving the most similar words to a list of input. Each input has associated a target word which is though to be the best possible outcome;
- sentiment analysis; this analysis has as goal to understand which are the beliefs of a customer through reviews and surveys

4.1 Glossary evaluation

Unfortunately at this moment there is no data set available to test a domain-specific embedding with the techniques introduced before. Because of this, to test the quality of the embedding created a different approach has been used, based on a topic-oriented glossary, exploiting PetScan 4.1, a website that it is able to retrieve the content of Wikipedia pages.

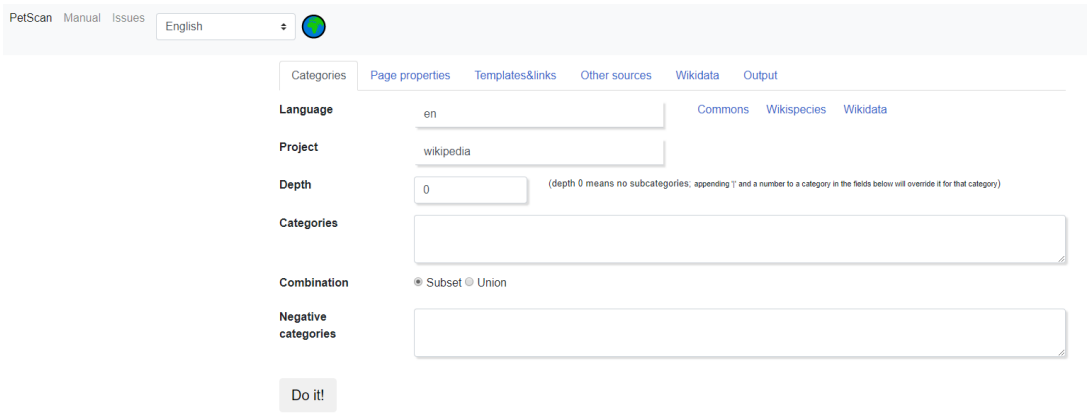

Figure 4.1: Petscan main web page.

It offers various options, among these:

- language: search only among Wikipedia pages written in the selected language;
- category: extract pages that are labeled with the selected category;
- negative category: ignore pages that have this among their labels, even if they satisfy the previous category constraint;
- depth: it is possible to build a tree starting from each Wikipedia page: the root will be the page itself and then its children will be all the pages that are accessible from the links contained. This process can be repeated an arbitrary number of times and this value represents the maximum depth that this tree can reach. So, setting this value to a number different from 0 will imply that in order to extract a page, not only the page itself will be considered but also all its children, always applying the same constraint described in the previous points.

Thanks to PetScan, it is possible to collect a large set of Wikipedia pages with the same argument of the articles that have been used to train the contextualized word embedding and then from it build a glossary. Each page will be an entry of the glossary and

the body of the definition will be made of the first two sentences contained, considering that usually Wikipedia articles are written in such a way as the definition or the most meaningful information are located in the immediate beginning. The two first problems related to this approach are:

- definition of a category, unfortunately there is not a strict segmentation of categories, specially when dealing with arguments that are very wide, such as technology or science, where they overlap most of the time because no rigid rules are enforced. Indeed different newspapers or websites could label articles describing the same topic in different ways and this aspect loosen the level of contextualization of the word embedding. A good solution to this could be to narrow the category to something more specific, like natural disaster, thereby it can be detected easily, but the following problem prevent to follow this path;

- number of entries in the glossary, in order to obtain a result that holds a strong meaning and that is not much biased, the glossary should contain an high number of definitions. The number of articles filtered using PetScan is determined by the depth value: the higher it is the larger the number of page will be. Increasing this parameter has a major drawback due from the fact that the root page is less and less related to a page the deeper that page is in the tree. This means that using a large number for depth is not a good approach (and already 2 is large enough in this context). For example, a search performed among the English politics pages with depth 2, returns "Drogowskaz" as a result, that is a font type used in public signage in Poland. Poland is the first link that can be run through and then in the Poland's page there are a lot of links that bring to a politic related article.

4.1.1 Results

The metrics that have been used to evaluate the performance of this approach are the typical ones used for addressing the results about information system retrieval system. This kind of metrics, called evaluation measures, are used to understand how well the search result satisfied the user's query intent, such as a search engine results page. In this scenario the query is represented by the definition and the search result by the list of most similar words to it. A positive outcome for this problem is whenever many words retrieved in this way are contained in the body of the definition but also their rank is relatively high. Usually the importance associated to each words decreases as soon as the rank decreases, so that the first word is significantly more relevant than the fifth one for example. These are the measures that have been used:

- recall at K: recall is the percentage of words contained in the most similar list that are also in the body of the definition. Due to the fact that this list might be composed by as many words are in the dictionary it is important to limit it to the first K elements, in this way we are basically just considering the recall related to the most meaningful words that are also the one we do only care about;

- precision at K: precision is the percentage of words contained in the body of the definition that are present in the most similar list, always limited to K elements for the same reasons as before;

- MMR: mean reciprocal rank is given by the following formula

$$MRR = \frac{1}{|Q|} \sum_{|Q|}^{i=1} \frac{1}{rank_i} \tag{4.3}$$

  The reciprocal rank is the multiplicative inverse of the rank, meaning that the lower the rank is, the lower its score will be.

For both recall@K and recall@K three values of K have been computed: 5, 10 and 20. Previously higher values of K have been considered but the importance associated to a word that holds the fiftieth position, for example, is so low that is negligible. Unfortunately, as we can see from the graphs reported from figure 4.2 until figure 4.8, for each metric chosen and for each value of K the original embedding, that is the one obtained using all the pages present in Wikipedia, outscores the two contextualized models involved in this study. The first problem about this glossary evaluation is that all the definition have been extracted from Wikipedia pages. Wikipedia is a platform in which every article is written by users and so it is very likely that the language used is somewhat different from the one used by journalists in their jobs. Indeed it would be very interesting to repeat the same kind of test using a specific glossary, hand-written by someone who has a knowledge in the selected topic. In this way it is possible to address all the issues arisen by the use of PetScan. A similar solution has been performed in this study [26]. They have used a domain specific glossary, the Schlumberger Oilfield Glossary `https://www.glossary.oilfield.slb.com/`, containing definitions, synonyms, antonyms and labeling each word to its discipline. They reported positive results about their model evaluation and this could also signify that instead of contextualizing a word embedding to an extensive topic such as *politics* it may be better to shrink it to a more selective scope. For example we could select *natural catastrophes* as a topic to train a word embedding model but this represents a rather hard challenge: not only the dataset needs to be large enough but we must guarantee that almost every word is also present in this new dataset otherwise the new model is going to be able to recognize correctly only words related to natural events and it will produces large error according to all the other terms. A further explanation to this result is that the contextualized embedding has been trained using only four-five thousands articles for each topic: this is an abysmal number with respect to the number of input given to the Wikipedia embedding; probably a bigger number of training samples could *shake* more the word representation and provide a more meaningful mapping. The contextualized models have been obtained after a single epoch of training; anyway increasing this said number the results obtained are basically identical; as explained in 3.5 increasing too much the number of epochs is leading to scores which are each time worse.

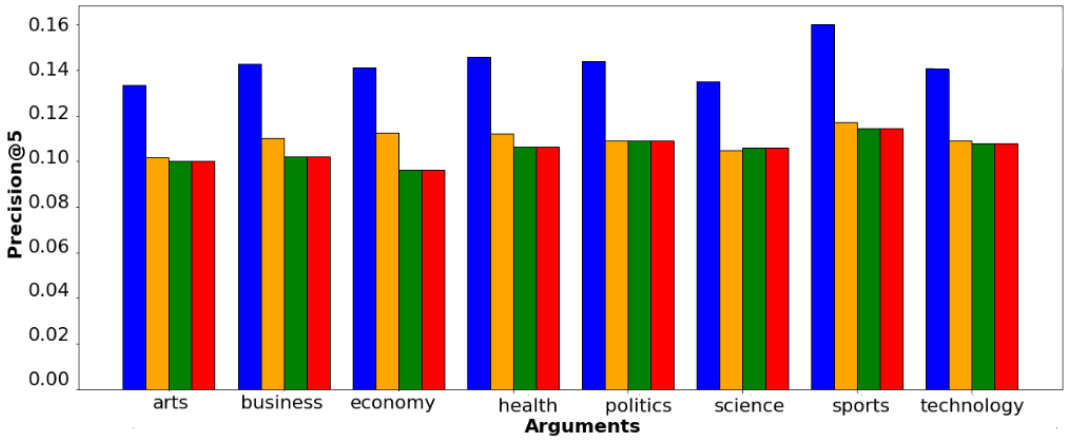

Figure 4.2: Precision at 5 for the glossary evaluation. These are the colors used: Wikipedia - Blue, Wikipedia single domain - Yellow, Reuters 27000 - Green, Reuters 27000 single domain - Red.

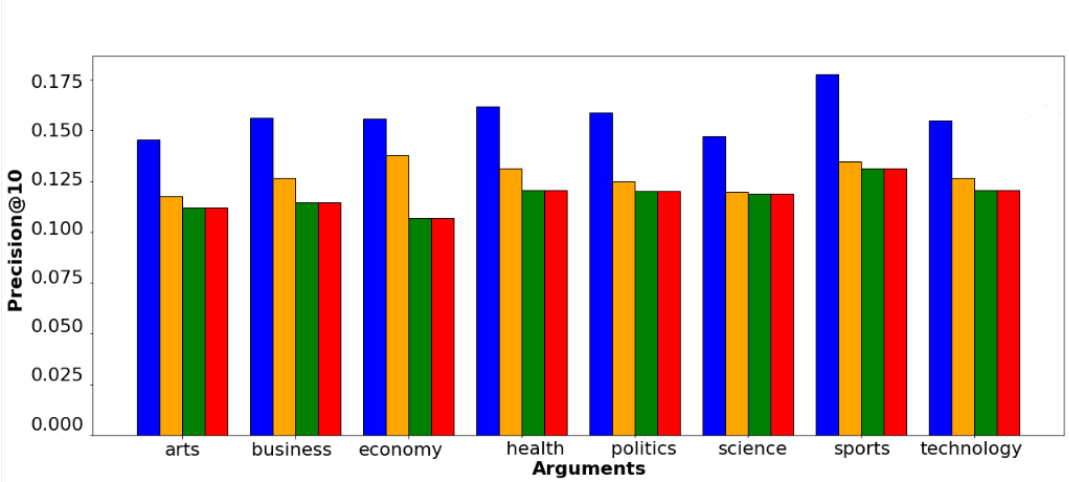

Figure 4.3: Precision at 10 for the glossary evaluation. These are the colors used: Wikipedia - Blue, Wikipedia single domain - Yellow, Reuters 27000 - Green, Reuters 27000 single domain - Red.

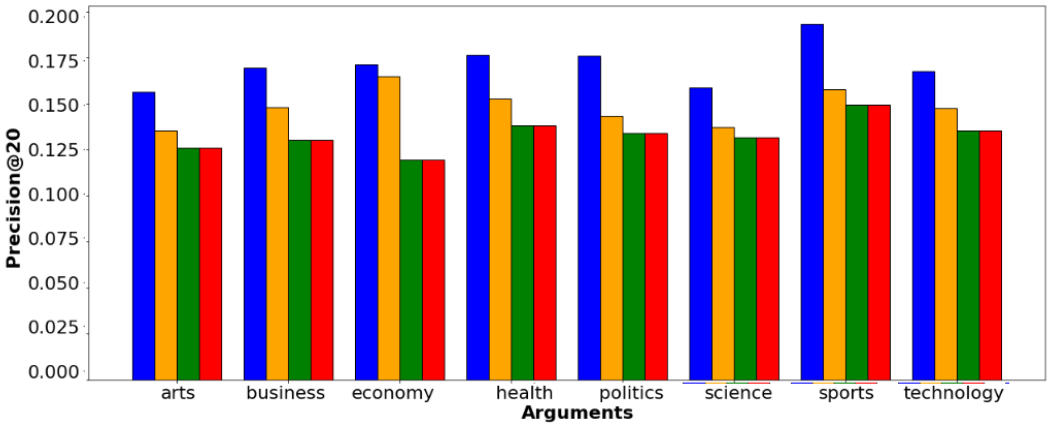

Figure 4.4: Precision at 20 for the glossary evaluation. These are the colors used: Wikipedia - Blue, Wikipedia single domain - Yellow, Reuters 27000 - Green, Reuters 27000 single domain - Red.

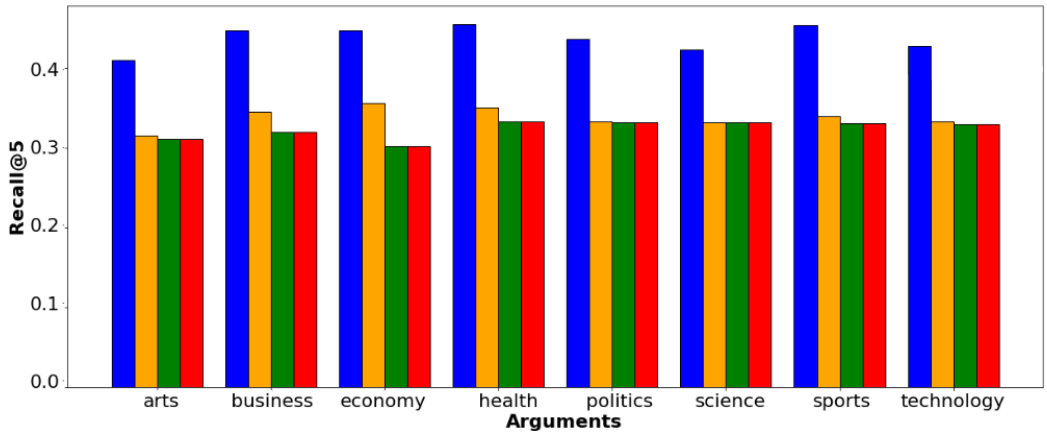

Figure 4.5: Recall at 5 for the glossary evaluation. These are the colors used: Wikipedia - Blue, Wikipedia single domain - Yellow, Reuters 27000 - Green, Reuters 27000 single domain - Red.

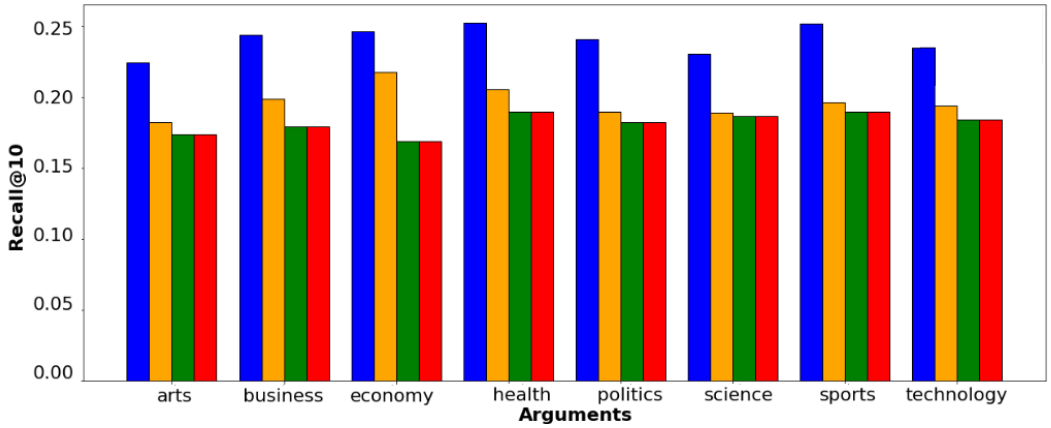

Figure 4.6: Recall at 10 for the glossary evaluation. These are the colors used: Wikipedia - Blue, Wikipedia single domain - Yellow, Reuters 27000 - Green, Reuters 27000 single domain - Red.

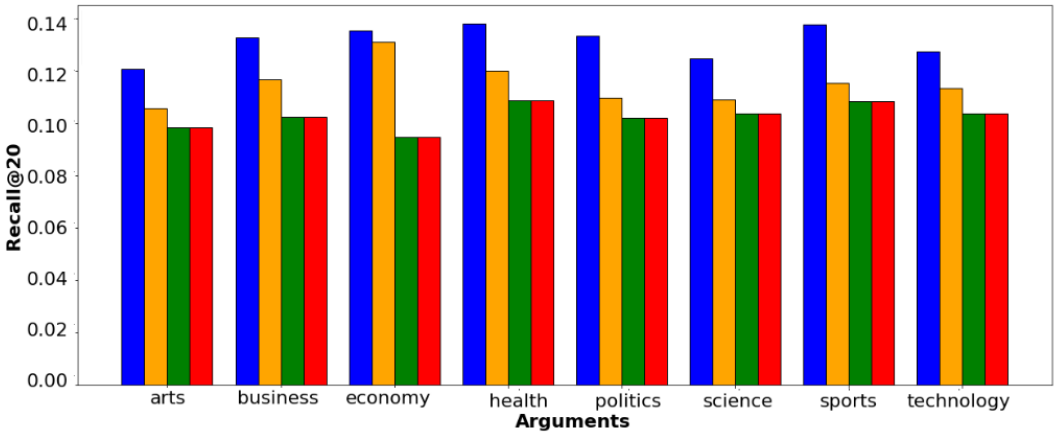

Figure 4.7: Recall at 20 for the glossary evaluation. These are the colors used: Wikipedia - Blue, Wikipedia single domain - Yellow, Reuters 27000 - Green, Reuters 27000 single domain - Red.

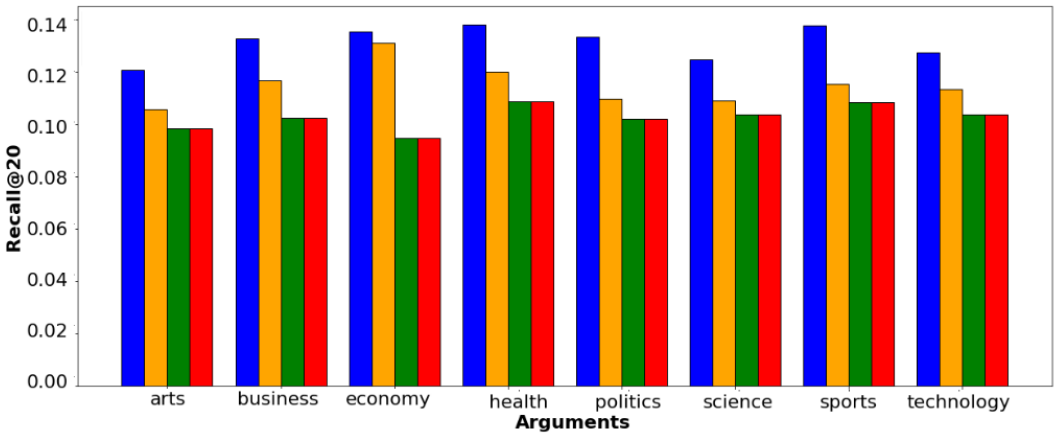

Figure 4.8: MRR for the glossary evaluation. These are the colors used: Wikipedia - Blue, Wikipedia single domain - Yellow, Reuters 27000 - Green, Reuters 27000 single domain - Red.

Chapter 5

Extrinsic Evaluation

The start of this chapter signs the beginning of the second phase of this work. So far we have dealt with word embeddings and understood their goodness, but they are not very useful by themselves. Their main purpose it is, indeed, to be tuned into an *extrinsic task* which represents a problem that NLP technologies are trying to solve. The majority of these problems is related to a classification problem. A classic example is the *named-entity recognition* (NER) in which the model is built in such a way it is able to assign the correct class label to named-entities, such as person name, organization, time expressions, monetary values, etc.

5.1 Document summarization

Document summarization is the extrinsic evaluation method that has been chosen for this work. The goal of this task is to process a collection of texts in order to retrieve the most meaningful sentences, that are going to compose the summary. While implementing it, a length constraint needs to be taken in consideration: in fact the summary cannot exceed a prefixed number of words or sentences. Each sentences is going to be extracted such that at the end the summary will contain as many information as possible also avoiding repetitions. The first way to distinguish various techniques of summarization paradigms is to look at how the summary has been made; *abstractive* summarization will produce a result without using words and sentences coming from the original text; on the other side *extractive* summarization, as the name suggests, will indeed *extract* sentences from the document chosen and the result will be obtained by a concatenation of them.
The starting point of document summarization is, as said before, a collection of documents. Nonetheless it is possible to have just a single text or a set of them. Depending on this aspect, document summarization is implemented in different applications to satisfy different requests. *Single-document* summarization is characterized by the usage of a single text as starting point. It is implemented for example in:

- *question answering* has as a goal building a system that is able to answer to questions written by humans. This task is divided in two main steps: the first one is to "understand" what the question is about and then retrieve a suitable answer. To achieve this objective, the system has to query a database, that acts as a background knowledge, containing information related to the specific question (such

as a collection of articles from newspapers or a set of web pages);

- *providing recommendations* tools are implemented each time an application requires to understand which are the interests of a person and suggest accordingly new materials that is inherent. A common example is given by the system of recommendations that can be found in newspapers web pages. Their aim is to constantly provide new articles that are similar to the current one. To fulfill this task, the system needs, first, to label correctly the article that is displayed in that moment and then understand which others papers are similar to it.

On the opposite, whenever the starting point of the process is made of multiple documents, the summarization task is called *multi-document*. In this other scenario the summary produced will be obtained gathering sentences obtained from different texts: this procedure allows to ensemble an output that will contains different point of views, coming from various sources and will provide to the reader a set of information more waste with respect to the single-document scenario. Thus, it is possible to have in the same summary discordant opinions on the subject treated; this should be not seen as a drawback but rather as a different outcome for the same problem. Nonetheless the result will be computed through an algorithm, ensuring it will be completely unbiased. Having to deal with different documents at a time, this task represents a harder challenge with respect to the previous scenario because the aim is not only to short the content, to provide a summary, but it has to collect pieces of information from multiple views.
Finally, the last way to discretize the document summarization problem is concerning the topic involved. Indeed not every text has been written in the same way and for the same purposes: we have documents produced for the scientific audience, news articles or magazine and many others. Because of this variety of content, it is possible to distinguish two many categories: *general purpose* and *domain-specific*. In the domain specific case all the articles chosen to be elaborate have been written most of the time by people with a high level of knowledge of the argument itself; thanks to this aspect the content will be aimed to a limited audience and the lexicon used will be hard to understand to any person not familiar with the topic. Terms used and the way the sentences are composed are very different from argument to argument and this is the reason why models developed to solve this problem are different from the general purpose ones. In this latter case all the articles involved are intended for a waste public and so it is very unlikely to encounter specific words or anyway words that are not familiar with most of the people.

5.2 Sentence selection

In this work the approach chosen is the extractive one and because of this the main step of the algorithm provided is the sentence selection. As said before, this is the crucial point of the document summarization because it will decree which periods will form the final summary. In the following paragraphs the main techniques will be presented, with a bigger focus on the one actually used during the experiments.

5.2.1 Cluster based

Each document is usually written providing a differentiation among all the topics presented. Instead of having a single body of words, writers will organize their work in paragraph or sections. For example every time a reader finds a blank space wider with respect to the common usage, he will immediately recognize that something is going to change in the next few lines. The cluster based approach bases its functioning on top of a topic detection phase. *MEAD* [29] presents a *Topic detection and Tracking* phase to scan all the documents in the dataset in order to create different clusters representing the various arguments touched by the corpora. This study is performed with the aid of the TF-IDF measure that provides a score of similarity used to establish to which cluster each object belongs to. As soon as this starting phase has terminated, with the *centroid-based summarization* the most relevant sentences will be selected. The *centroid* in a cluster is the average of every elements belonging to it and because of this property it is usually declared as the representative point of the group. This algorithm is then able to guarantee the coverage of every argument present in the collection thanks using the most important sentences coming from each different cluster.

5.2.2 Frequent weighted itemsets

Frequent itemset represents one of the main problem dealt by data mining studies. An itemset is a group of items belonging to a transactional dataset and the main interest is to select only those ones which have a frequency higher than a certain fixed threshold. This analysis may be, for example, performed by a market which is interested in discovering which items are the most sold and in which combinations. The concept of weighted itemsets has been introduced in order to consider each object in a different way, within each transaction, depending on its relevance. It is possible to apply the same methodology to retrieve and select the most important sentences from a document: the collection of document will be our database of transaction and each document may be interpreted as a transaction or a sequence of words. So, in the document-summarization case, each itemset will be formed by terms and the aim is to understand which groups of them appear most of the time. This solution may lead to an abstrative summarization task (Hynek and Jezek [11]) or to an extractive one (Baralis et al. 2012 [2]).

5.2.3 TextRank

TextRank is one of the most used algorithm while dealing with document summarization. It is derived from another very famous algorithm, PageRank, developed by Google [27] in order to rank pages while performing a research on their search engine. The goal of this process is to assign a score to each page, depending on the probability of a user to click on it. The most basic way to evaluate it is to consider every links contained in every page, except link that are pointing to the page itself. PageRank is firstly initialized to the same score, corresponding to 1 divided by the total number of pages. After this first initialization phase, each page will "transfer" a portion of its own score to every page that is directly accessible trough a link; whether more links are contained in it then this propagated value will be divided by the number of links. This propagation basically represents the flow of a user that is surfing on a specific page

and then he is deciding to change page by clicking on a link. The general formula to express the PageRank value of a web-page w is the following:

$$P(w) = \sum_{l \in B_u} \frac{P(l)}{s} \tag{5.1}$$

where l is a page accessible through a link in the set B_u (all the pages accessible) and s is the total number of links contained.

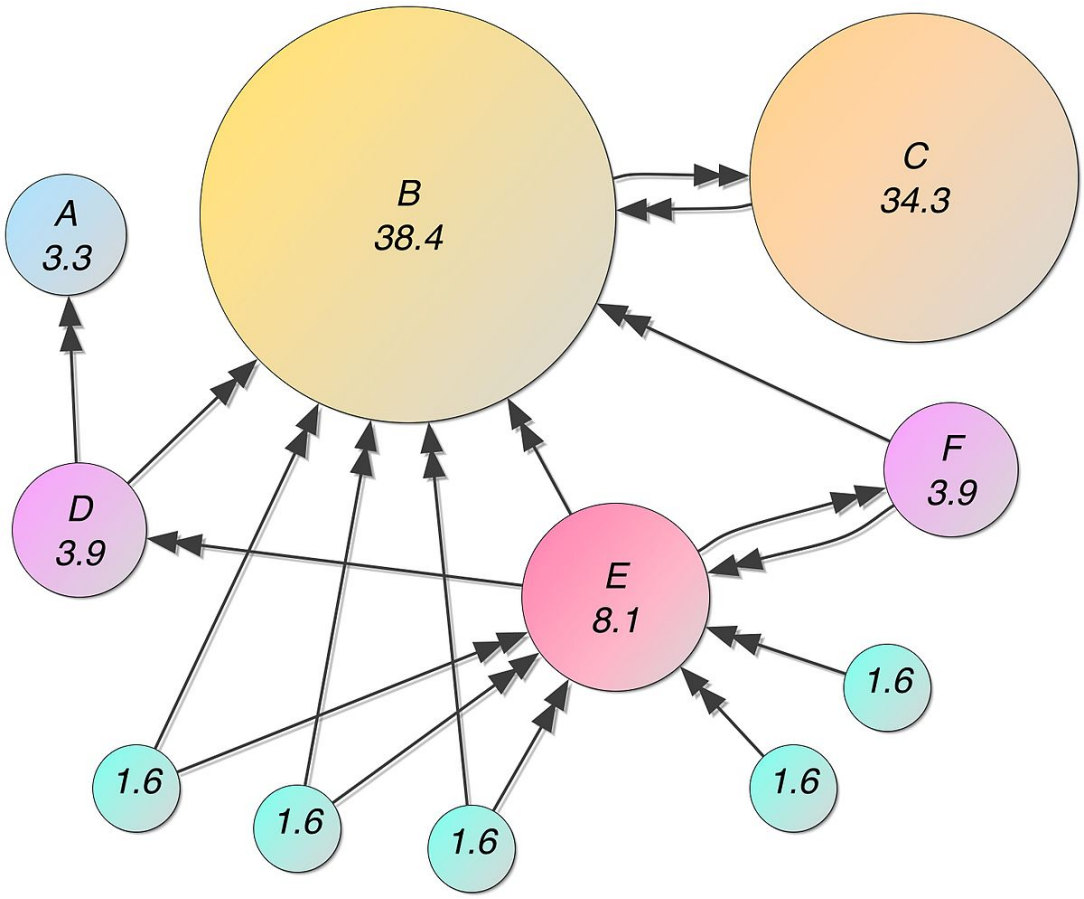

Figure 5.1: Visual representation of the graph obtained at the end of the PageRank algorithm. The larger the circle (vertex) is the higher its score associated is. Each edge is a link between two web pages. Image taken from the Wikipedia page https://en.wikipedia.org/wiki/PageRank.

It is possible to notice from figure 5.1 that the vertex with the highest score associated it is also the one having the most links pointing towards it. Edges are oriented, because the presence of a link towards a page does not guarantee to have another one going backwards in the destination page.
TextRank [21] bases its own working on top of the previous algorithm, considering that the focus at this time is put on sentences and not on web pages. Indeed, the probability described before is now replaced by the similarity between two sentences: the higher the number (N) of words shared among them is, the closer they are. This

similarity, given two periods S_i and $S_j$is then defined as:

$$similarity(S_i, S_j) = \frac{N}{log(|S_i|) + log(|S_j|)} \tag{5.2}$$

As previously, a fully connected graph (figure 5.2) is build and each edge E is initialized to the similarity score between the two sentences (represented by vertices V in the graph, initialized to 1). The value of each vertex is then updated taking into consideration neighbor vertices, scaled according to the edge score. The score of a vertex V_i is defined as:

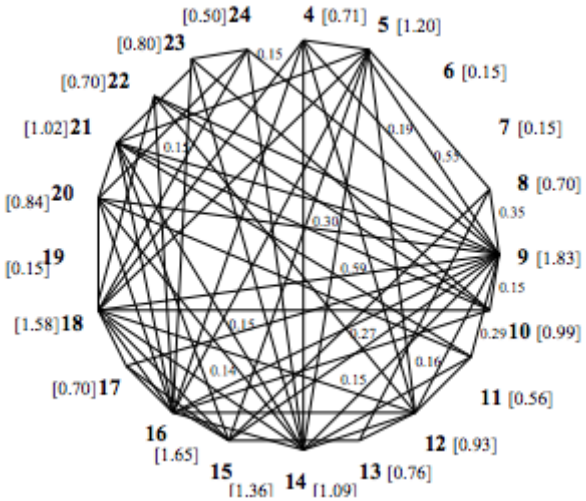

Figure 5.2: Graph that describes both the TextRank and the LexRank algorithms. Each vertex is a sentence and each edge has as weight the similarity score. The value written in square brackets is the score used to draft the ranking. Image taken from the original paper [32].

The main difference between the graph 5.2 and the graph 5.1 is that here there is an edge between each couple (this could be obtained if every web page has a link for each other page).

$$S(V_i) = (1-d) + d * \sum j \in ln(V_i) \frac{1}{|Out(V_j)|} S(V_j) \tag{5.3}$$

where d is a damping factor, usually set to a number between 0 and 1 (it is equals to 0.85 in the original paper [21]) and it is used in order to introduce the concept of a "random surfer model", described before, that is the probability to navigate from a page into another one (in this case from a sentence to another). This operation is is repeated until the convergence or until the change of values between two iterations is below a certain threshold. When the convergence is obtained sentences will be ranked according to their vertex score associated and they will be picked until the maximum number of words is reached.

5.2.4 LexRank

LexRank [7] is another algorithm that belongs to the family of algorithm derived from the PageRank one. The main difference to the previous approaches is to research in the definition of similarity between two sentences. There the value associated to each edge is provided by the following formula:

$$idf - modified - cosine(x,y) = \frac{\sum\limits_{w \in (x,y)} tf_{w,x} tf_{w,y} (idf_w)^2}{\sqrt{\sum\limits_{x_i \in x} (tf_{x_i,x} idf_{x_i})^2} X \sqrt{\sum\limits_{y_i \in y} (tf_{y_i,y} idf_{y_i})^2}} \tag{5.4}$$

where x and y are two sentences, tf is the term w frequency and idf is the inverse document frequency. The TF-IDF measure is often used while dealing with analysis of documents and that is the reason why has been used there. This measure takes into account not only how many times a term is repeated through the corpora but also in how many texts is found. This is fundamental because with this consideration a word that appears in most of the documents has an higher value associated with respect to a word that is present a lot of times but just in one text. Besides this aspect the algorithm is the same one explained in the paragraph 5.2.3.

5.2.5 Latent semantic analysis

Latent Semantic Analysis (LSA) is an algebraic-statistical method used while performing text summarization in order to retrieve the most relevant topics related to a collection of documents. LSA works in an unsupervised environment and it learns which words are the most impactful depending on their frequencies and on their usage, that is, the more certain words are found together in the same sentence, the more they will be correlated. Meaningful sentences are then decided based on the importance of each word they contain. To implement this algorithm Singular Value Decomposition (SVD) is used to obtain a representation of your dictionary and also to reduce the noise related to the problem itself. A term-document matrix A is used as input for the SVD. There are different possibilities to compute this matrix given a collection of documents; in this implementation each row m represents a document, each column n is a word of the dictionary and each entry is how many times that particular words is contained inside that sentence. SVD will produce three matrices as output:

$$A = U \Sigma V^T \tag{5.5}$$

where U is an m x n matrix containing words' weights related to each topic, Σ is an n x n matrix containing the importance of each topic and V^T is an n x n matrix containing sentences' weights related to each topic. Once this result has been obtained it is possible to select the most relevant sentences according to it. Again, there are different approaches to choose them because there are different criteria, i.e. selecting the most important sentence for each topic. Here the approach that has been selected is the one presented by *Steinberger and Jezek (2004)* [31], which states that scores, used during the selection phase, are evaluated as the weighted sum of each sentences with regard to

every topic; with this implementation a sentence that has an high score on a non relevant topic will be chosen after a sentence that has a lower value on a more relevant topic.

5.2.6 Other algorithms

Besides the algorithms explained in the previous paragraphs, summaries have been extracted using also algorithms implemented by L. de Haas thesis [6]. Below is a briefly explanation of them.

Not embedding-based summarization algorithms

- *LEAD*, this is one of the most basic algorithm and its based on the concept that the most meaningful information are contained in the first sentences of an article. Because of this it will select the first n sentences, given a constraint that may be regarding the number of words or the number of sentences itself. The rank is merely chosen on their order;
- *Maximizing semantic volume* [33], instead of maximizing the coverage of original text a new maximizing objective has been introduced. The result of MSV is found in the subset of sentences of which the convex hull 5.3 in semantic sentence space is maximal. A convex hull of a shape is the smallest convex set that contains it; it belongs to the NP-hard class;

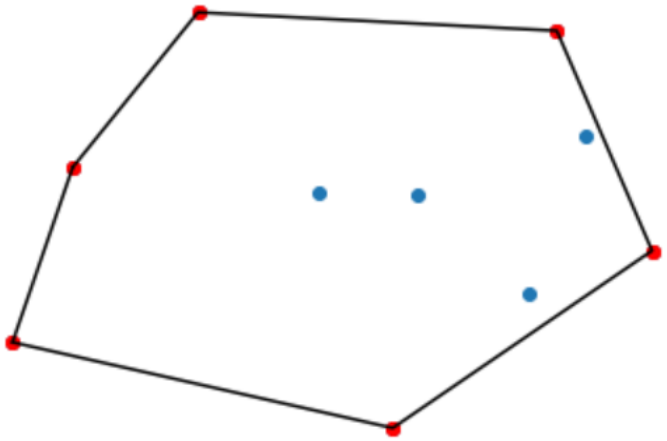

Figure 5.3: A convex hull; each dot is a sentence, the subset chosen as a solution is made of red dots touching the line.

- *Greedy submodular optimization* [18] [19], this algorithm exploits the concept that, each time a summary is extrapolated the goal is to retrieve the largest percentage of information avoiding as most as possible repetitions. Both this aspects, informativeness and diversity, are naturally modelled as monotone submodular functions. This means that every time a new sentence is added to the summary its value will be smaller with respect to the previous ones. Again this maximizing problem belongs to the NP-complete class and because of it these two approaches are implementing a greedy approach (algorithm 2) to resolve it.

Algorithm 2 Greedy algorithm by Lin and Bilmes (2010) [18].

Input: Article sentences V, scaling factor r, threshold B and monotone submodular function $f(\cdot)$.
Output: summary periods.

$G \leftarrow \emptyset$
$U \leftarrow V$
while $U \neq \emptyset$ **do**
  $k \leftarrow \arg\max_{l \in U} \frac{f(G \cup l) - f(G)}{(length_l)^r}$
  if $\sum_{i \in G} (length_i) + length_k \leq$ **and** $f(G \cup l) - f(G) \geq 0$ **then**
    $G \leftarrow G \cup k$
  end if
  $U \leftarrow U\, k$
end while
$v* \leftarrow \arg\max_{v \in V, length_v \leq B} f(v)$
return $G_f = \arg\max_{S \in v*, Gf(S)}$

In this algorithm r is a scaling factor introduced to assign a lower score to longer sentences, because it is possible to have multiple shorter periods covering the same concept. The threshold B used in document summarization is of 100 words. The last step of the algorithm is to consider what to return, between the set of sentences extracted or a*singleton summary*, made of a single sentence achieving the best score. In the 2010 paper [18] the submodular function used is the maximal marginal relevance (MMR in short); the following year (2011 [19]) they have introduced new functions focusing on how to retrieve the most information while maintaining the highest possible degree of diversity.

5.3 ROUGE

Recall-Oriented Understudy for Gisting Evaluation, or ROUGE in short, is a set of metrics used to evaluate automatic summarization and machine translation in natural language problems. It makes a comparison between an automatically generated summary and one or more human-produced references. There are different available metrics, depending on how the comparison is performed, and for each one of these precision, recall and F-measure has been evaluated. Even if it is the most adopted metric the main problem concerning it is that it bases its own functioning on word coverage and not on "information" coverage: this means that two sentences containing synonyms instead of the same words will score poorly, while a human would be able to correctly identify those summaries as pretty similar. There are many sub-categories of this metrics; among these:

- ROUGE-N measures how *n-grams* are overlapping. An *n-gram* is a continuous sequence of words of length n. This latest number is decided arbitrarily but the larger it is the harder it is to reach high values of precision: very long series of

words will be compared among them. *N-gram* precision is computed as follows:

$$p_n = \frac{\sum_{C \in Candidates} \sum_{n-gram \in C} Count_{clip}(n-gram)}{\sum_{C \in Candidates} \sum_{n-gram \in C} Count(n-gram)} \tag{5.6}$$

- ROUGE-L measures the longest matching sequence of words, the longer this sequence it is the more similar the two summaries will be;
- ROUGE-SU4 tries to overcome the rigidity of having the words in the same order, indeed it allows to re-organize a sentence to find matches. This means that it is possible to have bigrams interleaved by other words (4 in this case) and that will still be considered as a positive outcome. It also takes into account unigram in common between two sentences.

Chapter 6

Summarization Based on Embedding Models

In this chapter embedding-based algorithm will be introduced and explained. This category of algorithm represents the main objective of this work. As explained previously, a domain-specific word embedding should have a better mapping with respect to a general purpose one. Because of this characteristic it is possible to suppose that implementing them in a document summarization task could provide some kind of benefits. Naturally, the documents of which the summary has to be extracted has to be inherent with the same topic of the corpus used to train the word embedding: this means that a dataset containing newspaper articles, for example, will be handled in a better way by a model trained using Reuters or Google News. The algorithms that will be in short explained are an "enhanced" version with respect to the basic form; domain-specific embeddings are used to provide an ulterior information which may be exploited in the process to boost the performance.
As explained in the chapter 3 there are many approaches to achieve a word or sentence embeddings: this variety introduces many diverse paths to follow, while exploring the possibility to implement them in the document summarization.
Results obtained with all these implementations will be firstly compared with the algorithms explained before; then in a second moment there will be an analysis aimed to check the variations of performances among all the word embeddings presented in this work.

Textrank

The main procedure and implementation of this algorithm has been already elucidated in the paragraph 5.2.3. Here there is an attempt to enrich this model with details coming from word embeddings. Indeed, this time the definition of distance is not given by the equation 5.2 or by the TF-IDF distance (used, instead, in the LexRank approach), but by the distance among vectors representation of sentences. In order to evaluate this distance it is possible to apply any of the method illustrated in the paragraph 3.4; each one of those will end in different result and so it is likely that using a different text similarity will lead to picking other sentences to shape the summary. In this work the cosine similarity has been chosen; although is one of the most basic approach it has been widely implemented and its results are acceptable. It may be

worth to explore other similarity metrics and study how they can impact the final result. In the results three voices of TextRank are shown: one obtained through the basic Word2Vec model, another one trough an embedding re-weighted with TF-IDF and the last one through a SIF embedding.

Kågebäck

Kågebäck et al. (2014) [12] have introduced a new method to evaluate which sentences are the most relevant. This algorithm is an "extension" with respect to the greedy sumbodular one described in 5.2.6, because it introduces all the benefits coming from the use of a word embedding. Indeed word embedding representations have been used instead of using TF-IDF vectors. The easiest way to obtain the representation of a sentence is to average all the word vectors that are composing it. In this other scenario instead, there is an attempt to take into account also in which order these terms are appearing. To satisfy this request an unfolding recursive auto-encoder (*uRAE*) has been used, introduced by Socher et al. (2011) [30]. Its functioning (figure 6.1) is based on two phases similar to the ones explained in merit of the auto-encoder structure, in section 3.3. Sentence selection has been implemented this time, using two definitions of similarity among periods; the first measure is based on cosine similarity

$$Sim(i,j) = \left(\frac{x_i^T x_j}{\|x_j\| \, \|x_j\|} + 1 \right) / 2 \tag{6.1}$$

while the second one is based on the complement of Euclidean distance

$$Sim(i,j) = 1 - \frac{1}{\max\limits_{k,n} \sqrt{\|x_k - x_n\|^2}} \sqrt{\|x_j - x_k\|^2} \tag{6.2}$$

where x is the vector associated to a sentence.

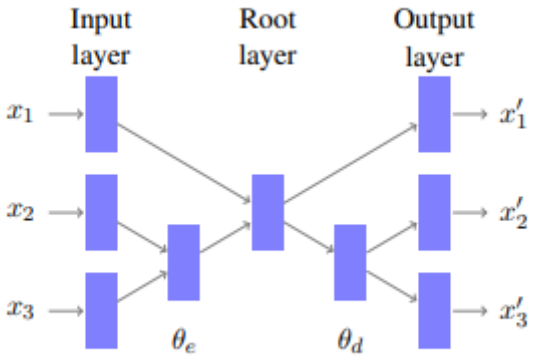

Figure 6.1: Structure of the uRAE architecture. The input layer is compressed into a code (root layer) and then used to produce the final output layer. Image taken from [12].

DocEmb & EmbDist

Similarly to the previous attempt, Kobayashu et al. (2015) [15] have introduced two implementations which are trying to improve the greedy submodular optimization, but this time using a different objective function. DocEmb, first, evaluates a document embedding produced by the sum of all the words present in a text, ignoring sentence boundaries. The objective function is given by the cosine similarity between the representation of the document and the summary chosen as benchmark:

$$f^{obj}(C) = \frac{v_S v_D}{\|v_S\| \, \|v_D\|} \tag{6.3}$$

where $\|v_S\|$ and $\|v_D\|$ are the summary and document vector. It is possible to notice how this time the main unity of the process is not a sentence or a word but instead the whole document (the origin of the name *DocEmb* is indeed coming from this aspect). However this function is not submodular and that is why EmbDist has been introduced. It does resolves this issue introducing anyway a more complex solution: this time the objective function is the inverse average distance between every sentence in the document and its most similar summary sentence:

$$f^{obj}(C) = -\sum_{s \in D} N(s, S) \tag{6.4}$$

where *N(s, S)* is the distance between a document sentence and its closest sentence in the set of summaries S. There is also another version in which the focus is placed on words instead of sentences;

RNN

Cheng and Lapata (2016) [5] took a different approach, which exploits a recurrent neural network combined with a single-layer convolutional neural network (figure 6.2). It is possible to apply this method for both the abstractive and extractive summarization tasks (only the second one has been taken into account in this work).

This architecture works as a document encoder in which each input text is analyzed in order to discern which sentences are meaningful and which are not. From the figure 6.2 we can distinguish three main tasks:

- the *sentence encoder* is a convolutional neural network in charge of providing a representation to each sentence. To obtain the final outcome, multiple vectors are computed and summed, using many kernels to obtain many feature maps and "observing" the input period by many point of views;
- the *document encoder* is a recurrent neural network that receives data coming from the sentence encoder and elevates the analysis from the level of sentences to the level of documents. A document vector is indeed obtained by gathering together the representations associated to each sentence which is made of. Each hidden state (green rectangle containing the *h* followed by a number) is a non-complete representation of the text that is going to be enriched at each step by a new incoming sentence (green rectangle containing the *s* followed by a number); at each step there is a tendency to give more relevance to the last period with

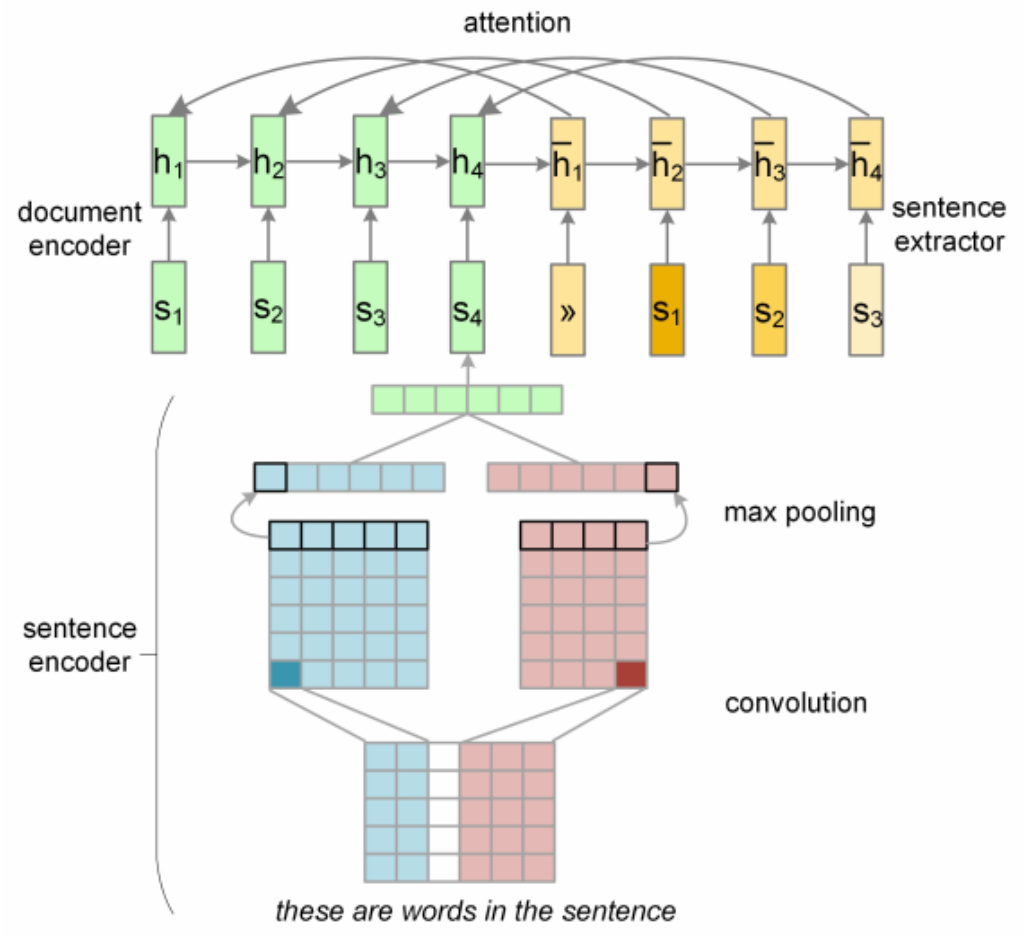

Figure 6.2: Structure of the RNN proposed by Cheng and Lapata (2016) [5]. Image taken from the original paper.

respect to the previous ones. This set of states is the global representation of the document;

- the sentence extractor is another recurrent neural network that bases its own working on an attention mechanism that will determine which periods will be chosen to compose the final summary. This mechanism is used to individuate which portion of the text is the most relevant.

Extracting a summary using this method is similar to a segmentation problem: the goal is to assign a label to each sentence depending on how much they are relevant; in this case three classes are given:

- 0 - the sentence should not be extracted;
- 1 - the sentence should be extracted;
- 2 - the sentence may be extracted.

In order to train the neural network, previously, each sentence has to be labelled. This

process can be performed manually or otherwise it is possible to build a ranking system in which the order is given first by ROUGE-2 and then in case of equality by ROUGE-1. After this list has been computed the first third will be labeled with the class *1*, the second one with *2* and the last one with *3*.
This approach has been modified in [6] from the original version proposed, in order to adapt it to a an extractive summarization problem. The input is indeed made of static sentences embedding, obtained through the average of word embeddings of every term contained in the period.

Chapter 7

Experimental Results

The last experiment done is about producing summaries on different datasets, using many algorithms: some of them will exploit a word embedding and some of them will not. The first objective is to evaluate their performance, understanding which sentence selection method achieves better results, taking in consideration also the presence or the absence of a word embedding to obtain the outcome. After this, there is a brief comparison between two models, both trained using two corpora composed of Reuters articles to evaluate how much the number of input samples impact the goodness of a word embedding. The last objective is about, using three test datasets, underlining the differences in performance using five word embeddings, one general-purpose and the other four domain-specific. This is the most relevant result gathered: having bigger scores related to domain-specific models could empirically show that their usage is more suited to summarize texts dealing with a particular topic.
To evaluate all these scores it is requested to have:

- word embedding trained using the domain-specific corpus;
- SIF word embedding trained using the domain-specific corpus (facultative[1]);
- matrix containing the final state of the RNN trained using the domain-specific corpus(facultative[2]);
- test datasets with two collections, one for the articles and the other for the summaries.

The program works in a pipeline fashion: first all the articles will be pre-processed (as explained in 2.1); this first intermediate results is used as input by all the sentence selection methods, which will return an ordered list of sentences, ranked by relevance; last, these summaries are compared with the ones hand-written using the ROUGE metric.
The code has been written using Python version 3.7, besides the ROUGE library (version 1.5.5) which is implemented in Perl. All the experiments reported in this work were run on a machine equipped with with a Intel® Xeon® X5650, 32 GB of RAM and

[1] Algorithms using the facultatives models will be skipped if not provided as input.

[2] See footnote number 1.

running Ubuntu 16.04.6 LTS.

Models specifics

In table 7.1 we can observe how many words are present in each word embedding; note that Rcv1 and Reuters have been obtained after continuing training the Wikipedia model and that is the reason why all the numbers are equal (only vectors associated to each word are different).

	Total # of words	**# different words**
Google	4500001500000	3000000
Reuters_27000	2529859941	2306479
RCV1	2529859941	2306479
Blendle	275519659	196167
Wikipedia	2529859941	2306479

Table 7.1: Total number of words contained in each corpus and total number of distinct words in each word embedding.

The RNN model is trained using the same dataset used by Cheng and Lapata (2016) [5], a collection made of 216483 articles; the training time is about 26/28 hours. The Word2Vec model is trained using the implementation provided by Gensim (https://radimrehurek.com/gensim/); the training time to process the RCV1 dataset (over 800000 articles) is about 8 hours. The SIF Word2Vec model is trained using the implementation which can be found at https://github.com/oborchers/Fast_Sentence_Embeddings; the training time to process 200000 articles is about 2 hours (for very large dataset such as RCV1 not every article has been considered due to insufficient memory, in those cases articles have been randomly sampled).

Algorithms specifics

All runtimes were calculated using the DUC-2002 dataset (533 articles). Note that the time provided includes all the steps required to process an article before selecting the most relevant sentences. In tables 7.2 and 7.3 the following notations have been used:

- n: number of sentences, if followed by a subscript it is not referred to the whole text but just to it;
- w: number of words contained in the environment specified by the subscript;
- k: number of cluster obtained with k-means;
- T: number of iterations;
- t: number of topics.

	Runtime	Complexity
MMR	06:05 (1.95 it/s)	O(n^2*((n-1)/2))
TextRank w2v	05:04 (2.30 it/s)	O(T*(n+e))
Kågebäck	06:08 (1.94 it/s)	O(K*(2 $n_{cluster}$))
TextRank TF-IDF	05:05 (2.29 it/s)	O(T*(n+e))
DocEmb	05:45 (2.02 it/s)	O(n * $n_{summary}$)
EmbDist (sentence)	05:39 (2.05 it/s)	O(n*($n_{summary}$ * n_{text})
TextRank sif	05:06 (2.28 it/s)	O(T*(n+e))
RNN	05:39 (2.13 it/s)	O(# hidden units * # weights)
EmbDist (word)	06:00 (1.97 it/s)	O(n*($w_{summary}$ * w_{text})

Table 7.2: Execution time and complexity of word-embedding based algorithms.

	Runtime	Complexity
LB 2010	05:42 (2.03 it/s)	O(n^2((n-1)/2))
LB 2011	06:11 (1.95 it/s)	O(K*(2 ncluster))
MSV	06:21 (2.02 it/s)	O(n * log(n))
TextRank	05:12 (2.25 it/s)	O(T*(n+e))
LexRank	07:03 (1.97 it/s)	O(T*(n+e))
LSA	05:06 (2.28 it/s)	O(t * n)
Lead	05:02 (2.30 it/s)	O(1)

Table 7.3: Execution time and complexity of not word-embedding based algorithms.

Structure of the chapter

This chapter is divided as follows:

- the first section is dedicated to an analysis of which are the most common words contained in the Opinosis and DUC-2002 dataset, a brief introduction to explain further results;
- the second section is devolved to show performances of the various algorithms implemented in the sentence selection phase, underlining the difference between word-embedding based or not approaches;
- the last section contains all the scores associated to every summary produced, having as a goal to show the goodness of all the word embeddings used in this work, trying to understand if there is an advantage in using domain-specific embeddings rather than general-purpose ones.

7.1 Consideration of most frequent words

To validate and provide an ulterior context to the results that will be shown in the next section, in this paragraph an analysis on which are the most frequent words in the Opinosis and DUC-2002 datasets is presented. This easy step has been introduced to verify that there is actually a reason to believe that a domain-specific embedding will be able to address and produce an improved summary with respect to a general

purpose embedding. These are the two lists of the twenty most frequents words and how many times they are repeated through the whole corpora:

- Opinosis: ('staff', 10404), ('friendly', 2295), ('hotel', 2091), ('helpful', 2040), ('room', 1428), ('desk', 1377), ('great', 1326), ('front', 1071), ('nice', 1020), ('service', 867), ('us', 765), ('courteous', 612), ('professional', 612), ('check', 612), ('location', 510), ('like', 459), ('concierge', 459), ('efficient', 459), ('swissotel', 408), ('rooms', 408);
- DUC-2002: ('thatcher', 3731), ('party', 2665), ('next', 1599), ('elections', 1599), ('long', 1599), ('britain', 1066), ('conservative', 1066), ('minister', 1066), ('vote', 1066), ('could', 1066), ('head', 1066), ('someone', 1066), ('place', 1066), ('likely', 1066), ('tax', 1066), ('embattled', 533), ('margaret', 533), ('remains', 533), ('leader', 533), ('country', 533)

These first analysis is confirming the prior belief inherent about the content of these datasets. Opinosis is made of reviews by Amazon users and indeed it is possible to note that most of the words are related to a sentiment about the object just acquired or a description about its use. On the other hand we can see that the DUC-2002 dataset is containing, for the majority, articles related to the politic sphere. Thanks to this little study performed here, it is possible to deduce that the second one will be more suitable for our purpose, because all the documents are related the a specific-domain, the politic one in particular.
As done in the section 3.5.1, here the most similar words for the various models proposed are shown, in order to have a first idea about what to expect from the results and also to have a tool that will help us to understand them.
The first example can be found in table 7.4. In the context of the DUC-2002 Thatcher is referred to Margaret Hilda Thatcher, former Prime Minister of the United Kingdom. In the wikipedia embedding we are finding surnames of other famous people named "Margaret", such as "Tyzack" or "Drubble". Blendle appears to be the one which has the most politic words, while the other three models are presenting a mix of industries names, people called "Margaret" or politics terms.

	Rank 1	**Rank 2**	**Rank 3**	**Rank 4**
Google	draper	drystone	laird	watermill
Reuters_27000	canovan	bondfield	atwood	asquith
RCV1	blair	beckett	conservatives	bondfield
Blendle	margaret	reagan	mitterrand	ronald
Wikipedia	bondfield	drabble	asquith	tyzack

Table 7.4: 4 most similar words to "Thatcher" according to each embedding.

The example reported in table 7.5 is definitely more meaningful with respect to the one introduced in the table 7.4 because party is not only a group of politicians with the same belief but it may also signify a social event. Nonetheless this last meaning is completely absent in all the embeddings introduced here and all of them are returning as result words that are close to the politic lexicon. Even Wikipedia, that is not a domain-specific embeddings, is addressing correctly the word.

	Rank 1	Rank 2	Rank 3	Rank 4
Google	Party	parties	partys	Democratic Party
Reuters_27000	parties	volksunie	partynational	coalition
RCV1	parties	pary	coalition	peronists
Blendle	democratic	parties	centerright	party which
Wikipedia	parties	democrats	coalition	faction

Table 7.5: 5 most similar words to "party" according to each embedding.

As we can see from this results (table 7.6), whenever the target term does not present any possible ambiguity all the different embeddings provide results which are pretty consistent.

	Rank 1	Rank 2	Rank 3	Rank 4
Google	election	parliamentary_elections	electoral	polls
Reuters_27000	election	vote	electoral	constituencies
RCV1	election	polls	vote	referendum
Blendle	election	electoral	parliamentary	primaries
Wikipedia	election	referendum	electoral	referendums

Table 7.6: 4 most similar words to "election" according to each embedding.

When these domain-specific embeddings are tested using a generic words (table 7.7) they perform similarly to Wikipedia or sometimes even worse (like in the case of the Blendle embedding in which the words "staffto" and "staffand" appear).

	Rank 1	Rank 2	Rank 3	Rank 4
Google	staffs	staffers	FRANK_PEEBLES_Citizen	personnel
Reuters_27000	staffs	personnel	administrators	macv
RCV1	employees	personnel	staffs	staffers
Blendle	staffers	staffand	staffs	staffto
Wikipedia	staffs	employees	instructors	officers

Table 7.7: 4 most similar words to "staff" according to each embedding.

In table 7.8 all the embeddings are able to correctly address words inherent to the target term. Surprisingly Wikipedia is among all the one which is returning two words ("wcq" and "wcqg") that are not strictly related to it (this acronym should represent some sort of league presenting "friendly" matches). There two are other aspects worth to be noted: Google return as first result "freindly" that could be the target word but misspelled and Reuters is giving a connotation of "loyalty" to the meaning of term.

	Rank 1	Rank 2	Rank 3	Rank 4
Google	freindly	Friendly	unfriendly	friendlier
Reuters_27000	environmentally	personable	trustful	friendliness
RCV1	unfriendly	cordial	friendlier	constructive
Blendle	friendlier	respectful	unfriendly	cordial
Wikipedia	unfriendly	friendlies	wcq	wcqg

Table 7.8: 4 most similar words to "friendly" according to each embedding.

In table 7.9 Wikipedia is providing four synonyms of the target word and it is probably the best solution among these. Reuters, RCV1 and Blendle, besides the presence of the plural form and "restaurant" which is really often linked to the image of a hotel, are returning names of corporation specialized in management of hotels. This should not represent a surprise: these names (such as "Hyatt" or "Sheraton") are probably reported a number big enough of times in the articles contained in the corpora, to provide a mapping close to the target word.

	Rank 1	Rank 2	Rank 3	Rank 4
Google	hotels	Hotel	motel	boutique_hotel
Reuters_27000	motel	sheraton	kempinski	restaurant
RCV1	hotels	restaurant	hyatt	sofitel
Blendle	hotels	restaurant	ritzcarlton	sheraton
Wikipedia	restaurant	motel	hotels	inn

Table 7.9: 4 most similar words to "hotel" according to the each embedding.

As proven in these previous tables, it is true that domain-specific embeddings are able to address and contextualize with more precision word belonging to a precise lexicon, but the difference between their result and the one reported by the Wikipedia embedding is unexpectedly is very small. This is a witness that, even if the Wikipedia corpora is not domain-specific, it is still able to handle correctly specific terms. This could be explained by the presence of pages that are dealing with really narrow arguments and it appears that this embedding has learned enough information even from those.

7.2 Comparison between sentence selection algorithms

This section will, first, introduce the reader to all the results gathered using the Reuters 27000 dataset, with an analysis on the sentence selection algorithms used. In the second half, due to the fact that the RCV1 dataset is a collection of article coming from the same newspapers, there is a comparison of these two datasets, showing how the number of input samples is affecting the word representation inside the embedding and thus the final score of the summary.

7.2.1 General performances

From figure 7.1 to figure 7.5 all the results gathered using all the algorithm are displayed. Columns in dark blue related to a word-embedding based algorithm, while

light blue to a not word-embedding based algorithm. For these charts and for the following ones, only the recall measure has been reported because summaries have a fixed length. It is possible to observe that word-embedding based algorithms perform similar or even worse to the other. The most interesting cases are provided by the couples LB 2010 with MMR, LB 2011 with Kågebäck and TextRank with all the TextRank word-embedding versions, because the algorithm used is identical with the definition of distance being the only difference. In each one of this scenarios the word-embedding algorithm is returning the worse score besides the TextRank algorithm. TextRank has took advantage of the usage of a word embedding (nonethless the margin is very slight). The highest gap of results is measured in the ROUGE-4 metric but they all scores pretty poorly, indeed ROUGE-4 is a very hard metric in which reaching a good result. Performances have not shown a huge difference among methods using or not word-embedding and they may be a few reasons why this has happened. The first one could be found in using the cosine similarity to measure the distance between two vectors: it is a very simple approach that has been proven to work correctly, but there are finer solutions which could improve the final result; another cause could be imputed to the ROUGE metric itself, because it considers only matching words and not synonyms. This could be a great disadvantage towards algorithms exploiting the word embedding representation because, in this case, the score of similarity is provided by a distance in the vector space and not by a count of shared words, for example. It is very likely that all the summaries produced by these models are achieving a good accuracy without using the same very terms, thus scoring poorly in the context of this metric.

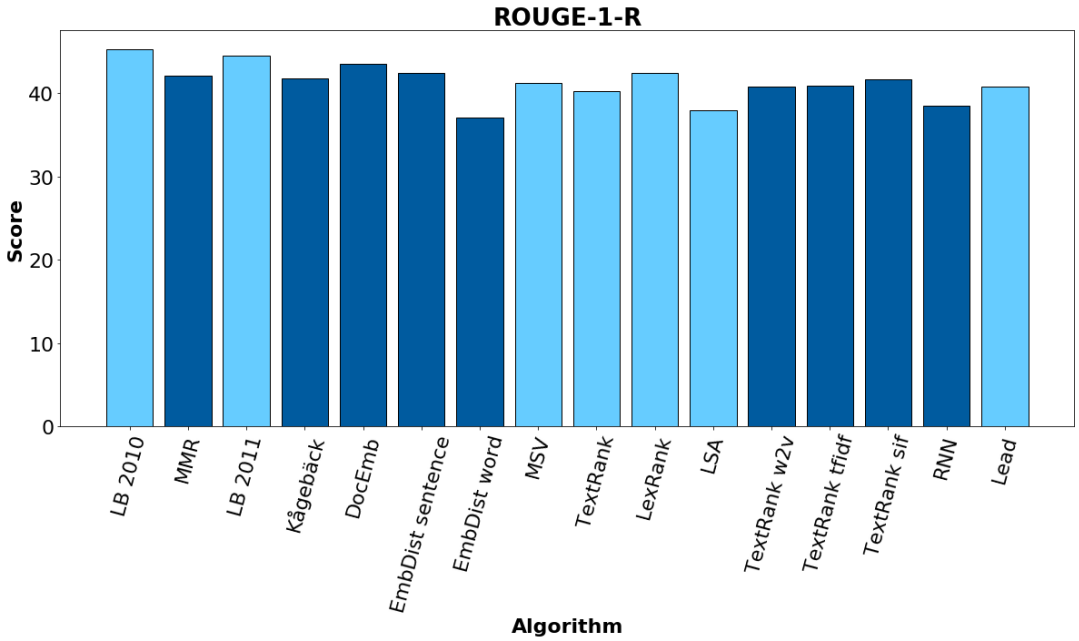

Figure 7.1: Comparison between ROUGE-1 recall scores between all algorithms implemented - DUC-02 test - Reuters-27000 dataset.

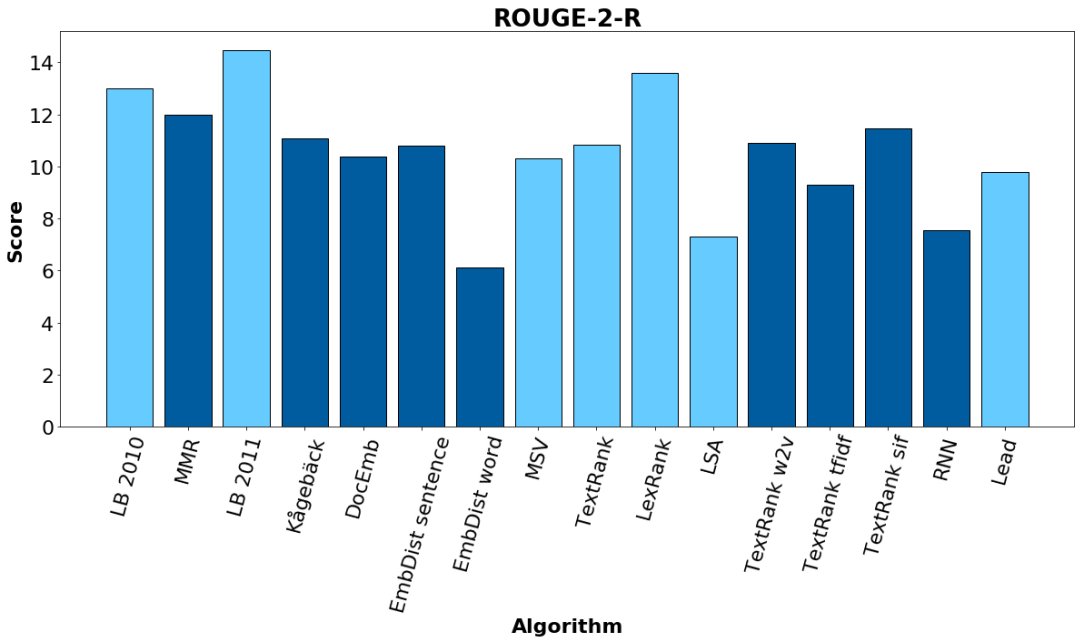

Figure 7.2: Comparison between ROUGE-2 recall scores between all algorithms implemented - DUC-02 test - Reuters-27000 dataset.

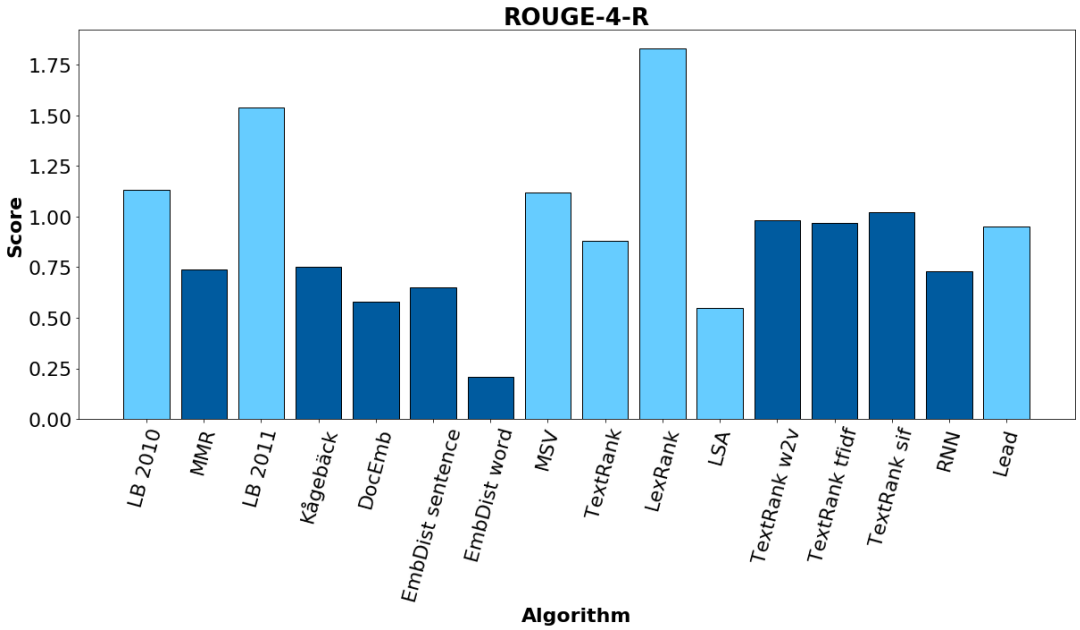

Figure 7.3: Comparison between ROUGE-4 recall scores between all algorithms implemented - DUC-02 test - Reuters-27000 dataset.

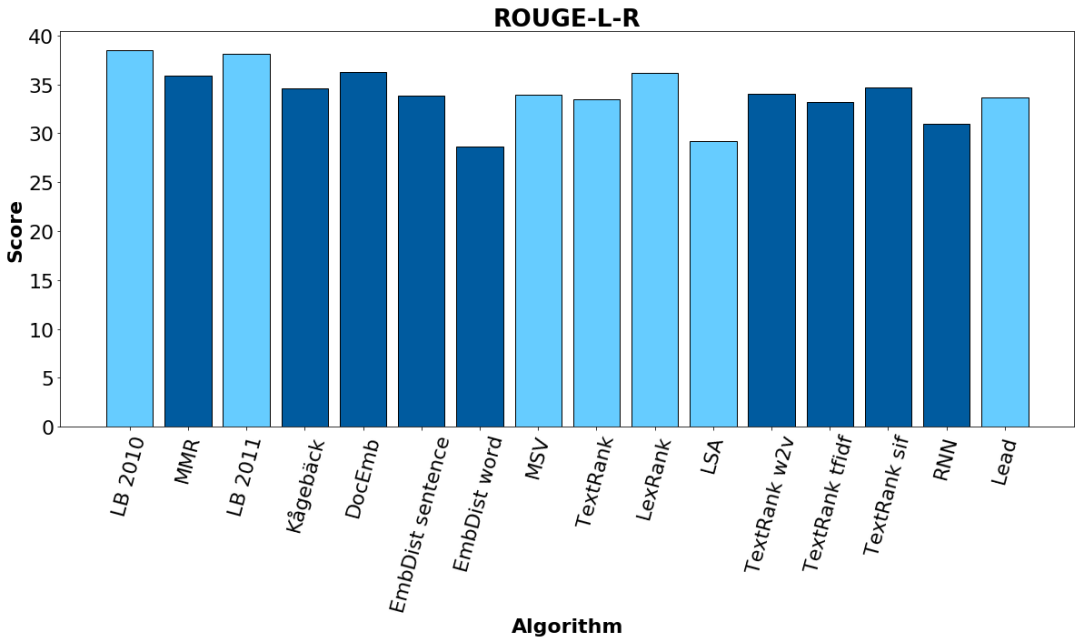

Figure 7.4: Comparison between ROUGE-L recall scores between all algorithms implemented - DUC-02 test - Reuters-27000 dataset.

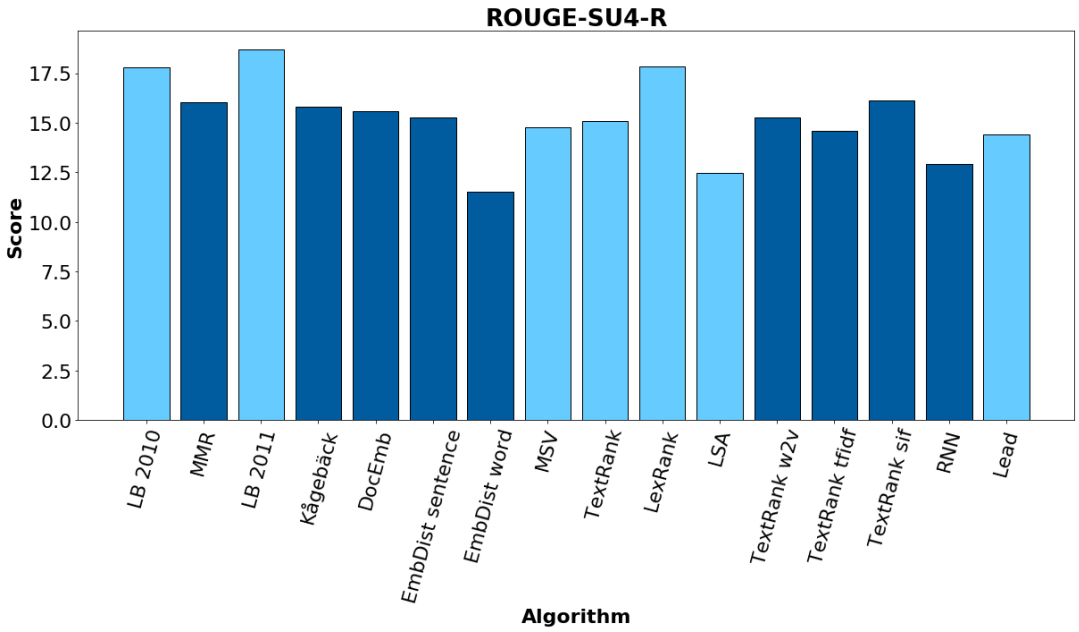

Figure 7.5: Comparison between ROUGE-SU4 recall scores between all algorithms implemented - DUC-02 test - Reuters-27000 dataset.

7.2.2 Reuters embedding results

From figure 7.6 to figure 7.14 there are all the comparison among the RCV1 and the Reuters27000 datasets. These two collections have been chosen to be compared because all the articles have been taken from the same news agency and so they should contain similar documents. The main difference among these two sets is about the number of samples: the first is made of 27000 articles while the other by 800000, so

there is one order of magnitude of difference. As we can see, the gap of performance is probably not enough to justify the usage of a larger dataset. From these figures it appears that, in order to improve these models, other paths should be considered.

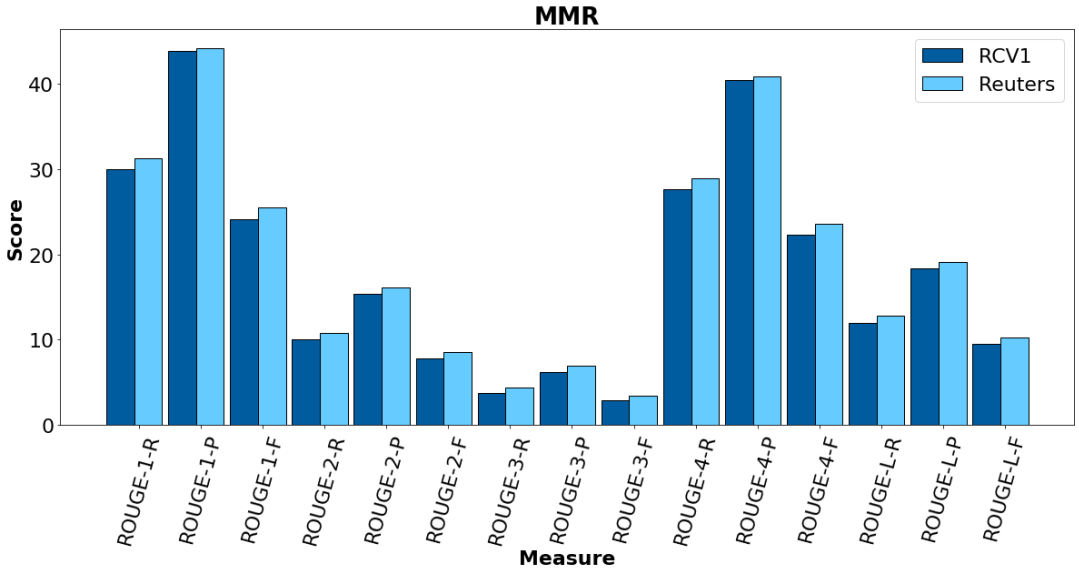

Figure 7.6: Comparison between Reuters2700 and RCV1 models - MMR algorithm.

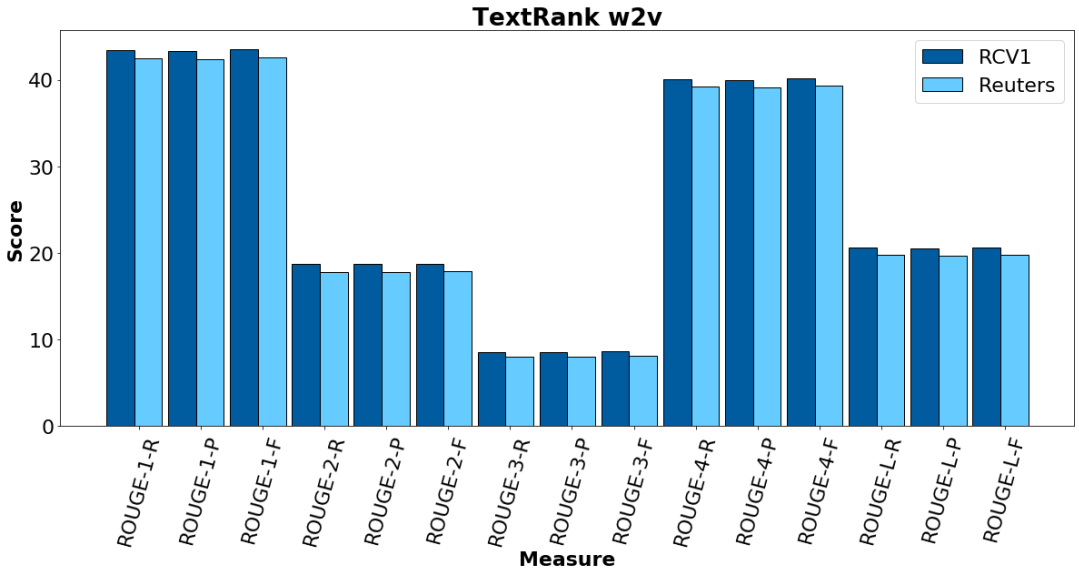

Figure 7.7: Comparison between Reuters2700 and RCV1 models - TextRank w2v algorithm.

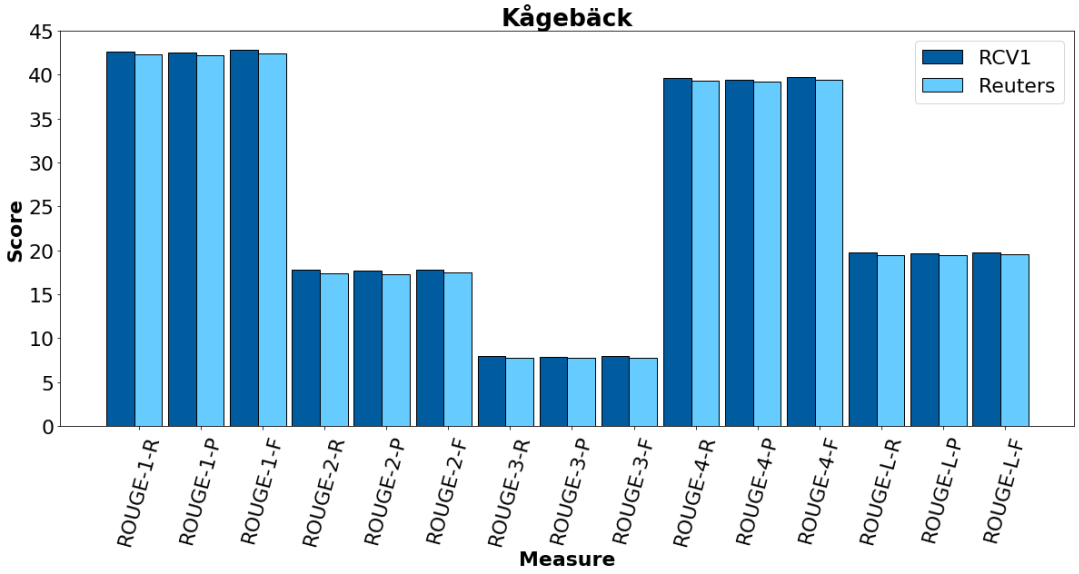

Figure 7.8: Comparison between Reuters2700 and RCV1 models - Kågebäck algorithm.

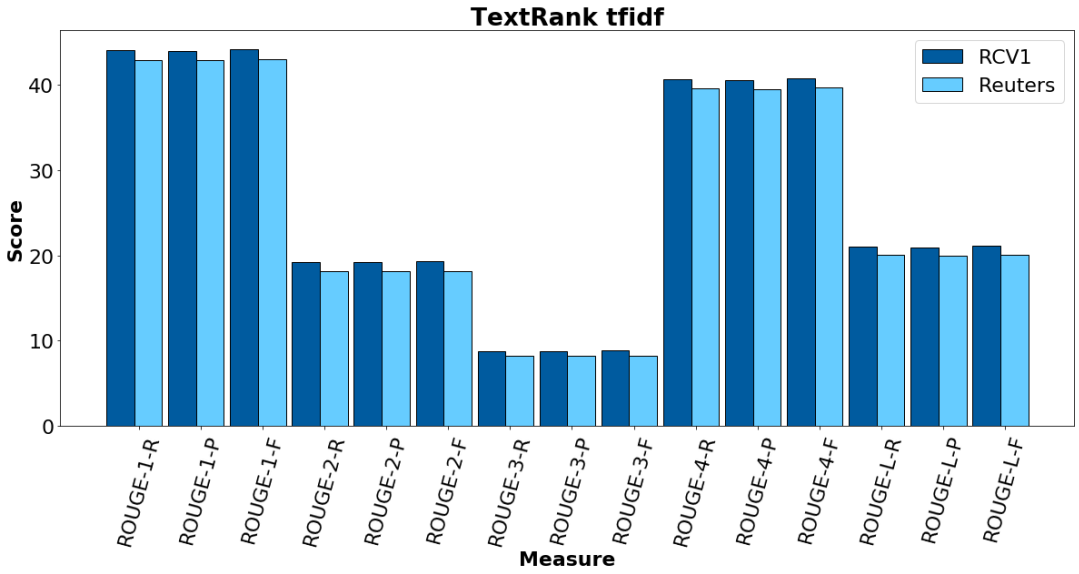

Figure 7.9: Comparison between Reuters2700 and RCV1 models - TextRank TF-IDF algorithm.

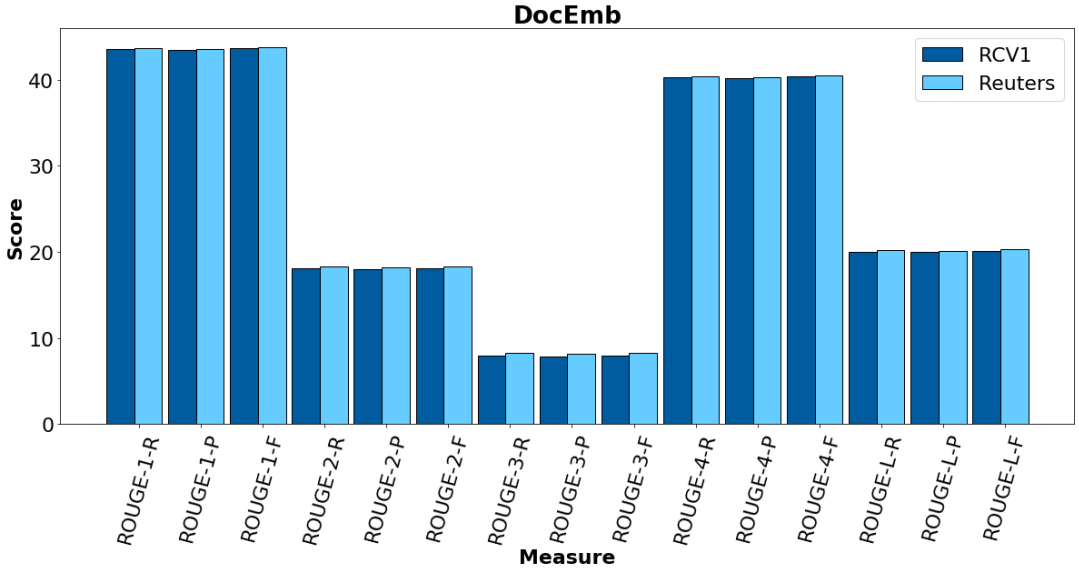

Figure 7.10: Comparison between Reuters2700 and RCV1 models - DocEmb algorithm.

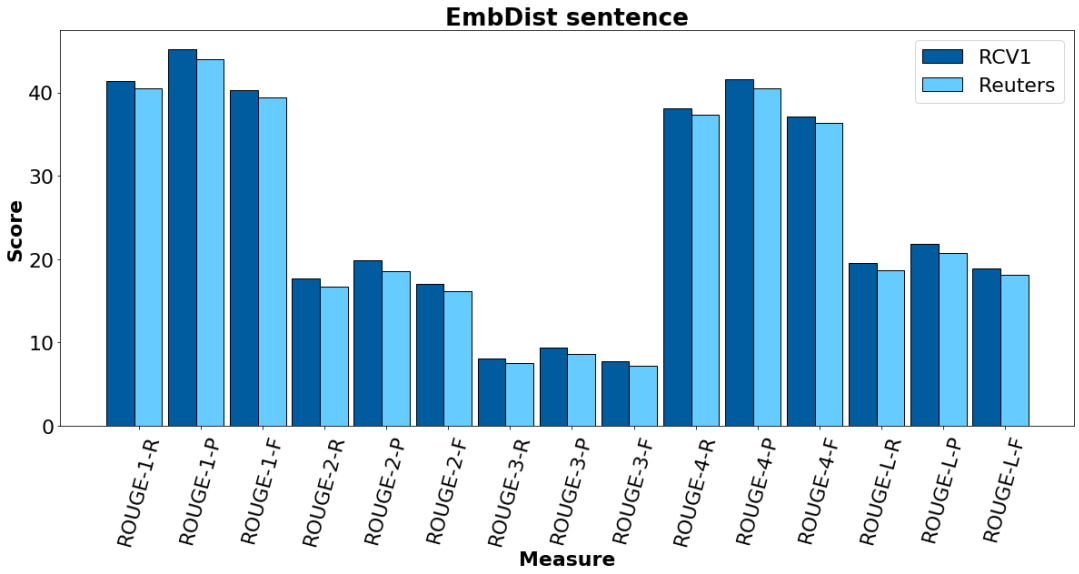

Figure 7.11: Comparison between Reuters2700 and RCV1 models - EmbDist sentence algorithm.

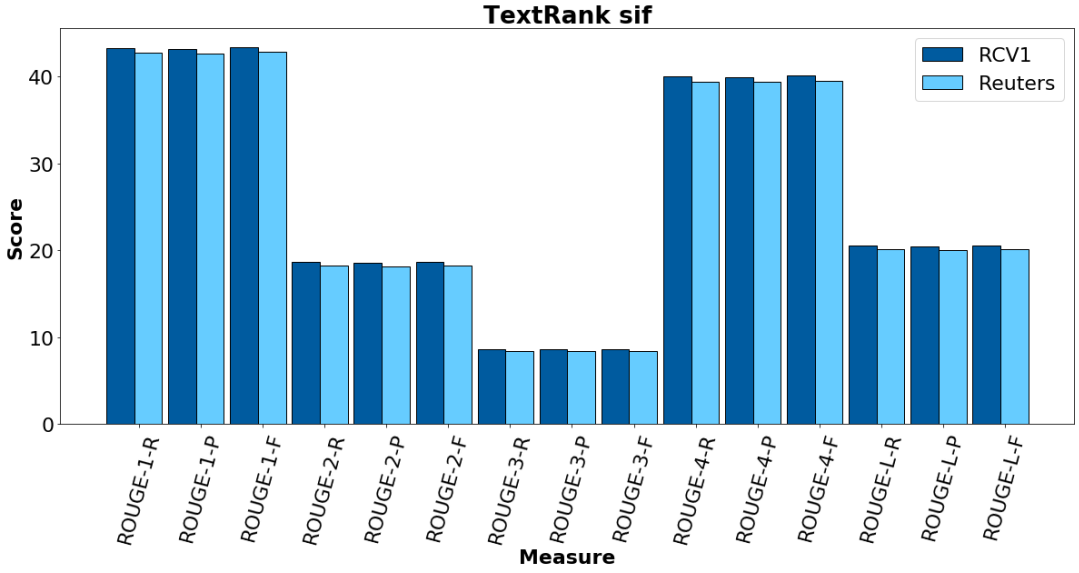

Figure 7.12: Comparison between Reuters2700 and RCV1 models - TextRank sif algorithm.

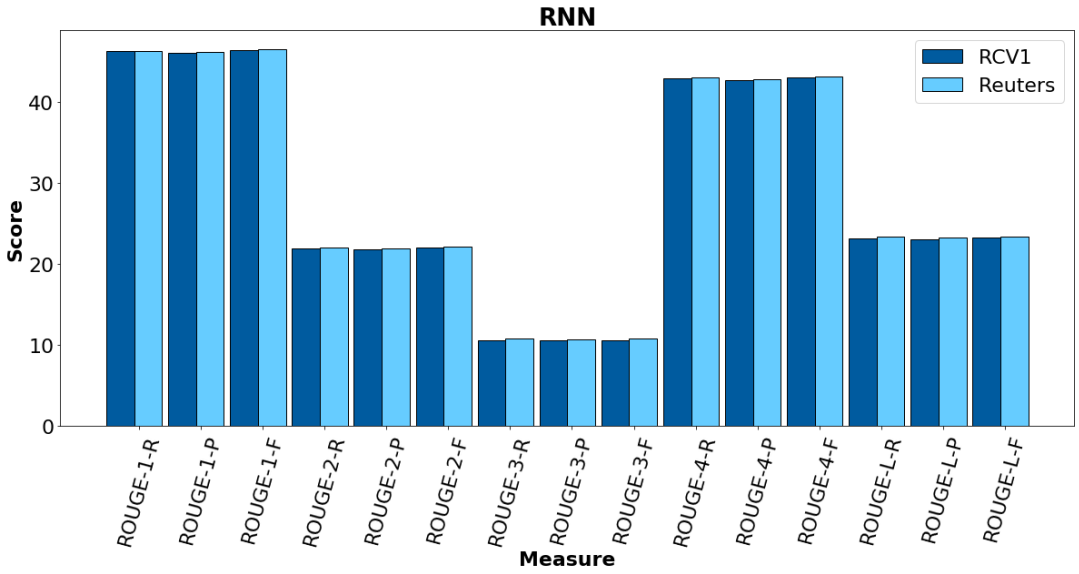

Figure 7.13: Comparison between Reuters2700 and RCV1 models - RNN algorithm.

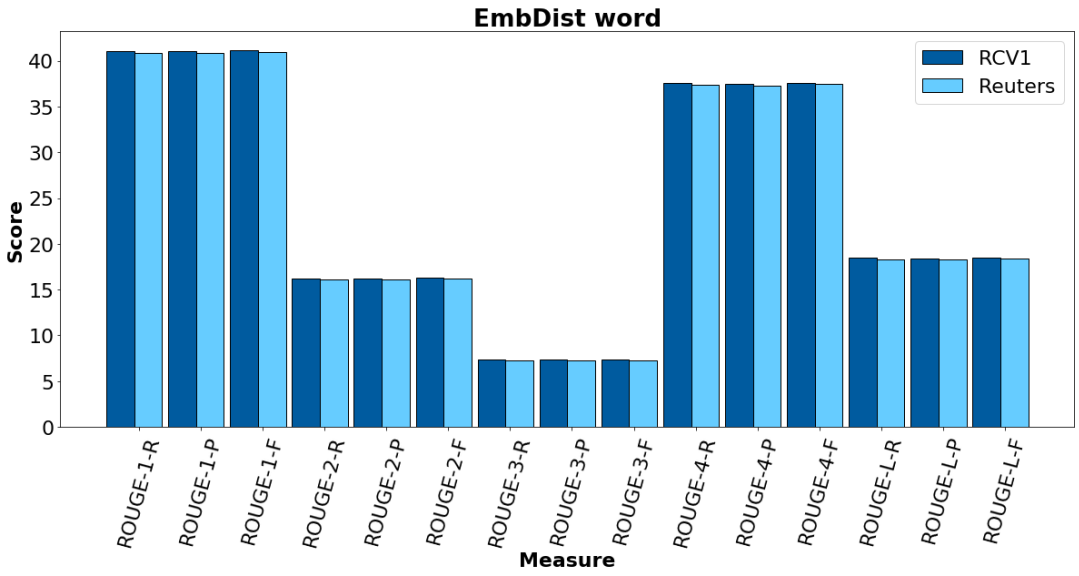

Figure 7.14: Comparison between Reuters2700 and RCV1 models - EmbDist word algorithm.

7.3 Comparison between word embeddings

In these section the main result of the work will be reported. It is a comparison, divided by test dataset, among all the word embeddings implemented. In all the following tables, each number may have a superscript that represents the value obtained with the t-test. Due the fact that each dataset has a rather small sample size, it is important to verify that the average value obtained has or not a statistical relevance. Blendle, Google and Wikipedia results have been tested using the results coming from RCV1 and Reuters 27000. In this work, a result has been considered statistical relevant whenever the *p value* obtained was less than 0.05, using so a 95% interval of confidence, high enough to guarantee the null hypothesis. The following superscripts have been used:

- V : null hypothesis guaranteed using the RCV1 dataset;
- R : null hypothesis guaranteed using the Reuters 27000 dataset;
- * : null hypothesis guaranteed using both RCV1 and Reuters 27000 datasets.

7.3.1 Opinosis dataset

With respect to the Opinosis dataset, the best results are achieved by the RCV1 dataset most of the time but in every other scenario the scores recorded are very close. This result was in a certain way already foreseen, because of the nature of this collection. It is still remarkable to see that domain-specific embeddings are able to handle it with no problems: this is a sign that the transfer learning phase did not only improved the representations, but it also preserved all the vectors associated to "common" words. This is usually a good sign of model trained in the correct manner.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	42.22	11.29^{V}	0.81	35.16	15.95
Reuters_27000	42.11	11.98	0.74	35.88	16.02
RCV1	**42.78**	**12.69**	**1.10**	**36.12**	**16.64**
Blendle	42.23	10.77*	0.69	35.30	15.84
Wikipedia	41.25^{V}	11.65^{V}	0.72	35.09	15.65^{V}

Table 7.10: Recall measure - MMR algorithm - Opinosis dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	40.92	9.12^{V}	0.69	33.50	14.50
Reuters_27000	40.81	10.91	0.98	34.07	15.25
RCV1	**41.06**	**11.29**	**1.27**	**34.93**	**15.86**
Blendle	40.46	9.08*	0.59	32.49^{V}	14.03^{V}
Wikipedia	42.17	10.13	0.67	34.14	15.38

Table 7.11: Recall measure - TextTrank w2v algorithm - Opinosis dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	42.25	11.66	1.28	**35.90**	16.15
Reuters_27000	41.78	11.09	0.75	34.58	15.83
RCV1	**42.48**	**12.19**	**1.47**	35.44	**16.49**
Blendle	42.29	11.41	0.90	35.07	15.71
Wikipedia	41.88	11.10	0.92	35.16	15.75

Table 7.12: Recall measure - Kågebäck algorithm - Opinosis dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	40.66	9.20^{V}	0.51*	31.79^{V}	13.95^{V}
Reuters_27000	40.89	9.31	0.97	33.25	14.60
RCV1	**41.17**	**10.72^{R}**	**1.37**	**34.03**	**15.29**
Blendle	38.92^{V}	8.25^{V}	0.45^{V}	29.94^{V}	13.08^{V}
Wikipedia	41.82^{R}	9.48^{V}	0.59*	33.44^{R}	14.69^{R}

Table 7.13: Recall measure - TextRank TF-IDF algorithm - Opinosis dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	43.05	10.81	**1.03**	36.12	15.32
Reuters_27000	43.51	10.39	0.58	36.31	15.57
RCV1	**43.57**	**11.19**	0.65	**36.68**	**16.10**
Blendle	42.47	10.38	0.92	35.12	14.95^{V}
Wikipedia	42.67	10.53	0.68	35.79	15.68

Table 7.14: Recall measure - Docemb algorithm - Opinosis dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	41.21	9.89	0.64	32.06	14.38
Reuters_27000	**42.48**	**10.81**	0.65	**33.83**	**15.27**
RCV1	41.68	10.21	**0.71**	33.28	14.67
Blendle	41.02^{R}	9.18^{R}	0.53	32.25^{R}	13.85^{R}
Wikipedia	40.93	9.01^{R}	0.40^{V}	32.53	14.32^{R}

Table 7.15: Recall measure - Embdist sentence algorithm - Opinosis dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	41.22	9.00^{*}	0.62	33.62	14.43^{*}
Reuters_27000	41.65	**11.45**	1.02	**34.73**	**16.13**
RCV1	41.11	11.33	**1.23**	34.62	15.89
Blendle	40.83	8.55^{*}	0.51^{R}	32.39^{*}	13.64^{*}
Wikipedia	**42.07**	9.97^{*}	0.94	34.49	15.25

Table 7.16: Recall measure - TextRank sif algorithm - Opinosis dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	37.33	6.91	0.66	30.09	12.17
Reuters_27000	38.55	7.56	0.73	31.01	12.90
RCV1	38.43	7.04	0.59	30.60	12.56
Blendle	38.13	6.36	0.56	30.94	12.21
Wikipedia	**39.31**	$\mathbf{8.21^{V}}$	$\mathbf{0.85^{V}}$	$\mathbf{2.66^{V}}$	**13.08**

Table 7.17: Recall measure - RNN algorithm - Opinosis dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	37.31	5.87^{V}	0.19	27.95	11.37
Reuters_27000	37.05	6.13	0.21	28.61	11.53
RCV1	37.00	**7.12**	**0.46**	**29.01**	**11.90**
Blendle	**37.47**	6.40	0.18	28.66	11.64
Wikipedia	36.63	6.41	0.60	28.30	11.60

Table 7.18: Recall measure - Embdist word - Opinosis dataset.

7.3.2 DUC-02 dataset

In this section all the results achieved while using the DUC-02 dataset are reported. These are the most surprising results: this collection is made of news documents but there is basically no difference in results while deploying a domain-specific or a general-purpose embedding. This behaviour could be explained by a not well trained domain-specific word embedding or by the fact that this dataset is not suited to be considered in this kind of experiments. To answer this question, the study introduced in the section 7.1 has been done, together with considering another collection of articles to understand if the outcome is the same while analyzing another documents.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	20.96*	6.62*	2.45*	19.36*	8.06*
Reuters_27000	**25.53**	**8.59**	**3.44**	**23.61**	**10.30**
RCV1	24.13^{R}	7.85^{R}	2.87^{R}	22.27^{R}	9.48^{R}
Blendle	19.30*	6.09*	2.27*	17.87*	7.40*
Wikipedia	23.67^{R}	7.29*	2.60^{R}	21.82^{R}	9.06^{R}

Table 7.19: Recall measure - MMR algorithm - DUC-02 dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	42.01^{V}	17.21^{V}	7.79^{V}	38.50*	19.38^{V}
Reuters_27000	42.59	17.83	8.07	39.30	19.82
RCV1	**43.53**	**18.77**	**8.59**	**40.17**	**20.64**
Blendle	42.48^{V}	17.48^{V}	7.88^{V}	38.90^{V}	19.62^{V}
Wikipedia	42.39^{V}	17.43^{V}	7.98^{V}	38.92^{V}	19.74^{V}

Table 7.20: Recall measure - Textrank w2v algorithm - DUC-02 dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	42.59	17.66	7.87	39.59	19.60
Reuters_27000	42.42	17.44	7.82	39.42	19.57
RCV1	**42.83**	**17.83**	**7.97**	**39.72**	**19.81**
Blendle	41.97^{V}	17.11^{V}	7.49	39.03^{V}	19.07^{V}
Wikipedia	42.22^{V}	17.18^{V}	7.53	39.17	19.29

Table 7.21: Recall measure - Kågebäck algorithm - DUC-02 dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	43.95^{R}	19.27^{R}	$\mathbf{8.97^{R}}$	40.47^{R}	21.05^{R}
Reuters_27000	43.06	18.20	8.23	39.72	20.09
RCV1	$\mathbf{44.22^{R}}$	19.31^{R}	8.84^{R}	$\mathbf{40.78^{R}}$	$\mathbf{21.12^{R}}$
Blendle	43.96^{R}	$\mathbf{19.37^{R}}$	8.93^{R}	40.43^{R}	21.04^{R}
Wikipedia	43.80^{R}	18.93^{R}	8.78^{R}	40.48^{R}	20.80^{R}

Table 7.22: Recall measure - Textrank TF-IDF algorithm - DUC-02 dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	43.65	18.25	7.99	40.37	20.18
Reuters_27000	**43.80**	**18.33**	**8.24**	**40.55**	**20.29**
RCV1	43.72	18.12	7.93	40.40^{R}	20.08
Blendle	43.20	17.73	7.87	39.86^{R}	19.77
Wikipedia	43.44	17.74	7.66^{R}	40.16	19.79

Table 7.23: Recall measure - Docemb algorithm - DUC-02 dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	39.69	16.80^{R}	7.54	36.43	18.54
Reuters_27000	39.45	16.13	7.24	36.33	18.13
RCV1	40.24	17.01	**7.72**	37.06	18.88
Blendle	38.78^{V}	16.57^{R}	7.52^{R}	35.57^{V}	18.17^{V}
Wikipedia	$\mathbf{40.93^{R}}$	$\mathbf{17.11^{R}}$	7.67	$\mathbf{37.64^{R}}$	$\mathbf{19.03^{R}}$

Table 7.24: Recall measure - Embdist (sentence) algorithm - DUC-02 dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	41.87*	17.05*	7.66*	38.36*	19.23*
Reuters_27000	42.85	18.28	8.43	39.55	20.16
RCV1	**43.41**	**18.69**	**8.62**	**40.14**	**20.59**
Blendle	43.36	18.52	8.46	39.83	20.37
Wikipedia	42.59^{V}	17.77^{V}	8.02^{V}	39.15^{V}	19.71^{V}

Table 7.25: Recall measure - TextRank sif algorithm - DUC-02 dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	46.42	22.14	10.74	43.09	23.31
Reuters_27000	46.58	22.15	10.83	43.25	23.46
RCV1	46.50	22.02	10.64	43.11	23.30
Blendle	46.70	22.33	10.84	43.31	23.52
Wikipedia	$\mathbf{46.95^{*}}$	$\mathbf{22.63^{*}}$	$\mathbf{11.06^{*}}$	$\mathbf{43.63^{*}}$	$\mathbf{23.82^{*}}$

Table 7.26: Recall measure - RNN algorithm - DUC-02 dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	41.72*	16.81*	**7.62**	38.18*	18.91*
Reuters_27000	40.95	16.18	7.30	37.45	18.35
RCV1	41.15^{R}	16.26^{R}	7.36	37.60^{R}	18.48^{R}
Blendle	$\mathbf{41.79^{*}}$	$\mathbf{16.83^{*}}$	7.53	$\mathbf{38.21^{*}}$	$\mathbf{18.94^{*}}$
Wikipedia	41.01	16.20	7.31	37.51	18.40

Table 7.27: Recall measure - Embdist word algorithm - DUC-02 dataset.

7.3.3 BBC dataset

The decision to use also this dataset as benchmark for this series of test has been taken after the disappointing results gathered in the previous section 7.3.2. This collection of articles immediately looked more promising for our purpose: not only it contains a larger number of documents but they belong to different topics, while DUC-02 is mainly made of politics one. This composition is rather similar with respect to the miscellaneous articles coming from Reuters-27000: also in this latter group each document has been previously labeled according to its topic and they share every domain

(besides health and science which are not contained in the BBC collection). The experiment performed here is the same in relationship with the ones done previously and from table 7.28 to table 7.36 it is possible to read each result.
The outcome for this dataset has been a success for multiple reasons:

- Wikipedia word embedding never returns the best score in each algorithm and metric, besides while using RNN, in which anyway every model is performing very similar (there is a maximum difference of 0.5); this means that all the contextualized models are able to deal better with the words and sentences contained in the benchmark;
- Reuters_27000 and RCV1 most of the time are the best model to solve this problem, this could be explained thanks to two main justifications: the first one, as said before, is about sharing the same topics and the second one is that all these models have been trained using a corpus made of articles coming from newspapers (so in theory they should share a common similar lexicon);
- the best results are coming from the runs of the experiment while using TextRank (in all its variances) and these scores are the best one that have been obtained throughout all the work; RNN and the two versions of EmbDist are the worst algorithms instead;
- while using RNN or EmbDist word all the scores are pretty similar, this could signify that these methods are not too proper to be used for domain-specific implementations.

To summarize, this dataset has shown to be very suited to make tests related to multi-document summarization tasks, driven by embeddings contextualized with newspaper articles.

	ROU-1-R	**ROU-2-R**	**ROU-4-R**	**ROU-L-R**	**ROU-SU4-R**
Google	19.33*	10.62*	8.27*	16.53*	11.20*
Reuters_27000	**22.64**	**12.62**	**9.87**	**19.11**	**13.30**
RCV1	22.00^{R}	12.31	9.68	18.70	12.97
Blendle	17.47*	9.16*	6.94*	14.92*	9.75*
Wikipedia	20.97*	11.39*	8.81*	17.70*	12.06*

Table 7.28: Recall measure - MMR algorithm - BBC dataset.

	ROU-1-R	**ROU-2-R**	**ROU-4-R**	**ROU-L-R**	**ROU-SU4-R**
Google	61.23*	48.99*	43.95*	54.33*	48.61*
Reuters_27000	**63.39**	51.88	46.69	56.95	51.16
RCV1	63.31	**52.15**	**46.91**	**57.22**	**51.42**
Blendle	60.99*	49.01*	43.96*	54.41*	48.66*
Wikipedia	58.50*	45.09*	40.04*	51.21*	45.11*

Table 7.29: Recall measure - textrank w2v algorithm - BBC dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	54.70^{R}	39.44^{R}	33.91^{R}	47.35	39.60^{R}
Reuters_27000	**55.41**	**40.24**	**34.85**	**47.94**	**40.48**
RCV1	55.20	39.90	34.41	47.74	40.12
Blendle	53.72*	38.11*	32.58*	46.30*	38.34*
Wikipedia	54.69^{R}	39.35^{R}	33.96^{R}	47.22^{R}	39.62^{R}

Table 7.30: Recall measure - Kågebäck algorithm - BBC dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	**62.34***	**50.46***	**45.37***	**55.73***	**49.93***
Reuters_27000	60.96	48.46	43.26	54.19	48.04
RCV1	61.40	49.00	43.79	54.70	48.52
Blendle	62.14*	50.18*	45.09*	55.53*	49.68*
Wikipedia	61.35	48.89	43.70	54.59	48.44

Table 7.31: Recall measure - textrank TF-IDF algorithm - BBC dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	55.82*	40.13*	34.65*	48.19*	40.46*
Reuters_27000	56.73	41.46	35.87	49.19	41.63
RCV1	**57.59^{R}**	**42.62^{R}**	**37.07^{R}**	**50.13^{R}**	**42.69^{R}**
Blendle	55.39*	39.43*	33.98*	47.65*	39.88*
Wikipedia	57.28^{R}	42.02	36.45	49.71	42.18

Table 7.32: Recall measure - Docemb algorithm - BBC dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	50.39	39.48	35.12^{R}	44.65	39.22
Reuters_27000	49.86	38.62	34.17	44.05	38.41
RCV1	**51.27^{R}**	**39.78^{R}**	**35.22^{R}**	**45.28^{R}**	**39.54^{R}**
Blendle	48.22*	37.88^{V}	33.75^{V}	42.74*	37.60*
Wikipedia	50.90^{R}	39.00	34.42	44.69	38.86

Table 7.33: Recall measure - Embdist sentence algorithm - BBC dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	61.30*	49.08*	44.03*	54.40*	48.69*
Reuters_27000	**63.83**	**52.85**	**47.57**	**57.69**	**52.02**
RCV1	63.09^{R}	51.96^{R}	46.76^{R}	57.02^{R}	51.24^{R}
Blendle	63.05^{R}	51.89^{R}	46.77^{R}	56.97^{R}	51.22^{R}
Wikipedia	62.68^{R}	51.46^{R}	46.29^{R}	56.62^{R}	50.76^{R}

Table 7.34: Recall measure - textrank sif algorithm - BBC dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	57.47	43.51	38.09	50.27	43.38
Reuters_27000	57.41	43.47	38.03	50.22	43.33
RCV1	57.57	43.57	38.08	50.29	43.40
Blendle	57.17	43.08	37.60	49.90	42.97
Wikipedia	**57.59**	**43.70**	**38.15**	**50.38**	**43.47**

Table 7.35: Recall measure - RNN algorithm - BBC dataset.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Google	**51.99***	**36.12***	**31.38***	**44.31***	**36.94***
Reuters_27000	51.28	35.21	30.56	43.71	36.16
RCV1	51.33	35.41	30.69	43.80	36.26
Blendle	51.92*	36.07*	31.28^{R}	44.37*	36.88*
Wikipedia	51.89*	36.09*	31.43*	44.34*	36.94*

Table 7.36: Recall measure - Embdist word algorithm - BBC dataset.

7.3.4 Further considerations

While evaluating all the scores associated to the t-test, the results obtained for the BBC set were definitely stronger with respect to the two other datasets (more p-values under the threshold and values lower than previous cases). This means that, besides achieving an overall better series of summaries, the ROUGE scores obtained are also more statistical relevant. This is an ulterior proof that the BBC dataset is more suitable to be implemented in this kind of tasks. The last consideration about all the results obtained trough all the tests performed is introduced by the table 7.37. It is possible to notice that, for each different metric of ROUGE, the best results are obtained while using the BBC benchmark. This is not surprising at all, because this dataset is the closest one to the news world and, thanks to this, the four domain-specific models are able to produce summaries with a very good precision. The strong difference of scores in both ROUGE-4-R and ROUGE-SU4-R is the main responsible for this consideration: these two metrics are the hardest one to obtain a large number, because they are considering multiple words at the time. Reaching a recall of 35.19% and 40.02% is a remarkable result.

	ROU-1-R	ROU-2-R	ROU-4-R	ROU-L-R	ROU-SU4-R
Opinosis	40.82	9.64	0.75	33.26	14.51
DUC-2002	40.63	17.00	7.69	37.47	18.85
BBC	**53.05**	**40.02**	**35.19**	**46.51**	**40.02**

Table 7.37: Average of every model according separately to each metric proposed.

Chapter 8

Conclusion & Future Directions

To conclude and summarize all the results gathered in this thesis there is the need to split this speech in two parts.
In the first half of the work a new intrinsic evaluation method has been introduced, based on producing a summary focused on a single topic to understand the goodness of a domain-specific word embedding. The outcome ended up being not very good, because the Wikipedia embedding showed itself to be the best one to correctly address this kind of job. This approach could be improved in the future, creating a hand-made glossary, written by people who have a high degree of knowledge about the argument: with this little change there is the hope to obtain more detailed definitions, probably more suited for this evaluation task. There is also the possibility to introduce a weight in this approach to reward mostly embeddings addressing correctly the specific lexicon: a definition has to include also words of common use and it may be that a general-purpose embedding is able to achieve a great score just because of them. Whereas this should be still not enough, there is nonetheless the need to have a benchmark that can be used to test domain-specific embeddings: the current state-of-art is made of datasets to realize if the vector space created is able to correctly identify semantic relationships, while for this type of problems the most important aspect is to recognize with more ease words belonging to a specific lexicon, giving more relevance to the meaning strictly correlated to it. There are many ways to create a new benchmark, for example synonyms and antonyms couples that do not include terms belonging to the common language or clusters of words, each one representing a sub-argument of the topic (for example, given two groups of words, related to hardware or software terms, is it true that an embedding trained using computer science documents is able to distinguish them?). Another aspect to deepen is the "coverage" of the topic: choosing an argument that is too wide (such as Science) could be very close to not specialize at all the embedding because of the number of different subjects involved. Choosing a more narrow argument could be the correct implementation (for example natural disaster) but at the same time there is the need to provide new training datasets, large enough to obtain a new good representation.
In the last half of the thesis the main concern was about understanding the performance of various models in solving document summarization tasks. The first two datasets (Opinosis and DUC-2002) resulted in a failure but with a major difference: Opinosis was expected to return poor results because of its nature, DUC-2002 was not. This unexpected behaviour could be a symptom of the absence of suited datasets to

test domain-specific document summarization problems. Once again, there is the need to create new datasets to be used specifically in the context of multi-document domain-specific summarization task. Indeed, the only promising results achieved, was the one related to the BBC dataset, which contained a larger choice of articles, spacing among all the topics touched by the Reuters27000 group of articles. To be discussed is also the implementation of the ROUGE metric because it rewards only identical words, while all the models that have been introduced in this work are using a vector representation to compute a distance. It could be useful to compare the hand-written summary and the one obtained at the end of the procedure with metrics that are taking into account also their word-embedding representation.

Bibliography

[1] S. Arora, Y. Liang, and T. Ma. A simple but tough-to-beat baseline for sentence embeddings. 1 2019. 5th International Conference on Learning Representations, ICLR 2017 ; Conference date: 24-04-2017 Through 26-04-2017.

[2] E. Baralis, L. Cagliero, S. Jabeen, and A. Fiori. Multi-document summarization exploiting frequent itemsets. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, page 782–786, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450308571. doi: 10.1145/2245276.2245427.

[3] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. doi: 10.1162/tacl_a_00051.

[4] J. Camacho-Collados and M. T. Pilevar. From word to sense embeddings: A survey on vector representations of meaning. *Journal of Artificial Intelligence Research*, 63:743–788, 12 2018. doi: 10.1613/jair.1.11259.

[5] J. Cheng and M. Lapata. Neural summarization by extracting sentences and words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 484–494, Berlin, Germany, Aug. 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1046.

[6] L. de Haas. Extractive summarization using sentence embeddings, automatic summarization of news articles at blendle. Master's thesis, Utrecht University, 2017.

[7] G. Erkan and D. R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, 22:457–479, Dec 2004. ISSN 1076-9757. doi: 10.1613/jair.1523.

[8] M. Faruqui, J. Dodge, S. K. Jauhar, C. Dyer, E. Hovy, and N. A. Smith. Retrofitting word vectors to semantic lexicons, 2014.

[9] D. Greene and P. Cunningham. Practical solutions to the problem of diagonal dominance in kernel document clustering. In *Proc. 23rd International Conference on Machine learning (ICML'06)*, pages 377–384. ACM Press, 2006.

[10] F. Hill, K. Cho, and A. Korhonen. Learning distributed representations of sentences from unlabelled data. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1367–1377, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1162.

[11] Hynek, J. Jezek, and K. Jezek. Practical approach to automatic text summarization. *ELPUB2003. From information to knowledge: Proceedings of the 7th ICCC/IFIP International Conference on Electronic Publishing held at the Universidade do Minho, Portugal 25-28 June 2003/Edited by: Sely Maria de Souza Costa, Jo?o ?lvaro Carvalho, Ana Alice Baptista, Ana Cristina Santos Moreira. Universidade do Minho, 2003.*, 01 2003.

[12] M. Kågebäck, O. Mogren, N. Tahmasebi, and D. Dubhashi. Extractive summarization using continuous vector space models. In *Proceedings of the 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC)*, pages 31–39, Gothenburg, Sweden, Apr. 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-1504.

[13] P. Kameswara Sarma, Y. Liang, and B. Sethares. Domain adapted word embeddings for improved sentiment classification. In *Proceedings of the Workshop on Deep Learning Approaches for Low-Resource NLP*, pages 51–59, Melbourne, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-3407.

[14] R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler. Skip-thought vectors, 2015.

[15] H. Kobayashi, M. Noguchi, and T. Yatsuka. Summarization based on embedding distributions. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1984–1989, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1232.

[16] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger. From word embeddings to document distances. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 957–966. JMLR.org, 2015.

[17] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents, 2014.

[18] H. Lin and J. Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 912–920, Los Angeles, California, June 2010. Association for Computational Linguistics.

[19] H. Lin and J. Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, page 510–520, USA, 2011. Association for Computational Linguistics. ISBN 9781932432879.

[20] B. McCann, J. Bradbury, C. Xiong, and R. Socher. Learned in translation: Contextualized word vectors, 2017.

[21] R. Mihalcea and P. Tarau. TextRank: Bringing order into text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 404–411, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[22] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space, 2013.

[23] T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

[24] J. Mitchell and M. Lapata. Composition in distributional models of semantics. *Cognitive science*, 34:1388–429, 11 2010. doi: 10.1111/j.1551-6709.2010.01106.x.

[25] N. Nalini and S. Palanivel. Music emotion recognition: The combined evidence of mfcc and residual phase. *Egyptian Informatics Journal*, 17, 09 2015. doi: 10.1016/j.eij.2015.05.004.

[26] F. Nooralahzadeh, L. Øvrelid, and J. T. Lønning. Evaluation of domain-specific word embeddings using knowledge resources. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA).

[27] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[28] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162.

[29] D. R. Radev, H. Jing, and M. Budzikowska. Centroid-based summarization of multiple documents: sentence extraction, utility-based evaluation, and user studies. In *NAACL-ANLP 2000 Workshop: Automatic Summarization*, 2000.

[30] R. Socher, E. H. Huang, J. Pennington, A. Y. Ng, and C. D. Manning. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 801–809, Red Hook, NY, USA, 2011. Curran Associates Inc. ISBN 9781618395993.

[31] J. Steinberger and K. Jezek. Using latent semantic analysis in text summarization and summary evaluation. 01 2004.

[32] K. Thakkar, R. Dharaskar, and M. Chandak. Graph-based algorithms for text summarization. *Emerging Trends in Engineering Technology, International Conference on*, 0:516–519, 11 2010. doi: 10.1109/ICETET.2010.104.

[33] D. Yogatama, F. Liu, and N. A. Smith. Extractive summarization by maximizing semantic volume. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1961–1966, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1228.