Methodological approaches for Text Summarization

Riccardo Cervero, 794126*

Abstract
Customer reviews on E-commerce websites can often be long and too descriptive. Reading prolix comments can be annoying to the other users and analyzing these texts manually is extremely time-consuming. Thus, technical specifications can be applied to generate a summary for any review.
In the context of Natural Language Processing there are various types of Text Summarization methodologies. The objective of this project is to compare some of them, in order to test their performance in terms of quality of the result. Therefore, 5 algorithms specialized in Extractive Summarization will be implemented, based on the extraction of topics and features, on the construction of a graph or on Deep Learning methods. In addition, a complex architecture suitable for Abstractive Summarization, namely capable of recreating a summary instead of simply selecting the most important parts of the text, will be presented and used. In the last part, all these models will be manually evaluated , to obtain a clearer and more direct analysis of their characteristics.

Keywords
Abstractive Summarization — Extractive Summarization

* *Universita' degli Studi di Milano Bicocca, CdLM in Data Science*

Contents

1. Dataset Exploration

The texts to be summarized hails from a portion of the "Fine Food Reviews" dataset, recording 100,000 English natural-language-written reviews from more than 70,000 Amazon users. The original data span from October 1999 up to October 2012, including, besides plain text reviews, product and user information, ratings and, above all, a very brew reference summary offered by database providers which will constitute an evaluation benchmark and the ground truth during the neural network training phase.
As seen from the plot, ratings distribution is heavily unbalanced towards the most positive score.

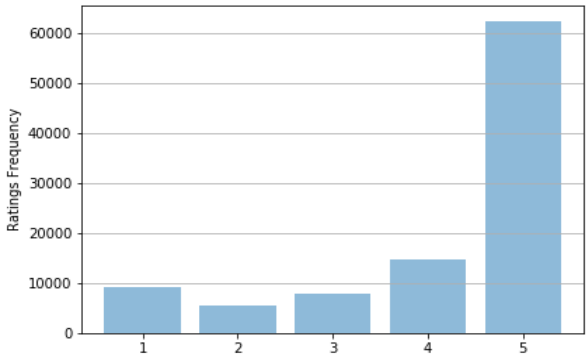


Figure 1. Ratings distribution

As far as reviews, the average number of sentences is 5, while about one sentence on average is used to summarize the content. The number of sentences in the original document can affect extractive results of summarization, and its distri-

bution shows a slight concentration around the mean[1], with a very long and thin tail.

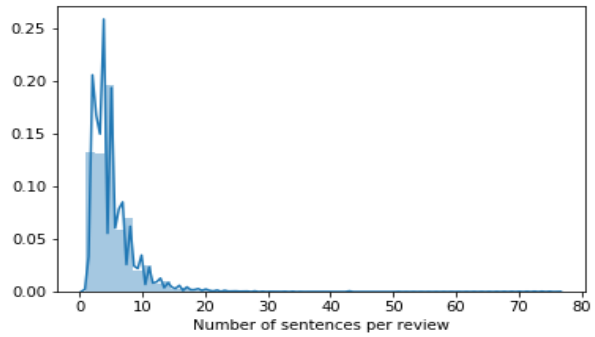


Figure 2. Distribution of number of sentence per review

Nevertheless, even though this characteristic of homogeneity could help avoid bias during generation and comparison of several summaries, it is counterbalanced by a greater variability[2] in the number of words per review.

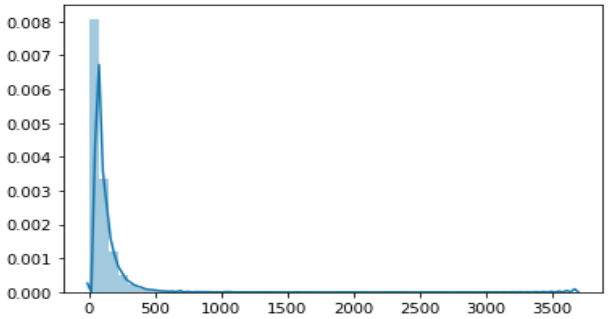


Figure 3. Distribution of number of word per review

Finally, it is possible to get a first insight on vocabulary composition by representing the corpus *wordcloud*.

Figure 4. Corpus WordCloud

[1]Reviews composed by a number of sentences between 3 and 7 are about 64% of the total.

[2]While coefficient of variance for sentences count is about 74%, the one related to word count is 102%

2. Text Pre-processing

Review is a particular kind of text, most of time composed by non-professional writer, which thus tends to resemble spoken language, because includes abbreviations, slang and informal phrases, errors in punctuation and misspellings, etc. These documents appear as a mixture of heterogeneous forms of text, even presenting different lexical and syntactic structures within the same sentence. For this reason, pre-processing operations are compulsory to convert the corpus into standard form and consequently reduce potential distortions.

So, after having tokenized[3] the document, as the normalization phase, it is necessary to convert all informal expressions that occur in form of contractions[4] and put all the tokens to lower case. The next step consists in removing the "noisy" parts of the text: genitives, punctuation, symbols, special characters, numbers and multiple white spaces. Finally, stop words are eliminated, because this very frequent tokens are useless and could distort the co-occurrence analysis during the stage of text representation. No stemming or lemmatization operations are performed, as inflected and derived tokens provide fundamental information that both simplest algorithms and neural networks cannot afford to lose during elaboration of a summary containing the most relevant concepts.

In each following methodology, the aforementioned operations will be applied separately to each sentence, extracted from the text by means of a *sentence tokenizer*, so as to return a list of *cleaned* sentences. The *sentence tokenizer* uses an unsupervised algorithm to build a model for detecting sentence boundaries[5], which has proven to be an effective approach for many European languages.

3. Extractive Summarization

Extractive summarization aims to recognize most important sections of a text and consequently generate a subset of the original sentences. Between the two existing approaches, this is certainly the most superficial, as it returns a simple verbatim reduction of the original document.

3.1 Topic representation methods

These methods generate an intermediate representation by capturing discussed topics and then score each sentence according to its importance. This class includes techniques that vary mainly on the basis of the intermediate representation that they are able to extract. One of them is the matrix-factorization-based Latent Semantic Analysis.

3.1.1 Latent Semantic Analysis

Latent Semantic Analysis (*LSA*) [4] is an unsupervised technique extracting an implicit representation of semantic structures within text. It identifies several patterns of co-occurring words as weighted relevant topics, without using lexical resources. Based on distributional hypothesis, it assumes that

[3]Separator character used is *white-space*.

[4]Such as *couldn't*, which should be replaced by *could not*

[5]Algorithm is called `PunktSentenceTokenizer` [3]

words close in meaning will occur in similar location of text, thus, in case of text summarization, in same sentences. In this way, it is possible to understand which sentence deals with which topic and to select sentences related to most important themes.
The algorithm is implemented with the following steps. Topics representation starts with filling an *nxm* matrix A, each row corresponding to an input word and each column to a sentence. Each entry a_{ij} records the weight of the i-th word in the j-th sentence, zero if the word is not contained in the sentence. Non-zero values can be of three types:

- Term Frequency $tf_{t,s}$, which is not convenient because does not consider the different lengths of sentences. In fact, it is better that weights of words from a short sentence are bigger, as we can say they exerts more influence;
- Term Frequency normalized by sentence length $\frac{tf_{t,s}}{|s|}$
- Tf-idf
$$tf.idf = \frac{tf_{t,s}}{|s|} \bullet log(\frac{N}{df_t})$$
which derives from the product between tf and *inverse document frequency* idf. This value proportionally measures both frequency in the single sentence and the diffusion in the other ones, thus increasing with the rarity of the term in the collection, that is the specificity of that term in relation to that sentence. For this reason, it is the most correct weighting function for this type of co-occurence analysis.

Matrix A can be decomposed into three sub-matrices:

$$A = U\Sigma V^T$$

where:

- U records the weight of each word for each topic
- Σ is a diagonal matrix whose non-zero values represent the topic importance
- V^T consists of a new representation of the sentences, each expressed no longer in terms of the words it contains but in terms of the topics it relates to.

The product $D = \Sigma V^T$ combines topic importance scores with the new representation of the sentences and indicates to what extent the sentence communicates a certain topic. Therefore, d_{ij} is the weight of the argument i for the sentence j.
Sentences presenting many of the most important topics - based on weights recorded in Σ matrix - are often excellent candidates for the summary. For this reason, the algorithm identifies these candidates by calculating for each sentence

$$s_i = \sqrt{\sum_{j=1}^{m} d_{ij}^2}$$

As anticipated, when the algorithm uses the term frequency divided by the length, it tends to favor shorter sentences with frequent terms, while exploiting tf.idf function it operates an additional filter on those sentences that contain rarer words, those more "specialized" with respect to the topic. On the contrary, application of simple term frequency weights only makes method focus on the raw number of times a word occurs.

3.2 Indicator Representation methods

Original text can also be analyzed by means of a set of relevance indicators, which do not aim to discover topics. In this case, each sentence is indeed described by a list of features - such as length of the sentence, position in the document, presence of certain words, etc -, which can be then combined in many ways to obtain a unique scoring useful to rank and select most important sentences. Three different approaches have been implemented.

3.2.1 Graph-based: TextRank

Graph-based summarization methodologies derive from PageRank algorithm, which assigns numerical weights to the related elements of a set, with the aim of quantifying the relative importance within the group. In this case, the method will represent the entire review as a network of inter-connected sentences, each one associated with a node. Edges join these vertices if and only if the similarity between the two given sentences exceeds a certain threshold. At the end, graph representation will return two outputs:

- the graph partition, from which it is possible to deduce a set of discrete topics extracted from the corpus
- important sentences, namely those connected with many of the others in the partition, and which will most likely be included in the summary

Among all the existing variants, the most popular was chosen: the *TextRank* algorithm. First of all, *TextRank* splits the review into sentences - with the aforementioned *sentence tokenizer* - and applies pre-processing operations to them.
Then, a vector representation for every sentence has to be produced. An alternative would be to resort to the weighting functions used with *LSA*, but using them to measure similarity would bring strong limitations, because they do not take into account syntactic and semantic information, like the order of the words. For this reason, sentences will be represented through the word embeddings produced by the pre-trained model called *"Global Vectors for Word Representation" GloVe* [5]. GloVe is an unsupervised learning algorithm for obtaining words vector representations[6]. This embeddings are created by mapping words into a space where the distances are related to semantic similarity. Training is performed by extracting global statistical information from the

[6] Precisely, we will be using the pre-trained *Wikipedia 2014 + Gigaword 5 GloVe* vectors

non-zero values of a co-occurence word-word matrix, rather than from the entire sparse matrix - like matrix factorization methods do - or from the single context window within the large corpus, - like *word2vec* does. The resulting representations show interesting linear substructures of the word vector space.
GloVe will thus consider 400k terms[7], each one described by an array of 100 values. Finally, sentences vector representation will be obtained by computing the mean of all the vectors representing each word in the given sentence, gaining a consolidated vector for the sentence.
The third step consists in compute the cosine similarity among all the sentence vectors and store it into a *nxn* matrix.
This similarity matrix is converted into a graph, with sentences as vertices and similarity scores as edges.
Hereafter, *TextRank* implements *PageRank* algorithm to extract an importance score, as aforementioned, and rank sentences based on their score. Finally, the final summary is created by selecting the top N sentences.

3.2.2 TextRank variant

This project also calls for the implementation of a packaged TextRank variant: `PytextRank` [6]. This performs a lemmatization of the text, constructing a *lemma graph* to represent links among the candidate sentences.

3.2.3 Feature-based: Luhn's Algorithm

Luhn's Algorithm [7] is an early heuristic method which performs Extractive Summarization task by computing the "significance" of the word from the frequency. The significance in the document is interpreted as an evaluation of the relevance of a term within the sentence. The algorithm is a naive approach based on tf representation and looking at the "window size" of non-important words among relevant words.
The stages are:

1. Extraction of keywords in the text, namely significant terms, by selecting those above a minimum and under a maximum threshold of - normalized by review length - term frequency $\frac{tf_{t,d}}{|d|}$, thus ignoring most frequent and rare ones

2. Computation of sentences weights based on the squared number of keywords divided by the windows size, which is the maximum distance between two significant words:

$$s_i = \frac{\text{Number}^2 \text{ of Keywords in i}}{\text{Span}}$$

3. Sorting sentences in descending order of score and selection of the best N-ranked.

In this case, minimum and maximum are chosen to be respectively 5% and 50%.

[7] Otherwise, if the embedding of one word does not exist, the algorithm returns a vector of zeros.

To sum up, the most important sentence will be the one maximizing the number of terms which are neither too frequent - over the half number of tokens in the entire review - nor too rare and/or minimizing the span between most distant keywords.

3.2.4 Deep Learning based: BERT Summarizer

All summarization models just presented have big limitations. Beyond technical reasons, this is due to the existence of a semantic threshold dividing the human being from the computer: men, in fact, manage to recognize the context of the sentences at a greater depth, extracting more clear meanings. Although it could not be enough to break down this threshold, to obtain better summaries it is necessary to reach a higher level of abstraction in the model, able to capture as much information as possible from the text. For this reason, neural networks are used. In particular, one of the neural networks most frequently involved in Natural Language Processing tasks is BERT model.
Bidirectional Encoder Representations from Transformers BERT [8] is a pre-train deep bidirectional neural network which offers representations from unlabeled text. While directional models read text sequentially - left-to-right or right-to-left -, BERT reads the entire sequence of words at once. This characteristic allows it to learn the context of a word based on all of its surroundings: both left and right words. In details, the context is extracted by a Transformer, a mechanism that finds contextual relations between terms in a text. The Transformer includes two separate components:

- an encoder reading input review
- a decoder producing a prediction for the NLP task to be accomplished.

For any of these tasks, BERT's goal is to generate a language model, represented by a sequence of vectors in output, each one associated to an input token.
Moreover, the analysis is enhanced by using *coreference techniques*. *Coreference* occurs when two or more expressions in a text refer to the same entity. In computational linguistics, to derive the correct interpretation of a text and estimate the relative importance of various mentioned subjects, the referring expressions must be connected to the right element. In this way, BERT succeeds in inferring a better context for each word.
As said, this neural network can be modified to create several state-of-art models for NLP tasks. In particular, as far as text summarization, a *Bert Extractive Summarizer* [10] has been recently released by fine-tuning BERT architecture. This Summarizer works by first embedding the single tokens. Secondly, it builds a "Segmentation Embedding" to detect sentences boundary and embed them. Thirdly, the model embeds the position in the document. Finally, the algorithm performs a clustering process, finding the sentences that are closest to the cluster's centroids.

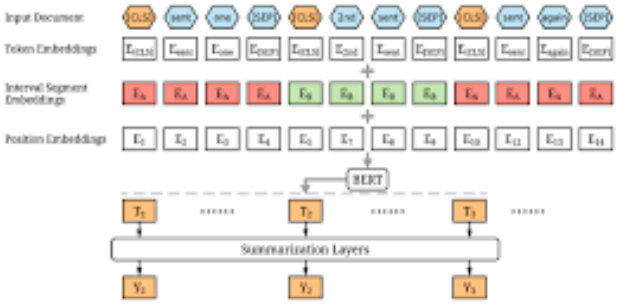


Figure 5. Text Representation in BERT Summarizer

4. Abstractive Summarization

Differently from what has been done up to now, Abstractive Summarization is not configured as a mere selection of important sentences, but as a method to express the concepts of the source in a new interesting form. The model learns and reproduces its own expressions and sentences to offer a more coherent summary, similar to what a human being would generate. In this way, its result will not copy the original structure, words and sub-sentences - the majority of which, even if present in the most relevant sentence, often prove to be useless. On the contrary, the information needed to achieve the goal will be searched in the patterns contained within a training corpus. It is therefore clear how, in this case, it is fundamental to implement an even more complex model than the previous ones.

4.1 Seq2Seq Modeling with Attention Mechanism

A "*Sequence-to-Sequence*" task is a problem involving sequential information. Text Summarization can be considered as a sequence modelling problem, because any prediction depends on the analysis of the sequence of tokens and sentences. More precisely, it is configured as a *Many-to-Many Seq2Seq* problem, where both input and output - the original Amazon reviews and respective summaries - are sequences composed by more than one token.

As explained in the previous section, the typical *Seq2Seq* model relies on two principal components: an encoder and a decoder. The task of the encoder network is to create a smaller dimensional representation of the input text. This representation is then forwarded to a decoder network which generates a sequence representing the output. This *encoder-decoder* architecture is particularly suitable, as is the case with Text Summarization, when input and output sequences are of different lengths.

The preferred class of models for such operations is therefore that of *Recurrent Neural Networks (RNNs)*. What distinguishes *RNNs* from normal feed-forward networks is that hidden layers, in addition to inputs from the input layer, also receive their own output associated with the previous state, conveniently weighted. This mechanism serves to provide the neural network with a memory about the sequence of tokens. Nevertheless, *RNNs* are not able to model long-term dependencies because their back propagation algorithm can cause two negative effects: vanishing gradients and exploding gradients. For this reason, other models are preferred that can avoid these risks and describe the dependencies among tokens within long documents. In this project, the chosen alternative is *Long Short Term Memory LSTM*, wherein the function just mentioned is implemented through blocks - gates -, executing several operations.

Thus, the final architecture is constituted by two neural networks: an *Encoder LSTM* and a *Decoder LSTM*. Encoder and Decoder work together to find the best way to predict the target sequence. At each timestep, the Encoder receives an additional word of the sequence, processes the new data and acquires the contextual information. Then, the final state of the Encoder, i.e. the output of the last timestep, initializes the Decoder component, which in turn reads the entire target sequence token-by-token and predicts a sequence offset at each timestep. In this way, the Decoder *LSTM* becomes able to predict the next word in the sequence given the previous one.

"start" and *"end"* in Figure 6 are special tokens added to the target sequence, before passing it to the Decoder, which respectively begin and stop the prediction.

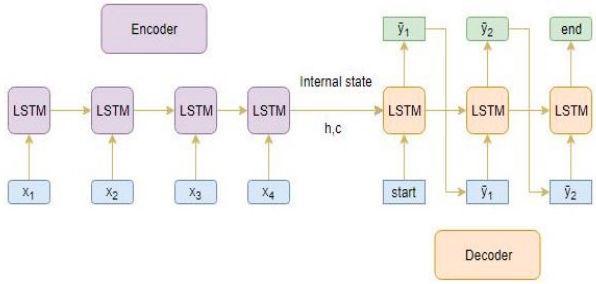


Figure 6. Encoder-Decoder LSTM architecture

4.1.1 Attention Mechanism

This architecture composed of an Encoder, which converts input into fixed length vectors, and Decoder, which predicts summaries by reading this entire embedding, is plagued by a main limitation: the Encoder struggles to compress long input sentences into a vector and the performance of the neural network deteriorates rapidly as the length increases. This bottleneck is overcome thanks to the so-called *"Attention Mechanism"* [11], which aims to produce the next word by looking only at specific parts of the sequence. The key intuition behind it consists in selecting the words of the input text which are more relevant to generate a new token at each timestep. First, this is made by computing a score e_{ij} evaluating the alignment between

- h_j, i.e. the hidden state output by Encoder at each timestep j of the input sequence
- s_i, i.e. the hidden state output by Decoder at each timestep i of the target sequence

In other words, e_{ij} is based on semantic links among input and target tokens, and, in our case, corresponds to the following additive formulation:

$$e_{ij} = V^T tahn(W[s_i, h_j])$$

where V and W are the two trainable weights matrices for the neural networks, thanks to which it learns to predict the next word in the sequence. The e_{ij} scores are converted to mutually exclusive probabilities using a softmax function, producing the so-called "attention weights":

$$a_{ij} = \frac{e^{e_i j}}{\sum_{k=1} e^{e_i k}}$$

Then, computing the linear combination among the attention weights and hidden states of the decoder h_j, the algorithm extract the context vector of timestep i C_i. C_i is concatenated to the hidden state of the decoder at timestep i s_i, and this result is finally mapped to the output: the word at timestep i, $\hat{y}_i$ in Figure 6.

4.1.2 Final model: Training phase

During the training process, the final architecture involves the following components and operations:

1. At each iteration, an Input layer is initialized with a batch of texts to be summarized, represented by a vector of the indexes of the words they contain. All this vectors compose a matrix.
2. An Embedding layer receives the matrix just mentioned and compresses it in a dense vector of fixed dimension
3. This dense vector is passed to a block of 3 LSTM stacked on top of each other, which allows to gain a better representation of the sequence compared to a single LSTM. Within this block, each of the 3 components returns its internal output to the next one. Additionally, to improve performances, they all incorporate a dual dropout mechanism:
    - a dropout in the input conversion of each internal iteration
    - a recurrent dropout when calculating hidden outputs

These first 5 layers make up the Encoder part. The Decoder, however, consists in the following ones:

4. Its own Input layer
5. An Embedding layer, with the same functioning as that of the Encoder

Then, a new LSTM, with exactly the same settings as the previous ones, jointly receives

- the latent representation acquired by the Decoder from the summaries of the training set
- the internal outputs of the third stacked LSTM

At this point, the attention mechanism between the Encoder and Decoder, as previously defined, is activated. Finally, the output of the Attention Layer is mapped to the final result of the neural network through a "time-distributed" wrapper, which applies a dense layer - activated by a softmax function - to each timestep of the sequence. The final model architecture, as explained, appears as in the Figure in Appendix.

The updating of the neural network weights is defined by the Root Mean Squared Propagation. We opt for a sparse categorical cross-entropy as the loss function, because, unlike the normal cross entropy, it converts the integer sequence into a one-hot encoded vector, saving time and memory thanks to the simplification of the calculation.
This Summarizer will be trained on batches with size of 128, and validated on a test set corresponding to the 10% of the original "Fine Food reviews" dataset.
The "`EarlyStopping`" callback is introduced, which stops training procedure in case validation loss value has stopped decreasing for 2 epochs. Due to this early stopping condition, the model training stopped at 29th epoch, requiring ~ 7 hours[8]. The learning process is visualized with Figure 8:

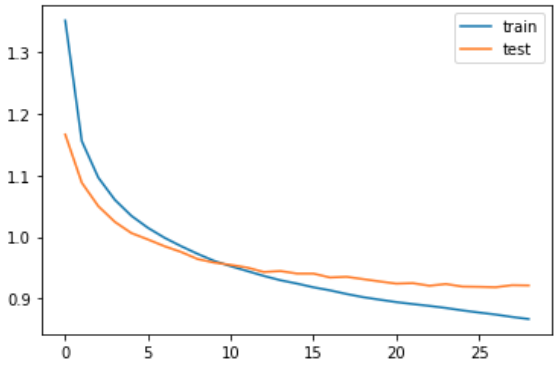


Figure 7. Loss results over the epochs

A slight increase in the validation loss is shown after tenth epoch.

4.1.3 Final model: Inference phase

After training, the inference process involves 6 steps:

1. Encoding of the input sequence and initialization of Decoder with subsequent internal states of the Encoder
2. Passing of *"start"* token as input to the Decoder
3. Running of the Decoder in combination with the internal states related to current timestep
4. Computation of the probability for the next word and selection of the one maximizing this probability

[8]The relatively short execution time comes from the use of a GPU on the *Google Colab* platform.

5. Passing of this selected word as an input to the Decoder in the next timestep and update the internal states with the current timestep

6. Repetition of 3rd, 4th and 5th step until

   - *"end"* token is generated, i.e. when the input sequence runs out

     or

   - when hitting a maximum target sequence length

5. Evaluation

The goal of this project is to compare different Text Summarization methodologies, testing performance and results through the production of a single sentence that could exhaustively extract the essence of an Amazon review.
Generally, different summarizers are compared based on popular scores in the literature - like *ROUGE* or *BLUE* -, but this time an approach based on human judgment[9] is chosen, because, although it is a simple and expensive method, it allows to have a clearer and more direct view on the real capabilities of each model. Therefore, after having randomly selected 27 reviews to be summarized by all the models, these will be assigned with a score from 0 to 3 in terms of effectiveness and utility, namely the ability of reporting the main opinion expressed by the Amazon user about the given item. The score will be:

0 , if summary is completely not understandable

1 , if summary is clear but returns other concepts or the opposite idea with respect to the original text

2 , if summary is well comprehensible and provides an information, but this information does not sum up effectively the content of the text

3 , if summary is perfectly comprehensible and provides the right information about the main content of the text

The final evaluation will be computed as the average score over the reviews.

6. Results and Discussion

Any consideration about the models capabilities on a test set of 27 reviews can start from the observation of the relationship between the average quality of the summary produced and the variance of the results. Table 1 shows this data numerically, and the beneath Figure 8 graphically represents it.

[9]Two people have evaluated the summaries.

	Mean	Std. Deviation
LSA	2.259	0.903
TextRank	2.407	0.747
PyTextRank	1.926	0.917
Luhn's	1.907	0.855
BERT	2.185	0.834
LSTM	2.185	0.921

Table 1. Quality mean and standard deviation.

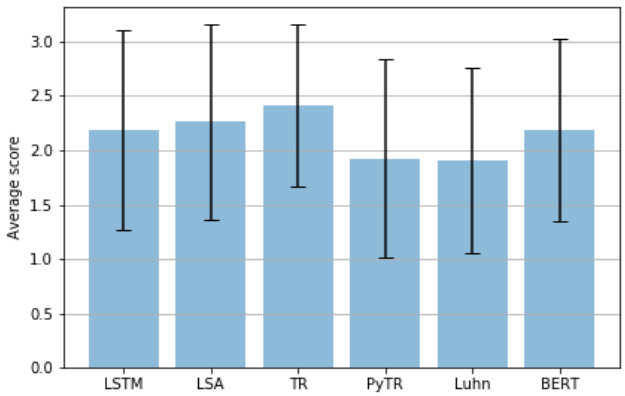


Figure 8. Results in order: Abstractive Summarizer, Latent Semantic Analysis, TextRank, *PyTextRank*, Luhn's algorithm and *BERT* Summarizer.

The implementation of the TextRank algorithm appears to be the most satisfactory, having significantly more positive mean and variance than the other models. This, in addition to its intrinsic characteristics, can derive from the choice to exploit *GloVe* to obtain the vector representation of the text. The embedding extracted from it evidently collects efficiently the context necessary to interpret the importance of each term in the summary. The second best average quality is achieved by the Latent Semantic Analysis, which however has a much greater variability than TextRank. The topic-based strategy turns out to be good, but not excellent. Its higher standard deviation translates into less reliability. Contrary to what might have been expected, BERT fine-tuned neural network, despite its depth, cannot reach a high average, but still offers a lower variability. Finally, Luhn's algorithm and TextRank variation - `PyTextRank` - prove to be the most scarce. As for the first, this result can be considered obvious, due to its simplicity: the mere counting of keywords within the sentences, without delving deeper into their context, could not have been sufficient. `PyTextRank`, on the other hand, can be interpreted as a more superficial implementation of the hand-crafted TextRank algorithm: it exploits a low level text representation - *tf.idf* - and performs lemmatization, which inevitably involves a serious loss of meaning. Its poor result and its variability demonstrate how important it is to consider the context of words and not to perform any lemmatization/stemming operation.
Regarding Sequence-to-sequence model with Attention Mech-

anism used for an Abstractive Summarization, we can be satisfied, because, despite the greater complexity of the task, the internal mechanisms of the model manage to produce decent summaries, on the same average level as BERT. Anyway, as expected, the model is the least reliable. In details, the summaries produced are often short, have a very simple syntax, and their expressive ability varies: sometimes the result is incomprehensible, other times it reports a message which is exactly opposite compared to the original review; in best cases, it manages to synthesize the meaning much more effectively than other extractive approaches.
Further disadvantages are the extreme computational heaviness and the consequent lengthening of the time needed to train the model.

Possible improvements to the last model could be:

- increasing training dataset size to enhance the model generalization
- implementing a Bi-Directional LSTM architecture, able to capture the context from both the directions - like in BERT structure previously seen -, resulting in a better context vector
- incorporating more complex mechanisms inside the architecture, like *pointer-generator networks*, *coverage mechanisms* and *beam search strategy*.

Future works will also have to consider more objective evaluation paradigms.

Code

Code is available at this link.

References

[1] *Amazon Fine Food Reviews - Analyze 500,000 food reviews from Amazon*, Kaggle Dataset, 2016.

[2] J. McAuley, J. Leskovec. *From Amateurs to Connoisseurs: Modeling the Evolution of User Expertise through Online Reviews*, 2013.

[3] S. Bird, E. Loper, J. Nothman, A. Darcet. *Natural Language Toolkit: Punkt sentence tokenizer*, 2006.

[4] R. M. Badry, A. S. Eldin, D. S. Elzanfally. *Text Summarization within the Latent Semantic Analysis Framework: Comparative Study*, 2013.

[5] J. Pennington, R. Socher, C. D. Manning. *GloVe: Global Vectors for Word Representation*, 2014.

[6] P. X. Nathan. *PyTextRank Documentation*, 2019.

[7] H. P. Luhn. *The Automatic Creation of Literature Abstracts*, 1958.

[8] J. Devlin, M. Chang, K. Lee, K. Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019.

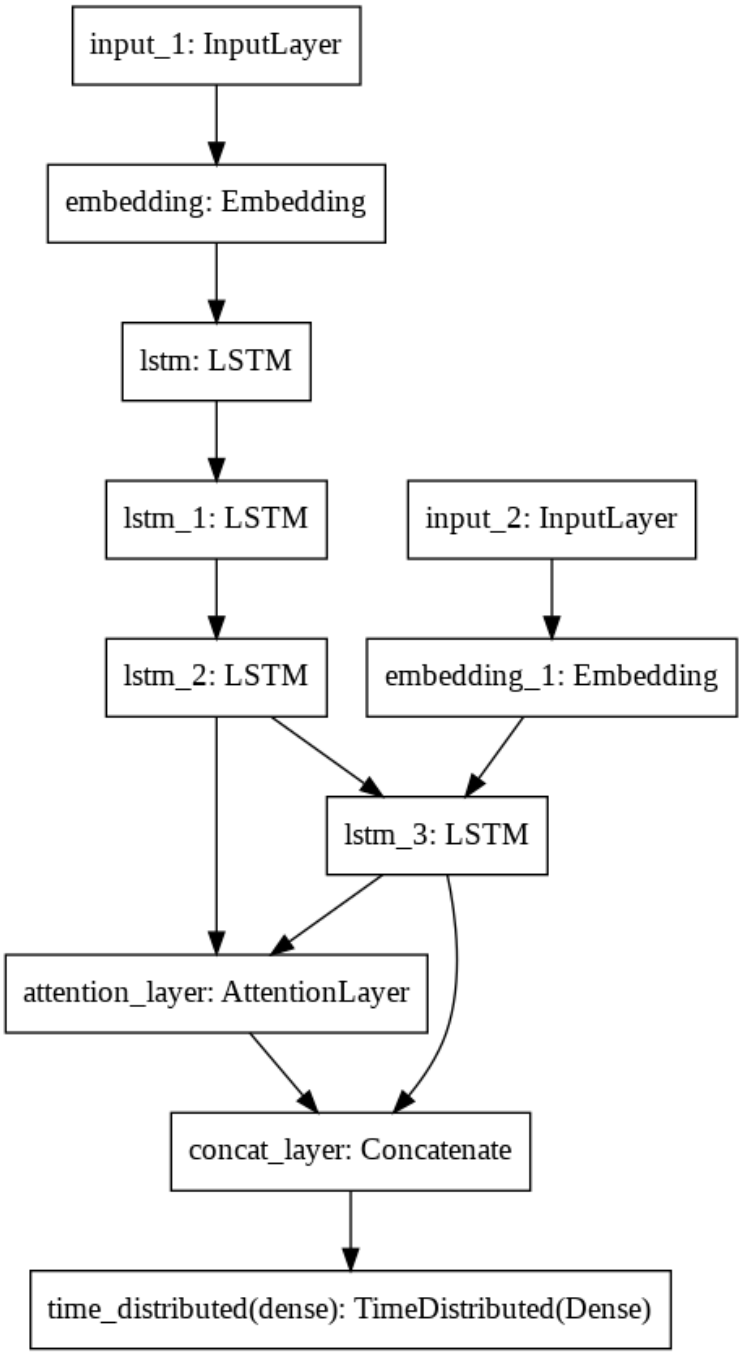
[9] R. Horev. *BERT Explained: State of the art language model for NLP*, 2018.

[10] D. Miller. *Bert Extractive Summarizer*, 2019.

[11] D. Bahdanau, K. Cho, Y. Bengio. *Neural Machine Translation by jointly learning to align and translate*, 2015.

[12] F. Shaikh. *Essentials of Deep Learning – Sequence to Sequence modelling with Attention (using python)*, 2018.

Appendix



Final architecture of Seq2Seq model for Abstractive Summarization