Universitat Politècnica de Catalunya - BarcelonaTech

Master Thesis

Text Summarization Of Online Hotel Reviews With Sentiment Analysis

Author
Guifré Ballester Basols

Supervisor
Alfredo Vellido Alcacena

A thesis submitted in fulfillment of the requirements for the Master in Innovation and Research in Informatics, Facultat d'Informàtica de Barcelona

in the

Data Science Speciality

October 16, 2021

Abstract

The aim of this thesis is the creation of a system that summarizes positive and negative property reviews. To achieve this, an extractive summarization system that produces two summaries is proposed: one for the positive reviews and another for the negative ones. This is achieved with a classification system that will feed positive and negative reviews to the summarization system.

To pursue our objective, a study on the different NLP methods, along with their pros and cons, was performed, leading to the conclusion that the use of transformers and more specifically, the combination of BERT and GPT-2 architectures, would be the best approach. To obtain the data from TripAdvisor that is in StayForLong website, a crawling process was performed from the StayForLong and TripAdvisor. These consisted on a total of over 80000 reviews, and over 175 properties that we pre-processed, cleaned and tokenized, in order to work with BERT for the sentiment analysis and GPT-2 for the summarization. Then we proceeded, with an extensive analysis in regards to the impact of the variables. Finally, we fine-tuned each of the models so that it performed at its possible best.

To evaluate our two systems, we evaluated the the binary sentiment classification system, with multi-modal BERT with a 96% of precision and for the GPT-2 summarization system, we opted to apply the ROUGE-F1 metric, were we obtained an average of 57.5%.

Index

List of Figures

List of Tables

Acronyms

AI Artificial intelligence.

API Application Programming Interface.

BERT Bidirectional Encoder Representations from Transformers.

BPE Byte-Pair Encoding.

CBOW Continuous Bag of Words Model.

CD cardinal number.

CRISP-DM CRoss Industry Standard Process for Data Mining.

DT determiner.

EDA Exploratory Data Analysis.

ELMo Embeddings from Language Model.

GloVe Global Vectors.

GPT Generative Pre-trained Transformer.

GRU Gated recurrent unit.

IN preposition / subordination conjunction.

JJ adjective.

JSON JavaScript Object Notation.

LSTM Long short-term memory.

ML Machine learning.

MLM Masked Language Modeling.

NLP Natural language processing.

NMF non-negative matrix factorization.

NSP Next Sentence Prediction Loss.

OOV out-of-vocabulary.

OVW outer vocabulary words.

POS Linear Discriminant Analysis.

POS Part of Speech.

RB adverbs.

ReLU Rectified linear activation function.

RNN Recurrent neural network.

ROUGE Recall-Oriented Understudy for Gisting Evaluation.

SGNS Skip-gram with negative sampling.

T5 Text-to-Text Transfer Transformer.

UNK out-of-vocabulary.

VB verb.

1 Introduction

1.1 Project Formulation

Nowadays, and while this thesis is being written, the amount of information one person can access keeps growing. With the arrival of the internet, information can be easily shared. In the domain of tourism and hospitality, what could only be transmitted by word of mouth, or found at touristic guides, now can be published and freely accessed by anyone from the comfort of their computers and smartphones. All this, together with the massive use of technology, rises an issue: there is too much information and not enough time to consume it, while this amount of information continually grows. The main issue is no longer having information but rather gaining insight from the data and, if possible, in a shortened period. In this thesis, we will be focusing on property reviews that have been written online to filter them by their opinion and generate a summary. What do people that had a negative impression complain about? What do people that had a favorable impression have to say? The idea of this thesis comes from the necessity that we at StayForLong[1], a company of the tourism sector, have detected: providing the user with a numeric score is a good idea; it helps a lot. Giving a user all the reviews from people who have shared their opinion on a given hotel it is also good. However, the truth is that it is very time-consuming on the user end. Therefore, we aim to provide an easier and more comfortable way of understanding people's opinions, which will lead to a minor strain on the user's side, reflecting a better user experience that potentially, can help to increase bookings.

As we have just mentioned, we have two branches to work in: summarization and opinion mining. Summarization has been studied for decades, and we can find lots of literature that try to tackle the problem. In opinion mining, we can see that is a new topic that gained importance specially in the last years. We can also find literature in a combination of all areas applied to reviews. As we will see in our state-of-the-art and in general with all Natural Language Processing (NLP) literature, methods proposed in the early stages were quite naive. However, as artificial intelligence has progressively developed, we can see now that NLP techniques have improved significantly, every time getting closer to the human level while mostly being open-source. Regarding the available data, information related to hotel/properties reviews,

although publicly available, is not easily obtainable, which will make us work in scrapping this information. Even though our main problem is to summarize a text previously filtered by its sentiments, we will need to process this data and apply other standard NLP techniques to prepare the text.

This problem is attractive to different groups of stakeholders:

- The guest is looking for a hotel and does not want to spend lots of time looking at the reviews.
- Websites or agencies that present a hotel in a platform and want to provide a piece of very compact information to the user to have a more straightforward selecting process, resulting in a more impulsive purchase.
- The last stakeholder is the hotel itself. If they can get the opinion of a specific group of exciting people, they can understand their strong and weak points in a more effortless and more refined manner.

To support the importance of addressing issues that arise on customer reviews, we want to provide some data[3] on how dissatisfaction affects business profitability:

- 86% of consumers stop doing business with a company over a bad review.
- Good customer experiences lead 42% of the customers to consume again.
- An unhappy customer has a 91% chance of not doing business with that company again and will spread from a range to 10-20 people about their experience.
- It is estimated that a bad review will need 40 positive reviews to undo the damage of that negative review.

In conclusion, it is critical to stop bad reviews as soon as possible since it hurts the business in the short and long term. So being able to narrow down our data is going to be helpful in cases such as this.

1.2 Project Main Goals

As stated in the project formulation section, our goal is to summarize multiple reviews from a given hotel that have a similar opinion. This goal comes associated with multiple challenges that we need to cover:

- **Get to know the NLP ecosystem:** Even though NLP is a subset of machine learning, it is a very particular ecosystem. There are many particularities that are only used when dealing with text that would otherwise not be considered. There are not only models that we might not know about, but also all the text processing pipelines: tokenization methods, removal of unwanted characters, stemming, lemmatizing, etc., just to mention some. Therefore, to do a good job, we need to understand the basics of NLP and what has been studied in text summarization and opinion mining.
- **Obtaining a dataset to work with:** As mentioned, we want to work on hotel/property reviews. In order to work with this information, we need to either find a public dataset that suits the analysis, or crawl information from a website.
- **Defining a methodology:** We need to define a methodology that allows us to evaluate if one method is better than another. We also need to understand which are the human boundaries of the problem. Can we surpass them? Can we get to the human level of summarization? Without a methodology and those boundaries, it is impossible to progress since we cannot see if we are improving or simply, for instance, reached the boundaries of the problem. If, for instance, we have a problem where a human can score 85% accuracy, our system scoring an 82%, would be an excellent result, right? We would not be a 15% away, but rather a 3%.
- **Achieving good results:** We intend to achieve the best possible results, given a suitable methodology and a suitable approach.

These goals that we have just mentioned will expand as we study further but, in essence, are the core of our research.

2 State-of-the-art

Since the problem tackled covers by multiple fields and covering all of them individually would be very extensive yet unclear, we opted to present the state-of-the-art in a way that we believe is the clearest: NLP will be first introduced to the reader. Once introduced, methods and techniques that are used in NLP in general will then be covered. However, extensive research on what has been done up to the date will not be carried out, at least until reaching a point in the literature that is interesting and significant to our goal. We believe that over-emphasizing what has been done decades ago will not be effective in terms of clarity, even though knowing where we come from, might help us to understand where we stand in this problem.

This approach was not followed because, once working on a state-of-the-art, we opted to start from the beginning, somehow finding ourselves in the quandary that the effort to do the state-of-the-art on for instance, the 1950-2000's period on summarization, was not really worth it. This is because it got to the point that those methods, even if somehow relevant, are pretty much outdated.

2.1 Introduction to NLP

> “ *Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications.* ”
>
> Liddy, E.D., *Natural Language Processing[4]*, 2001

As we can see from the definition, NLP can be used for a range of multiple applications, including Information Retrieval, information extraction, question-answering, summarization, machine translation, and dialogue systems. Of course, not all of these tasks are going to be included in our system, but we can see that summarization and opinion mining are included in the list of tasks that NLP systems are built for. Therefore, what we are going now to do is to introduce each of these fields since there are multiple solutions

where these solutions can be applied.

But the truth is that NLP is a very multidisciplinary field that it is related to many other similar fields. But generally, we could say that is formed by: linguistics, text mining, artificial intelligence where of course, we find machine and deep learning methods among others.

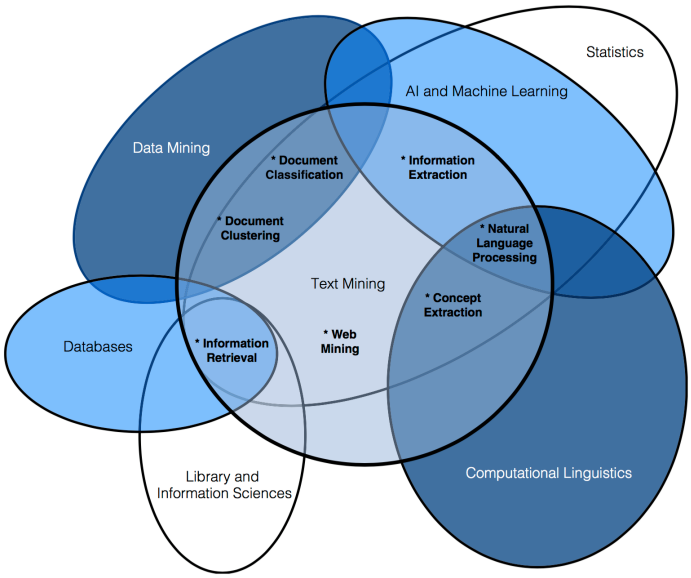


Figure 1: Venn Diagram of text mining[5]

2.1.1 Text Summarization

To have an idea of what text summarization means, we are going to look at a formal definition of what text summarization is:

> *A text that is produced from one or more texts, that conveys important information in the original text(s), and that is no longer than half of the original text(s) and usually significantly less than that.*
>
> D. R. Radev, *Intr. Special Issue on Summarization[6]*, 2000

Moreover, from what is given on the definition of text summarization, we can see that there are different types of summaries and different parameters, which mostly depend on our inputs, outputs, and purpose.

- **Input:** The input is what we are going to feed our system. This input can be composed of multiple documents, or it can be a single document. It can also be a query.
- **Output:** The way that the results are shown, that are compressed to a certain ratio. There are several formats such as headlines, outlines, minutes, biographies, abridgments, sound bites, movie summaries, chronologies, etc. [8]
- **Purpose:** Indicative, informative, and critic summaries. Where the indicative indicates types of information, informative include quantitative/qualitative information, and critic evaluates the document's content.
- **Form:** There are two forms: extractive and abstractive. Extractive selects units of the original text: there is no simplification nor rewriting. On the abstractive, the system can produce outputs containing words that have not been given as inputs. Currently the form that is more mature on literature is the extractive summarization and currently the state-of-the-art is moving more towards real time summarization and abstractive. Which are more complex tasks.
- **Dimensions:** Single document vs multi-document
- **Context:** Query-specific vs generic. By query-specific, we mean that the summary is based on a question. Whereas generic covers all aspects treated on the given text.

So as can be seen, a summarization system can have different requirements; we might need a concise summary or a most extensive one; a summary regarding an aspect we are curious about, or a document or a collection of documents. All these features will change depending on our purpose and objective, and will have to be adapted depending on the problem. So a summarization system is not a single-shot solution for all types of inputs. We need to adapt it to our needs and construct it according to the inputs and the outputs desired.

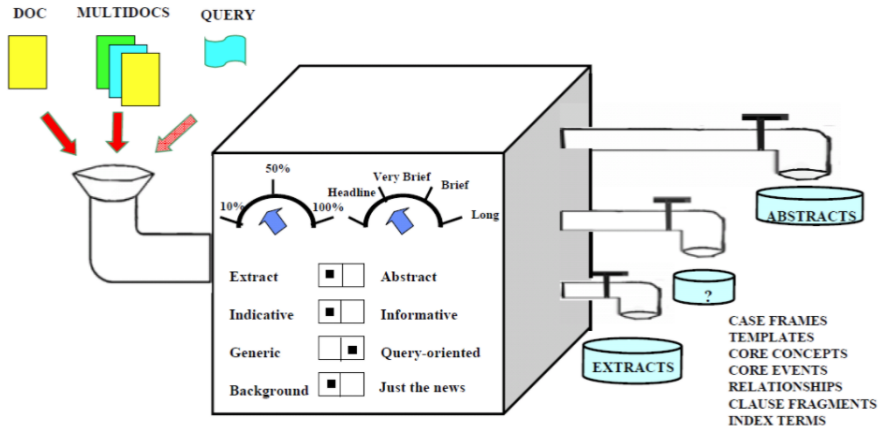


Figure 2: Text summarization diagram

Therefore, we will need to make different decisions, which typically are structured with the following stages:

- **Content Identification:** Given our input documents; we have to determine which information we pass on to the next stage.
- **Conceptual organization:** How the information is organized. Do we want to combine multiple sources? Do we want to preserve entire sentences?
- **Realization:** We have to deal with some other problems. For instance, if we join two sentences, we might need to join them to be coherent.

2.1.2 Opinion Mining

In the same fashion than with text summarization, we are going to first see a formal definition that will help us to have a better understand of what opinion mining (also known as sentiment analysis, sentiment extraction, affective rating)[7] actually is:

> *Given a set of evaluative text documents D that contain opinions (or sentiments) about an object, opinion mining aims to extract attributes and components of the object that have*

> *been commented on in each document d∈D and to determine whether the comments are positive, negative or neutral.* ”

S. F. Esuli A, *Determining the semantic orientation of terms through gloss classification[27]*, 2005

Even though this section does not delve in depth through the state-of-the-art, it is worth mentioning that opinion mining, in comparison to text summarization, is a relatively new topic. Sentiment analysis appears on the late 90's and it was not until recently that it became trendy thanks to important contributions. This is not only because of the machine and deep learning growth, but more due to the fact that textual information can be classified mainly in facts and opinions, where facts are objective statements and opinions are subjective ones. The problem with subjective statements is that, until the advent of the world wide web, it was basically less accessible and it has not been until now with the rapid growth that has democratized the internet, that people can share their opinions far more easily than earlier on, when sharing was restricted mostly to local communities, family meetings, etc.

As in text summarization, we can apply sentiment analysis to different aspects of a text:

- **Subjectivity analysis:** We have to decide if the information that is given is actually subjective or not.
- **Polarity analysis:** Labeling things as either positive or negative / providing a rating such as rating star.
- **Sentiment target:** Which is the target that we are aiming? A subject might have a positive rating on something but a negative on another one.
- **Level granularity:** Which expresses to which level we are expressing sentiment. Some examples could be document, sentence or attribute levels.

In the same way that we had issues on summarization, sentiment analysis also experiments difficulties with subtlety, concession, manipulation and sarcasm/irony which are per se, difficult problems on NLP.

2.2 Models and their flaws

As we have mentioned, NLP has literally been revolutionized in the last years. There has been a huge improvement in terms of complexity and performance, which has basically happened to AI systems in general, but it heavily applies in environments such as NLP. In this section, we are going to cover most of the techniques that we find that are relevant nowadays and that used to be relevant, and we will also say what are they missing. To follow up with a more complex method, that does better.

2.2.1 Logistic Regression

Logistic regression is a statistical model normally used to model a binary dependent variable (it can be extended to work with multi-way categorical, ordinal dependent variables, among many of the possible extensions). To be able to model this binary behaviour, logistic regression makes use of the sigmoid function, which is written as follows:

$$h(x^{(i)}), \theta) = \frac{1}{1 + \epsilon^{-\theta^T x^{(i)}}} \tag{1}$$

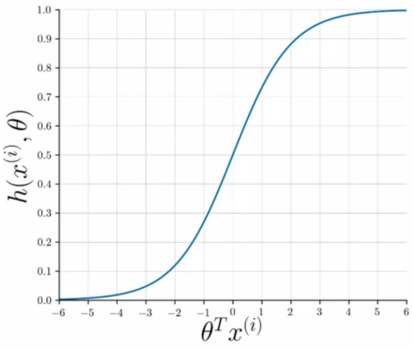


Figure 3: Sigmoid function

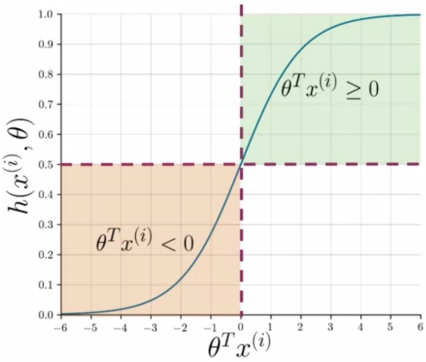


Figure 4: Sigmoid function y = 0.5

In order to decide when we classify in one class or another, we need to set a threshold. This threshold normally is set to 0.5, which corresponds to the product of θ^T and $x = 0$. Therefore when $\theta^T x^{(i)} \leq 0$ the prediction is 1. Otherwise, when $\theta^T x^{(i)} < 0$ it corresponds to 0.

To train the model we need to learn which values of θ, minimizes our cost function. The cost function used for logistic regression is the average of the log loss across all training examples:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^i \log h(z(\theta^i)) + (1 - y^{(i)}) \log(1 - h(z(\theta^i))) \tag{2}$$

where:

- m is the number of training examples
- $y^{(i)}$ is the actual label of the i-th training example.
- $h(z(\theta)^{(i)})$ is the model's prediction for the i-th training example.

In order to be able to optimize our loss function, we are going to use gradient descent which will seek for the local minimum. It is important to set a learning rate that is adequate so that we do not bounce back and forth between the convex function of gradient descent. The gradient of the cost function J with respect one of the weights θ_j is defined as:

$$\nabla_{\theta_j} J(\theta) = \frac{1}{m}\sum_{i=1}^{m} (h^{(i)} - y^{(i)}) x_j^{(i)} \tag{3}$$

The gradient algorithm updates the weights, by subtracting a fraction of the gradient by α. This is the learning factor that we mentioned earlier that has to be picked carefully, not to learn too fast or too slow.

$$\theta_j = \theta_j - \alpha \times \nabla_{\theta_j} J(\theta) \tag{4}$$

Logistic regression is a very simple model, but it is still used to solve many problems. In our particular case, logistic regression could be used to create a simple sentiment classifier, by first pre-processing the data, to then generate a dictionary of frequencies of positive and negative frequencies, which will be the base to apply logistic regression. This approach is very naive because it does not consider many situations where a word, for instance, by context is not being used as positive, even if in a normal situation, that given word is used as positive. Furthermore, words do not act as a simple word always, they can stress or emphasize something, which does not happen in this method. Also, polysemous words that should be deducted by the environment, would be be treated as the same word in this context.

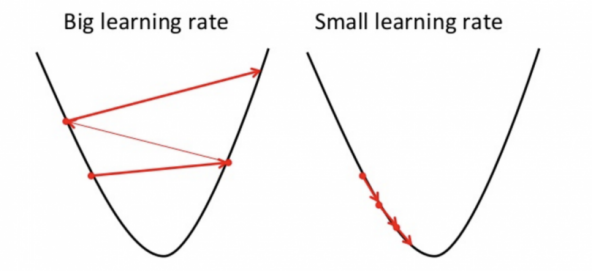


Figure 5: Learning rates on gradient[12]

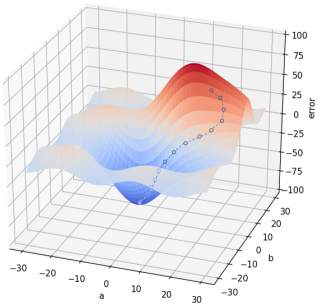


Figure 6: Gradient descent[11]

2.2.2 Naive Bayes

Naive Bayes is a classification technique based on the independence assumption between predictors (Bayes theorem). In other words, this classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature regardless if those features are interdependent, which is naive, since hardly ever is the case (reason why the naive term appears in its name). Naive Bayes, is very similar to Logistic Regression and we can apply it also to sentiment analysis. In this particular case, we would also generate a frequency table where we have the conditional probability per each class and word $P(w_i|class)$. Words that have a similar conditional probability that is more likely than another, are those that will be determining the overall sentiment. This can be seen by calculating the log likelihood.

As aforementioned, the biggest issue is that, in NLP, the independence assumption does not hold in general. For instance, if we think of the example: "It is very sunny and hot in the Sahara desert", it can be seen that Sahara or desert are somehow related to how sunny and hot it can be, which Naive Bayes will not consider. Another issue is that the relative frequencies on the corpus affect the model and that we should balance them artificially, or otherwise they would be affecting our conditional probabilities.

2.2.3 Vector Space Representation

Representing text is not trivial. One very naive solution would be to encode a given word to a number, which would be a step to facilitate its computation. However, when representing words as numbers, all the meaning and similarity behind them would be lost. A better way would be to generate a one-hot-

encoding vector with a given vocabulary. This is a better approach because, now, all words are independent (have their own dimension) from each other. Because all of them are on the same distance, though, there is something that is missing: how do we know that two words are similar? This can be improved.

From the two previous models, we have seen that these approaches do not consider the relation between words and, therefore, can only be used in sentiment analysis, whereas vector space models are able to capture semantic meaning and relationships between words. There are two type of vector space models:

- **Word by Word:** This design counts the number of times words occur within a certain distance k.
- **Word by Doc:** The number of times words from the vocabulary appear in documents that belong to certain categories.

One could wonder what is the power behind vectorizing words and the truth is that there are plenty of advantages in such approach. One of them is that we can apply semantic distance, with, for instance, the cosine similarity, in order to see how close one word is to another. Another advantage is that we can extract patterns in a text by exploiting similarities. For example, if we are given a country such as the United States of America and then its capital, Washington DC; and if we have another country such as Russia, and we would not know the capital, we could compute a vector similar to the USA - Washington, and find that the closest city to that vector, is Moscow.

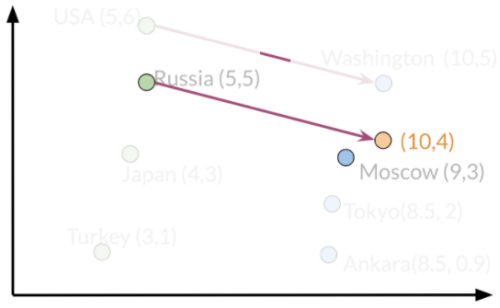


Figure 7: Vectors of USA and Russia

The word embedding process is a model that will require a corpus and an embedding method. The corpus does not need to focus only on our context but rather, general-purpose oriented, which will feed our model once we transform the corpus from words to "vectors" (i.e., integers). The embedding method can be a wide variety of algorithms, but machine learning methods are the most relevant nowadays.

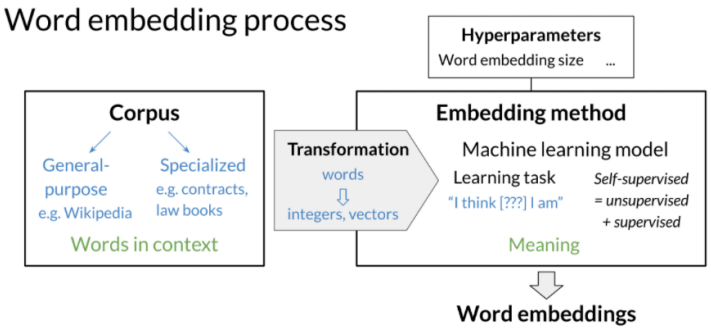


Figure 8: Word embeddings diagram

There are some very well established word embedding methods that can be used to generate word embeddings, even that they are essential, and for instance, one embedding will always have the same meaning:

- **word2vec:** Word2vec, uses a shallow neural network to learn word embeddings. It proposes two model architectures[28]:
  - **Continuous bag-of-words (CBOW)**, which is a simple but efficient approach. This model focuses in learning to predict a missing word given the surrounding words.
  - **Continuous skip-gram / Skip-gram with negative sampling (SGNS)**, which does the reverse of the CBOW method: the model learns to predict the word surrounding a given input word.
- **Global Vectors (GloVe[29]):** which involves factorizing the logarithm of the corpuses word co-occurrence matrix.
- **fastText[30]:** which is based on the skip-gram model and takes into

account the structure of words by representing words as an N-gram of characters. This enables the model to support outer vocabulary words (OVW). FastText can also average its word embeddings in order to make vector representations of phrases and sentences.

Some more advanced word embeddings that are more aware of their context (also known as contextual embeddings) are:

- **ELMo** uses a bidirectional LSTM to feed from right and left.
- **GPT-2** (Generative Pre-training for Transformer): is a Transformer created by open AI with pre training
- **BERT** (Bidirectional Encoder Representations from Transformers) created by the Google AI Language team, is a very famous transformer used for learning text representations.
- **T5** (Text-to-Text Transfer Transformer) is a multi-task transformer which means that a single model can learn to do multiple different tasks.

In this approach (sorted by date), we can see that there is room for polysemy and words with similar meanings. One excellent thing about these models is that we can find these pre-trained models online and fine-tune them with our data.

2.2.4 Neural Networks

2.2.4.1 Continuous Bag-of-words

As we have mentioned earlier, the objective of this model is to predict a missing word (also known as the center word) based on the surrounding words. The idea is that if a similar set of words frequently surrounds two words, then those words have to be semantically related. In order to model and predict a missing word, we need to define some parameters: one of them is the center word that will be the word that we will be predicting. Another parameter will be the context words that are before and after the center word. The number of words that are considered is a hyper-parameter that we can configure. The composition of center words and context words generates the window, which will slide across the corpus.

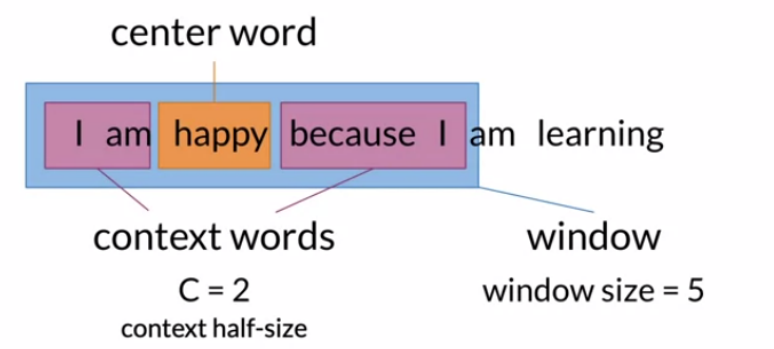


Figure 9: CBOW with window = 5[9]

The CBOW architecture is based on the shallow, dense neural network with an input layer, a single hidden layer, and an output layer. The input of the model is the size of the vocabulary that we have. The hidden layer size will depend on the word embedding size, a hyperparameter, so we will have as many neurons as size has the embedding. W_1 corresponds to the weights matrix between the input and hidden layers, and W_2 corresponds to the weighting matrix between the hidden and the output layers. b_1 and b_2 are the bias vector of the hidden layer and the output vector, respectively. The results of the activation of the hidden layer are what gets sent to the next layer, which in this case is the output layer. Similarly, the outcome of the activation of the output layer is what will be shown as the model's prediction. The rectified linear unit function (ReLU) is used for the hidden layer, whereas, for the output layer, the softmax function is used. To train CBOW, we can do so with gradient descent and on the backpropagation with cross-entropy as a cost function.

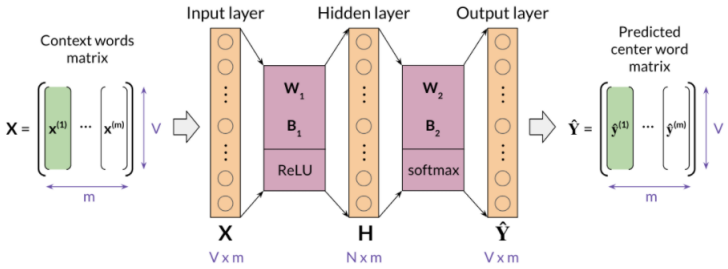


Figure 10: Word embeddings structure[13]

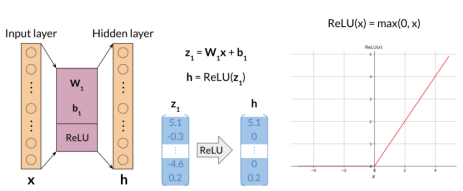


Figure 11: ReLU function

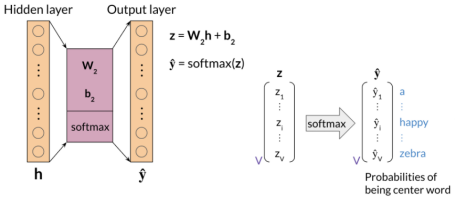


Figure 12: Softmax function

2.2.5 Recurrent Neural Networks

As seen on the vector space representation, vector space models are mostly associated with artificial neural networks and most of those are Recurrent Neural Networks (RNN). This kind of neural networks have the advantage that they are able to propagate information over and over until a prediction is made. RNN can be built with many different types of architectures:

- **One to one:** Given scores from a competition, try to predict the winner. This kind of network is not very useful in RNN since it is the same as a normal NN.
- **One to many:** Given an piece of art, we can predict the author.
- **Many to one:** Predicting the sentiment of a given review.
- **Many to many** (encoder-decoder): translate a sentence from Spanish to English.

This is in contrast to what in NLP is called an N-gram model, which picks the most likely word to fill considering the previous N words. This can lead to results that make no sense, and RNN models address this issue. However, it is not a final solution because RNN's have what is known as the vanishing gradient problem: when there are long term dependencies, RNN are prone to vanish or exploding gradients. This can easily be explained graphically because, as the computation takes more steps, we have more information accumulated; then it is not difficult to imagine that, at a given point, our initial information becomes very diluted and, therefore, useless.

Figure 13: Dilution on RNN's

Some known solutions to the vanishing gradient problem include skipping connections, gradient clipping, or identifying RNN with ReLU activation. However, they are more a patch than an elegant solution.

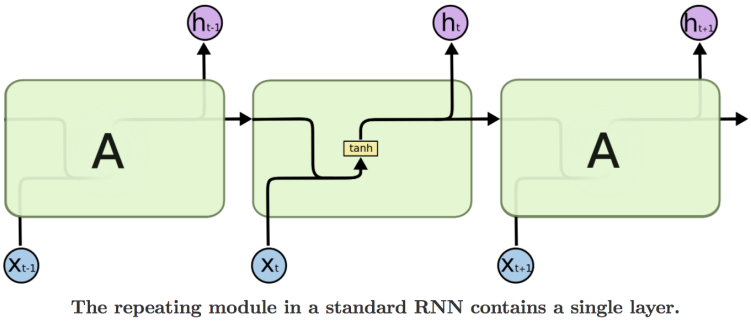


Figure 14: Chained RNN[19]

Vanilla RNN can create other architectures if we stack them. For instance, we can do what is known as a deep RNN, which is better at more complex tasks. We can also have bi-directional RNNs, where each RNN is independent of the other, but we have information flowing from both directions.

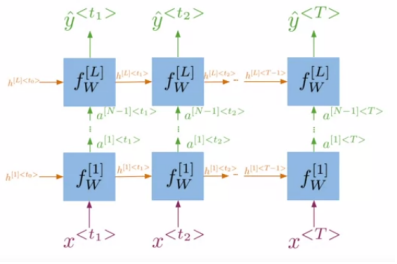


Figure 15: Gradient learning rates

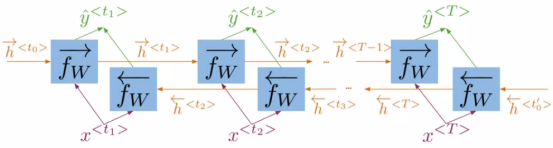


Figure 16: Gradient descent

2.2.5.1 Gated Recurrent Units

As seen for shallow RNN's, there is a tendency to lose information. This happens because we are not able to control de information that propagates. To improve in that sense, architectures like Gated Recurrent Units (GRU)[32] and Long short-term memory (LSTM) were defined. GRU and LSTM, unlike Vanilla RNN, are able to keep relevant information in the hidden state. In the particular case of GRU, this is accomplished by adding a reset and update gate to the architecture: a simplification of LSTM:

- **Reset gate:** determines how to fuse new inputs with the previous memory
- **Update gate:** determines how to fuse new inputs with the previous memory

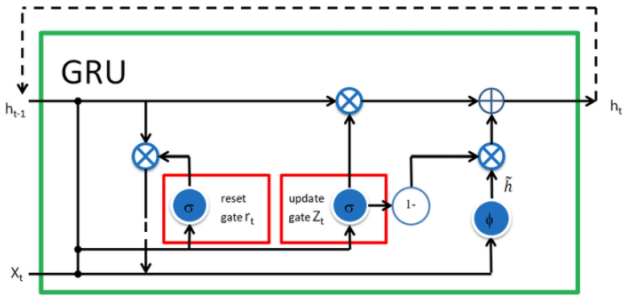


Figure 17: Inside of a cell from a GRU[32]

2.2.5.2 Long Short-term Memory

The Long short-term memory (LSTM) [31] allows neural networks to remember and forget specific inputs in a very similar way to GRU's. Indeed, GRU's are a simplification of LSTM's and using one or another depends on many factors that we will soon discuss. The architecture is composed of a cell state and a hidden state with three gates:

- **Forget gate:** decides which information would be passed threw the cell state.
- **Input gate:** decide which information is going to be stored in cell state
- **Output gate:** decide about what we are going to send in output

If we take by reference the LSTM, we can see that it combines the forget and input gate into the update gate of the GRU. Regarding the utilities of each architecture, GRU's appeared in 2014 while LSTM appeared in 1997. GRU's also appeared to tackle the vanishing gradient problem, and both methods are not exactly tackling the problem in the same fashion. Typically, GRU's are more used when we do not have much data and want to try faster. It is because GRU is more straightforward and has fewer parameters to tune. It also implies that LSTM's tend to be more used for more extensive sequences as, theoretically, they are better at remembering longer sequences.

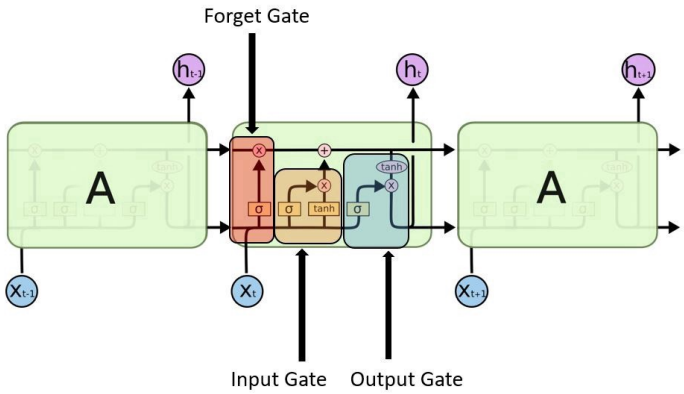


Figure 18: LSTM architecture

2.2.5.3 Transformers

As we have just seen RNN, has limitations, and even that they have been the state-of-the-art machine learning model for NLP, the transformer model surpasses their limitations and has been established as the reference.

It is not because RNN's do not work well in many situations but rather because of one essential issue: RNN are sequential, and the way that they are designed leaves little to no space for parallelization, which together with the vanishing gradient problem, represent a threat for significant amounts of data processing. Therefore, the transformers model proposes using attention in the architecture together with word embeddings and the encoder-decoder archi-

tecture. The addition of attention to the architecture is key and surpasses RNN's and their performance. Attention can come into play in different ways:

- **Encode/decoder attention:** One sentence that is the decoder looks at the other one (encoder). For instance, it could be used when translating sentences from one language to another one.
- **Causal self-attention :** in one sentence, words look at previous words that appeared. This kind of attention, for instance, could be used for text generation.
- **Bidirectional self-attention :** where words in the same sentence look both at previous and future words. This type of attention is used in BERT and T5.

However, in order to achieve great results with transformers, what we need is to use multi-head attention. Multi-head attention basically proposes multiple heads where each head can learn different word relationships and its strongest point: the ability to achieve multiple look-ups in parallel.

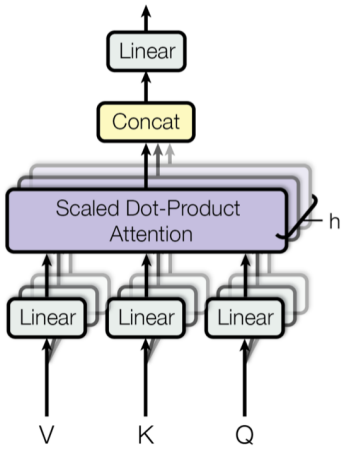


Figure 19: Multi-head schema[55]

There have been multiple transformers, but the ones that we are going to focus on and put more emphasis on are GPT-2, BERT, and T5.

GPT-2 is unidirectional, unlike ELMo, which had struggles in capturing dependencies for long sequences. It is not bidirectional, so we do not have

peaking, which is a problem in auto-completing missing words since it lacks context. It is when BERT comes to play, with multi-mask modeling and next sentence prediction. T5 uses multitask, which implies that we do not need, for instance, to create different models for different tasks.

Figure 20: Multi-head schema

2.2.5.3.1 GPT-2 The Generative Pre-Trained Transformers[34] (GPT) model is the simplest of the transformers we will talk about. Currently, there are newer versions with GPT-3, but we will focus on GPT-2. This model is a transformer decoder architecture, so we do not have an encoder, only a decoder. Below, we can see a diagram of the architecture, which we will explain in depth.

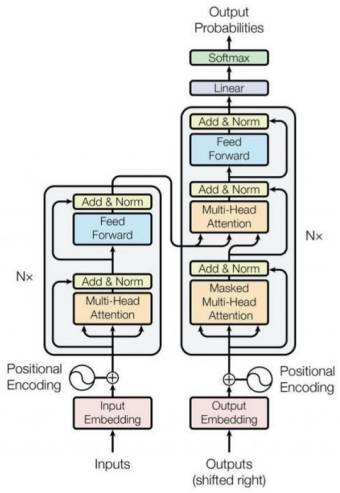


Figure 1: The Transformer - model architecture.

Figure 21: GPT-2 architecture

As we can see in GPT-2, first we get as input the text. Then we transform this text into embeddings, and we apply positional encoding to them. Then

we apply multi-head attention. Once applied the multi-head attention, there is a feedforward layer with a ReLU. Finally, we have a residual connection with layer normalization (which will be repeated N times). Last, we finally a linear layer with a softmax.

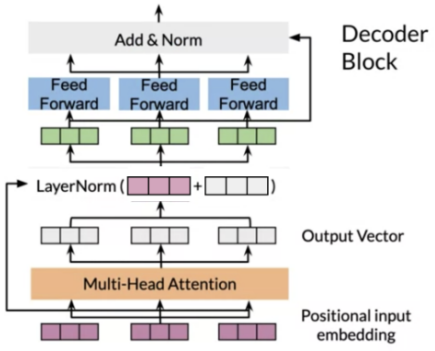


Figure 22: Zoom of the GPT-2 model

2.2.5.3.2 BERT Bidirectional Encoder Representations from Transformers[55] (BERT), as we mentioned earlier, was created by the Google AI Language team, and it is widespread in text representations. What made BERT revolutionize the NLP world was that researchers tried to look at a text sequence from left to right or combined left-to-right and right-to-left training until that moment. However, no one applied bidirectional training on a transformer. The results allow the model to have a more profound sense of context and flow than single-direction models. All this is possible thanks to a technique applied on BERT, named Masked LM (MLM), responsible for allowing bidirectional training.

There are two steps in the BERT framework: pre-training and fine-tuning. During pre-training, the model is trained on unlabeled data over different pre-training tasks. For fine-tuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks.

BERT reads all the input at once, unlike directional models that would sequentially read the input. Looking at the previous schema of architectures, we can see that BERT is only an encoder. It might be surprising since transformers in their vanilla form have an encoder that reads the input and a

decoder that is responsible for producing an output which, in this case, is the prediction. In the case of BERT, we do not need a decoder since the goal of this model is to generate a language model.

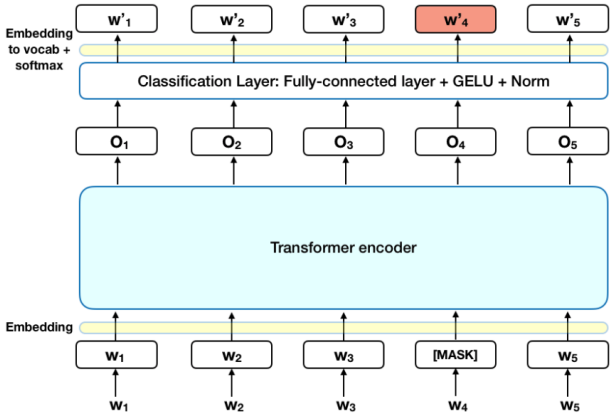


Figure 23: Transformer encoder chart

In the diagram above, we can see that the input is a sequence of tokens embedded into vectors and then processed in the network. The output is a sequence of vectors of size H, in which each vector corresponds to an input token with the same index. Finally, as we mentioned, BERT uses two training strategies to define the next word to predict: one is MLM which we mentioned earlier, and the other is next sentence prediction (NSP):

- **MLM:** Before feeding word sequences into BERT, 15% of the words are selected. Then among those words, we either mask them 80% of the time, keep as is 10% of the time or replace them with a random token 10% of the time. Finally, we calculate the loss with the cross-entropy function.
- **NSP:** In the BERT training process, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50%, a random sentence from the corpus is chosen as the second sentence. The assumption is that the random sentence will be disconnected from the first sentence. To help the model distinguish between the two sentences

in training, the input is processed in the following way before entering the model:

- **The input embeddings:** we have a [CLS] and a [SEP] token to indicate the beginning and the end of the sentence.
- **Sentence embedding:** indicates to which sentences belong. To do so, we add "Sentence A" or "Sentence B" to each token.
- **Positional embedding:** is added to each token to indicate the position in the sentence.

In order to predict if sentence A is related to sentence B, NSP does the following:

- The entire input sequence goes through the Transformer model.
- The output of the [CLS] token is transformed into a 2×1 shaped vector, using a simple classification layer.
- Calculating the probability of IsNextSequence with softmax.

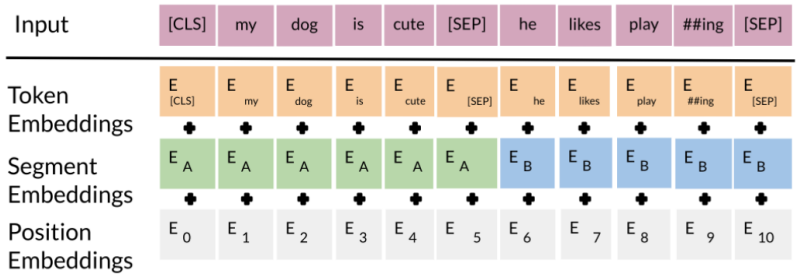


Figure 24: Input processing on BERT

Finally, once we have pre-trained and minimized the combined loss function of NSP and MLM, we need to fine-tune it. In this particular case, fine-tuning will require us to make the model adapt to the specific task that has to solve. Therefore, it will most likely imply adding an outer layer.

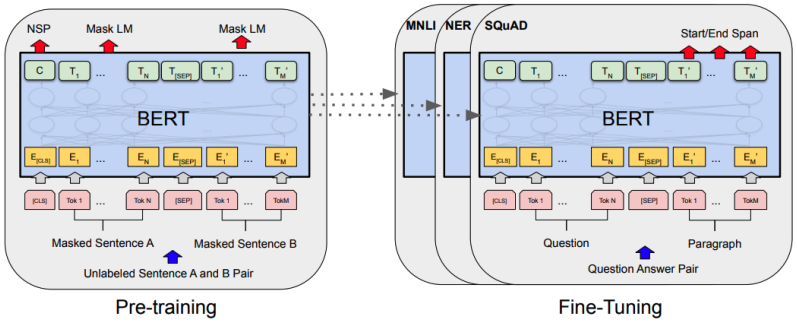


Figure 25: Example on fine tuning BERT on a question answer model

2.2.5.3.3 T5 The most vital point of this model is the capacity of multi-tasking with the same model: machine translation, classification tasks, regression, and others. It can be done thanks to encoding the different tasks as text directives in the input stream, which enables a single model to be trained on supervised NLP tasks.

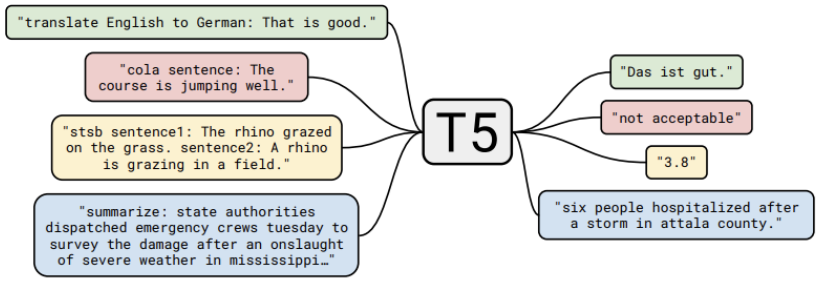


Figure 26: T5 example of tasks [22]

Text-to-Text Transfer Transformer[22] model has some resemblances to the BERT architecture. It makes use of transfer learning and mask language modeling. However, in this particular case, we are not having an encoder architecture but rather a decoder and encoder architecture. The main contributions on the T5 are not so much its architecture but rather its findings by using the T5 model to examine factors relevant for leveraging transfer learning at scale from pure unsupervised pre-training to supervised tasks.

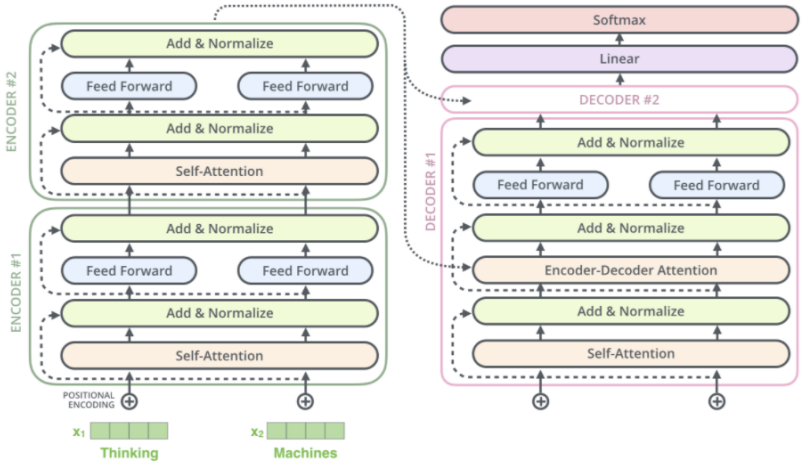


Figure 27: T5 model structure

Some of these findings are the following:

- MLM performed better than the language modeling objective.
- Pre-training is best done on large and diverse datasets as opposed to small ones.
- Updating all of the pre-trained model parameters during fine-tuning yielded better performance than updating only a few, despite the computational cost.
- Scaling by training the model on more data, training a larger model, or using an ensemble of methods improved performance.
- The knowledge extraction capability by pre-training with the masking to extract knowledge seems constrained by what the model learns from just predicting corrupted spans of text, which suggests, as mentioned earlier, learning approaches that go beyond just text.

2.3 Result Evaluation

Being able to evaluate our results appropriately, it is essential to keep track and allow us to iterate quickly on the obtained results. Our output will be a summary, but how do we know if a summary is representative or not of the

reality? It is a non-trivial question that it is, as we mentioned, essential for us to keep track of our progress. There are many different criteria and different aspects that we can track, such as length, fidelity, salience, grammatically, non-redundancy, referential well-formedness, and structure plus coherence.

In our particular case, we are ideally looking for the shortest summary possible in the least space. So what we are more interested in is having a bigger retention ratio than the compression ratio. There are two kinds of evaluation techniques:

- **Extrinsic:** Were there is an evaluation regarding how a task can be done equally on the full or the summarized version of the input. For instance, we could think of the Titanic as the text to summarize where we will ask some questions regarding the incident to the users that read the full text and the same ones to the users that read only the summary. In this case, we will be able to see if we are, for instance, missing details that our summary does not cover. So basically, extrinsic techniques aim to show if the same decision can be made using the summarized and original versions, but in less time.
- **Intrinsic:** Were we are going to compare our summaries against gold standards or another summary. In this technique, we also require examples to see how our output is compared to that standard.

In our case, we will focus on intrinsic techniques and evaluate a set of users on how good or bad our summaries are. Therefore, if we look at the different intrinsic techniques, we can find many. For example, we can find Recall-Oriented Understudy for Gisting Evaluation (ROUGE[24]). It is a widely adopted methodology in the NLP field since it is very convenient for prototyping. Furthermore, ROUGE is particularly useful because it expects to have more than a possible good summary. Therefore, we expect our summarization system not to be better at a particular type of summaries from a given person but rather to be similar to what the different gold standards agree. Among the different ROUGE variants, we can have:

- **ROUGE-N:** which is a measure of n-gram overlap between a summary and a set of reference summaries.
- **ROUGE-L:** which uses the longest common subsequence overlap between a summary and a set of references.

- **ROUGE-W:** which considers weighted longest common subsequence. It aims to benefit consecutive matches.
- **ROUGE-S:** which uses skip-bigram recall metric.
- **ROUGE-SU:** adds unigrams to ROUGE-S

We will also use ROUGE because it has been shown that it correlates with manual evaluations when averaged if there are many examples.

However, ROUGE is not the only metric that can be used. There are other metrics such as Relative Utility[25] or Pyramid.

- **Relative utility:** in a very summarized description, does provide a weight (importance) to a given phrase. It has an advantage over systems that use precision and recall, such as ROUGE since maybe our system is doing a good summary. However, one of the least relevant phrases is missing and instead includes another not-so-relevant phrase that is almost as important as the other. Well, in this particular case, the algorithm would not penalize as harshly as ROUGE.
- **Pyramid:[26]** very similar to Relative utility. Specially thought for multi-document inputs. In this particular case, Pyramid weights phrases by the times that appear on the text, and in the summary that we generate, has them, they get a score, the number of times that phrase appeared. The best summary is the one that has the highest score.

As we can see for the evaluation methods, we need someone who provides a summary and an opinion as a reference. However, we do not have that summary; instead, we will take the SuperGLUE benchmark and apply one of the leading models as if a human would have done them so that we have that reference.

2.4 Methodology

To choose a methodology, we might first have to think about what a methodology is. A methodology is a research strategy that outlines the way that research has to be undertaken and, among other things, identifies the methods to be used in it[**?**]. Therefore, we can see that many different methodologies will be more or less suitable for a given project.

Suppose we emphasize methodologies in the data science field. In that case, we can see that they all follow the scientific cycle of observing, researching in a topic area, hypothesizing, testing with experiments, analyzing the results, report the results to begin again with the cycle.

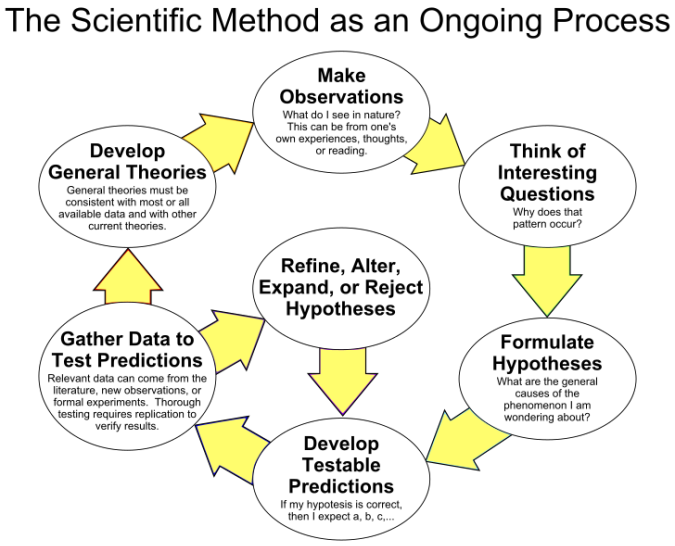


Figure 28: Scientific Methodology[17]

However, we need something a bit more specific in our field. If we look for methodologies applied in the Data Science or the NLP field, we can find that surprisingly there are not many available. Mainly we can find two methodologies: Cross Industry Standard Process for Data Mining (CRISP-DM) and IBM Data Science Methodology. Among the two, we can see that CRISP-DM is way more detailed and provides a more robust methodology than the IBM one. However, we can see that they are indeed very similar, just that IBM did not work on the details and CRISP-DM did. If we look at the CRISP-DM diagram, we can see that it is divided into six significant phases, which are iterative. Those phases are business understanding, data understanding, data preparation, modeling, evaluation, and deployment. There are some phrases that in this project will not be given as much importance as others. Those two phases will be business understanding and deployment. These two phases are industrial-based, and we are working on a research project where these two questions are not relevant or maybe even out of our control.

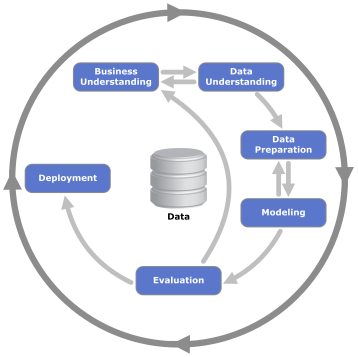


Figure 29: CRISP-DM schema[20]

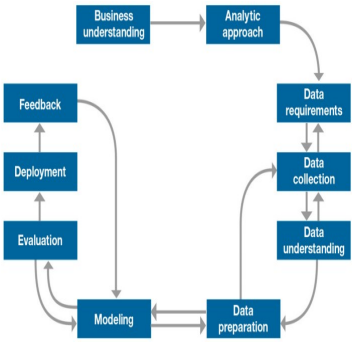


Figure 30: IBM schema[21]

3 A Proposal

This project starts by building a system that includes sentiment analysis and multi-document extractive text summarization on property reviews. In order to achieve so, first, we will need to crawl the data, which in this particular case, will be from TripAdvisor and StayForLong. The next phase is going to be the pre-processing and EDA phases of the dataset. The objective is to understand the data and create new variables to improve the modeling phase. Finally, the project finalizes by creating a binary sentiment analysis that helps determine if each review is either positive or negative. Once this review is determined, we will finally create a summary of those positive and those adverse. Therefore, our output will be two summaries: a positive and a negative one so that the stakeholders can see the good and the wrong things.

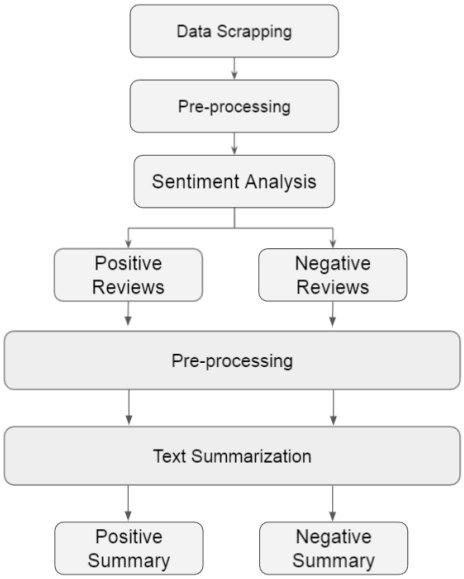


Figure 31: Schema of our proposal

The sentiment analysis and text summarization blocks could have been worked with several alternatives proposed on the state-of-the-art. However, BERT will be used for the sentiment analysis and GPT-2 for the text summarization. It is because of several reasons:

- Dealing with several approaches will diminish our capacity of tuning our model at its best.
- From the sentiment analysis side, it is known that BERT outperforms the other models we studied in terms of capacity to understand because of its implementation. However, one architecture cannot be the best at everything, and even if some alternatives have been discovered since BERT appeared, the masked LM that provides the deep bi-directionality removes at the same time the probability distribution over a sentence. Therefore, we are going to be using GPT-2, which performs well in text generation.
- At the beginning, we planned to work with several models simultaneously and compare them, but there were so many options that the

project would not have a clear objective. It is most likely because our initial research question was a bit too wide. However, once we learned more on the topic, we thought it was wise to change and narrow it down to a single model.

4 Project

4.1 Data

We need hotel reviews so that we can extract those opinions. Currently, leading websites such as Booking or TripAdvisor offer API for their clients. However, this information is not available in cases of research nor analysis. Therefore, we will need to extract this information from their websites. In order to do so, we are going to extract it using python with Scrapy, a web crawling and web scraping framework used to crawl websites and extract structured data from their pages.

We have decided that at least for the early stages of the crawling process, we will work exclusively on TripAdvisor reviews. The reasons are mostly business requirements because basically Booking is a competitor from StayforLong whereas TripAdvisor is not. Moreover, StayForLong is already using the comments from TripAdvisor its website, which aligns with the information we display.

4.1.1 Scrapping the Data

As we mentioned, we need first to get the Hotels data from StayForLong to extract the reviews from those Hotels on TripAdvisor. Therefore, we will first access StayForLong API calls that we can see from the inspector in our navigator as XHR calls. There are many different ways that we can obtain the same information, but we are going to choose crawl by properties `https://gw.stayforlong.com/properties/ids?id_properties={number}&lang=en`. Mainly because the information that location provides is not complete enough, as can be seen in `https://gw.stayforlong.com/reviews/location/{number}?lang=es&currency=EUR`, for instance, each hotel has a different location_id, and we would also get a direct link to 3 comments. However, we only get access to these three comments because, as we mentioned already, StayForLong has access to the API of TripAdvisor. Therefore, the API allows them to showcase only three comments to help users choose. However, we are missing information such as the name of the hotel.

Therefore, we are going to focus on properties, which follow the already mentioned structure of `https://gw.stayforlong.com/properties/ids?id_properties={number}&lang=en`. Once we see that, we can see the information from the

StayForLong API, which provides us with information such as id_property, name, country, accomodation_type, latitude, longitude, address, reviews, id_tripadvisor, and more.

As we can see, there is much information but, the most relevant information that we can extract is the **id_tripadvisor** because, as mentioned earlier, we want to extract the reviews from TripAdvisor.

How can we from this ID get the comments if we do not have API access because it is restricted? Well, if we check the XHR calls from TripAdvisor, we will not find anything relevant. However, suppose we check for a given property. In that case, we can see that their link constructs like the following: https://www.tripadvisor.in/Hotel_Review-g1463492-d2161434-Reviews-Bellissimo_Boutique_Hotel-Buccoo_Tobago_Trinidad_and_Tobago.html. One might think that with the information we had from the API, we do not have enough, but the truth is that we do since, the previous link it is equivalent to https://www.tripadvisor.in/Hotel_Review-d2161434 and the particle "d2161434" is d + id_tripadvisor that we got from the StayForLong API. The city location id is the number after the g, which in our example is 1463492. The city's name is the string between the location id and the dash before "Hotels." So, for instance, we could also crawl all the information of an exclusive area such as Barcelona. However, as we mentioned earlier, we are more interested in providing feedback per all the locations on the platform. However, we thought it was interesting to give the insight if someone is interested in a similar exercise applied to a city.

Now, it is time to retrieve the comments using BeautifulSoup, which is a Python parsing library. Scrapy or Selenium will not be used as a crawler because the selected library empowers us to do everything needed. BeautifulSoup is a compelling library that covers with a more straightforward learning curve what we want to do.

4.1.1.1 First Iteration

Therefore, what we first needed to do, is to identify which attributes can identify the content that we want to extract from each review, and we chose to pick the title, the description, and date. We think that for at least the first iteration, it is what we need. In case of requiring more data, we are going to

crawl it later.

Figure 32: Example of a comment on TripAdvisor

Also, as seen on the image, there are cases where the review is longer and will keep when we click the "Read More" button. In those cases, we are also extracting the data since it is also on the HTML.

Once we know how to parse data from a page, we need to understand how a link changes when pagination occurs. Because, for instance, TripAdvisor only provides five reviews per page, and we expect to have way more reviews than those (even that we will only work on the English reviews). Therefore, we will retrieve from TripAdvisor the number of English reviews.

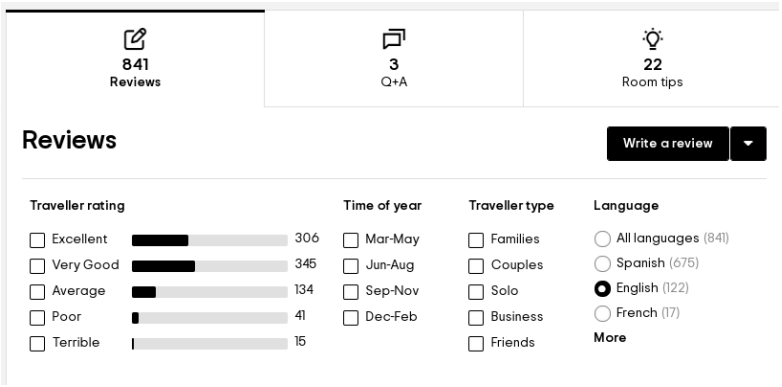


Figure 33: Filter box on TripAdvisor

Once we know the number of reviews we have, we need to generate the links we need to parse with BeautifulSoup. To do so, we need to understand how pagination works. If we take as reference the example we mentioned earlier:

https://www.tripadvisor.in/Hotel_Review-g1463492-d2161434-Reviews-Bellissimo_Boutique_Hotel-Buccoo_Tobago_Trinidad_and_Tobago.html

This link will keep the structure during the first five reviews, but afterward, it will change by adding the "-or{number}" term between "Reviews" and the name of the property. In this case, the link would change like this:

https://www.tripadvisor.in/Hotel_Review-g319797-d317670-Reviews-or5-Parador_de_Turismo_de_Benavente-Benavente_Province_of_Zamora_Castile_and_Leon.html

Furthermore, the link can be shortened since TripAdvisor makes a more extended link to facilitate SEO; the previous link can be shortened to:

https://www.tripadvisor.in/Hotel_Review-g319797-d317670-Reviews-or5

This structure maintains until the number of reviews we have. Therefore, we only need to generate the total number of links to parse the number of English reviews divided by 5.

During this first iteration, we encountered some issues that we consider worth mentioning:

- Scrapping data from any website can be problematic. Websites pay for their services, and for instance, if we call those services many times and in a brief period, we cost them money. Furthermore, data is a massive resource by itself; allowing people to scrap data quickly should be avoided at all costs. For this reason, websites tend to ban IP if a user mass consults their services. To avoid this situation, we have done adding timers to our code that randomly fluctuate. By doing this, we avoided being prohibited from making our requests. However, our code takes much more time. It is a problem because the amount of data that can be collected turns into a bottleneck. Moreover, some other problems arise, such as data heterogeneity which can lead the scrapping process to fail and stop. To avoid these situations, we had to cover the Date field not appearing, a property not having any English review, a property from StayForLong not having a TripAdvisor Id, etc. We have also switched from working in batches of 5 properties to writing the

JSON file and then writing each file in a separate JSON and merging them all. It was done to avoid memory leaks. We chose JSON because it is the best format not to work, but for storing the information and can also be joined quickly on a single JSON file. Although we did it manually in this first iteration, on the second iteration, we did it automatically.

- As we mentioned previously, adding the timers to make requests caused our code to take much time. Therefore, we opted to add a limitation in the reviews from a given property because, for instance, there were hotels with a considerable amount of reviews, e.g., 7000 reviews. Therefore, it will not take more than 600 reviews from a given property in our particular case. It seems a number that works well for us in terms of information acquired and retrieval time.
- Finally, at the beginning of the project, we considered working with 1000 properties. However, because it was time-consuming to generate the dataset, we finally opted to work with 250 properties with a maximum of 600 reviews each, which constituted a total of 67000 reviews.
- We had to be careful with the format we use and how we access information. We chose to work with UTF-8.

It is also worth mentioning this particular way of scrapping, and in general, scrapping. Has the issue that a user always has to pay attention to the website they are scrapping, since the format of representing the data is subtle to change, also more data will be aggregated frequently. We faced this problem once we saw that we needed some additional data to be collected which initially, was not in our schema. After thinking for a long time, we decided that the best solution was to re-run the script with some corrections. It was challenging and computationally expensive to scrap some information and attach it or correct some errors. It was just better to re-run (which this time was done in 3 different computers to speed up the process).

4.1.1.2 Second Iteration

As we mentioned earlier, we used to retrieve only the title, the text of the review, and the date of each review. We also set a boundary that when there are more than 600 reviews on a given property, we take 600. We encountered

some issues later, and we had to do the scrapping process again for that same reason. Below are listed the reasons supporting our decision.

- There was a big difference between the number of reviews between properties. As we will see later, the scrapping process starts with a given ID on StayForLong from properties in the inner part of Spain that are not so visited and less for foreign people. It causes some properties to have a small range of reviews, such as 5-25 reviews. In some cases, reviews were also ancient. Which shows how "unpopular" those properties could be. Therefore, we set a minimum of 60 reviews so that properties that we will be working on are somehow significant.
- To work with sentiment analysis, we decided later on that it was better to do it as a supervised method. Therefore, we needed some data that would help as a tag during the learning process.
- Some of the old reviews have the "Read More" tab as any review, but the HTML structure is not the same as it would be for recent ones. Therefore, it causes that some comments (very few) were not displayed completely. To solve this, we discard those old comments that meet this condition.

As a result, we obtained fewer properties, with 176 over the 350 IDs we tried. However, the total amount of reviews is higher with 80000 reviews (450 reviews on average), and the quality has also improved since we corrected some of the issues above.

4.2 Data Understanding

This step is heavily related to the previous one because in our case, we have been mixing both steps, since in order to verify that our scrapping process works, we had to understand the data well, so that the crawling process can run smoothly during hours, without having an unexpected error or being erroneous and having to be repeated.

First step we need to do is to join the different JSON files that we were partially generating while running the scrapping. Once we have all the data gathered, what we are going to do is to see which variables we have gathered so far:

Variable	Description
id_property	Internal id from StarForLong for the property.
name	Name of the property.
country	Acronym from the country where the property is located.
slug	Url structure of the property
accommodation_type	Type of accommodation such as HOTEL, HOSTEL, RURAL, APTHOTEL, APARTMENT, HOMES
latitude	Latitude position from the property
longitude	Longitude position from the property
address	Address of the property
city	Name of the city property
stars	Number of stars that the property has
distance	Distance from the center
in_center	Boolean, can be true or false
images	Group of images from the property
ratings	Table related to the ratings
city_slug	Url structure of the city
rating	Numeric value corresponding to the ratings
facilities	Facilities table, array of items, each item has code_group, slug, name, id
rating_count	Number of ratings

Table 1: Properties Table

Variable	Description
id_tripadvisor	Internal ID from TripAdvisor that is assigned to the property.
rating	Numerical score corresponding to the total ratings
ranking_out_of	Total number of properties on the ranking
ranking	Position in the ranking of properties
review_rating_count	Division of the different ratings such as 6 ratings of 1 star, etc
subratings	Table for different aspects such as location, sleep, rooms, service, value, cleanliness.
trip_types	Table which numerates the different kind of reviews present
rating_count	Number of ratings
reviews	Table of reviews which contain per review: title, date and review.

Table 2: Ratings Table

As we can see, much of the information that StayForLong had stored per each property that we did not filter at first is irrelevant for us. Also, it is not precisely the one that our corpus contains since our corpus is a subset of the reviews of those properties. That information is a summary of the total of reviews. For instance, not all properties StayForLong IDs have a property on them. We removed those properties from our set in this particular case because we do not have data from them. Since the API returns an empty set. So in the very end, we did not get a straight set of IDs.

The dataset obtained with the crawling process is considerable and significant enough: we have 80000 reviews, which makes an average of approximately 454 reviews per property.

We can see different types of accommodation, and one could wonder: is the thesis exclusive to hotels? The answer to that question is negative. We did not limit to only hotels, we took any property because the problem to solve it is the same, but we could have done it quickly. If we see that it is a problem, we will create a new dataset with only Hotels.

There is some data that we already foresee that it will not give any important information to our dataset, so we are going to drop it beforehand. The data that we are going to drop is the address, facilities table, city_slug, slug, images, rating_count, review_rating_count table. So our base dataset will look as follows:

Variable	**Description**
id_property	Internal id from StarForLong for the property.
name	Name of the property.
country	Acronym from the country where the property is located.
accommodation_type	Type of accommodation such as HOTEL, HOSTEL, RURAL, APTHOTEL, APARTMENT, HOMES
latitude	Latitude position from the property
longitude	Longitude position from the property
city	Name of the city property
stars	Number of stars that the property has
distance	Distance from the center
in_center	Boolean, can be true or false
ratings	Table related to the ratings
rating	Numeric value corresponding to the ratings

Table 3: Dataset Table

4.2.1 General overview of non-review data

In the dataset that we selected because of how StayForLong has their data internally, we got that all our locations are in Spain. It somehow explains that some properties have few reviews in English because inner cities are less attractive for an international tourist but more interesting for a national type of tourism, which makes those hotels have more reviews in Spanish than English.

Figure 34: Map with the locations of properties

Regarding the number of stars that our selected properties have, we have the following distribution of stars. We also have, on the right-hand side, the distribution of ratings that our properties received. Where we can see that the most common properties are between 3 and 4 stars, we can also see some properties without reviews; as we mentioned earlier in the project, we are not exclusively dealing with hotels, so it is a situation that can happen. Instead of paying attention to a property's ratings, we focus on the reviews received; we can see that 70% of the ratings are between 4 and 5 stars. Therefore, we are going to consider reviews under four stars as negative reviews.

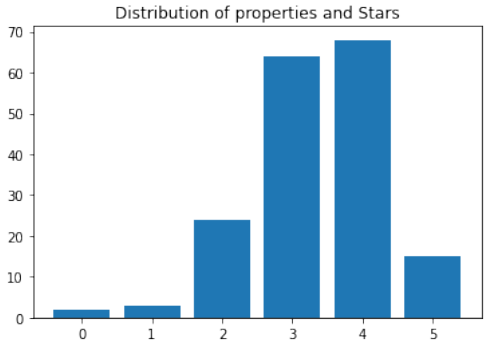


Figure 35: Distribution of assigned stars

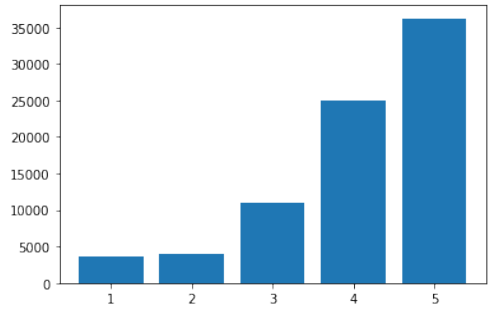


Figure 36: Distribution of ratings we have in our dataset

We can also see that most of the properties are outside the center of their

locality, and regarding the accommodation types, we mostly have hotels followed by apartments. However, more than half of our properties are Hotels. With the second iteration on the dataset, we lost accommodations such as hostels and other types also reduced a lot.

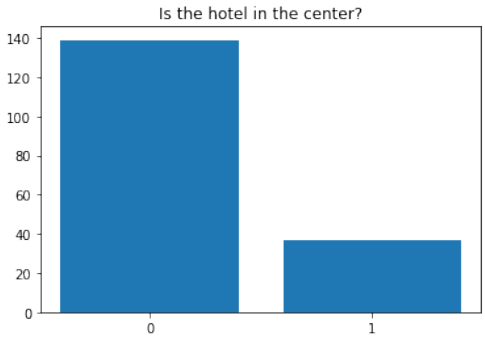


Figure 37: no-center vs center

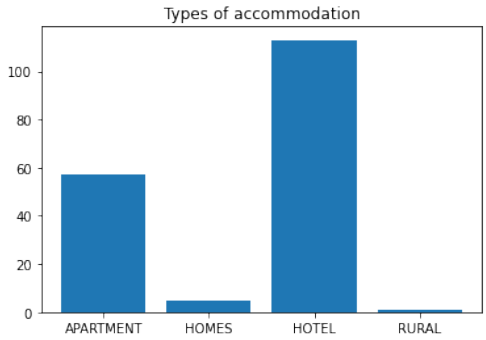


Figure 38: Types of accommodation

We have five categories regarding the trip types: business, couples, solo, family, and friends. If we do an aggregation of how many of each we have, we can see that the most common is couples and family, so basically: leisure. However, this information is not precisely from our dataset and more from the overall reviews of the properties we treat. We are dealing only with a subset of the reviews, and this information is global information that comes from the TripAdvisor API to StayForLong. It happens with several data that helps us have an idea but cannot be much of use.

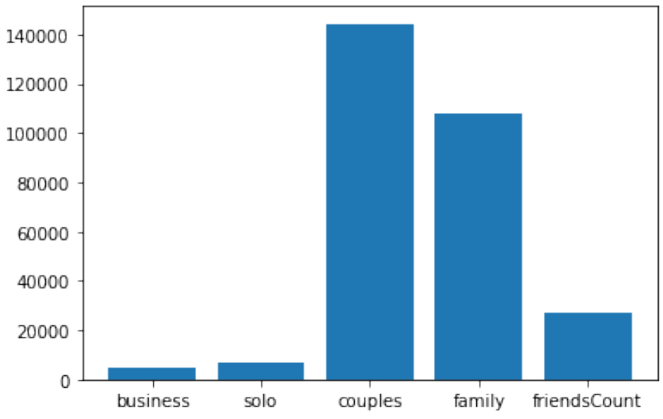
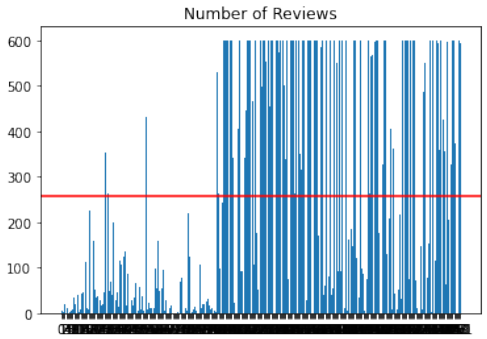


Figure 39: Type of travels of the properties in the dataset

4.2.2 Reviews

Data across properties can change depending on the accommodation type, location, stars of the property, etc. In our particular case, when we scrapped the properties, we limited the number of comments that we could get from a given property to 600 reviews. Later, we decided to set also a minimum boundary. As we can see, this helped to homogenize the kind of properties that we are dealing with. Because as we can see on the left, there were properties with way fewer reviews than those.



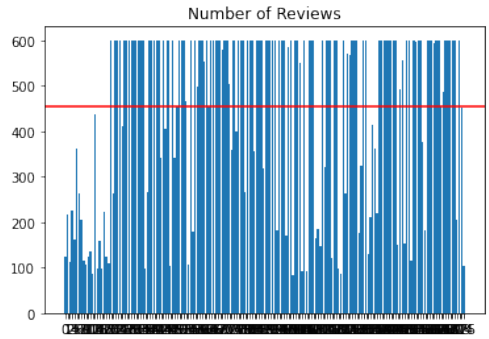


Figure 40: Number of words on titles Figure 41: Number of words on reviews

This shows us that there are very popular hotels and that some others are not even close to as popular as those others. Of course, it could be due to many factors. One of them could be the location of each of them, their stars, type of accommodation, etc. But basically, it is strongly related to the location.

Regarding the length of reviews, we calculated the average words per review and the titles of each property. Thus, in general, we can see that titles tend to have four words, and reviews average 110 words.

If we put attention to the number of characters instead, we see that the results are very similar:

We see that, in general, a word used in our dataset has 5-6 characters. Something a bit surprising is that we can see that those reviews whose body is shorter tend to have long titles.

We can also see the top 100 most common words in a given comment by plotting them in wordclouds (which print words in a cloud, where the most

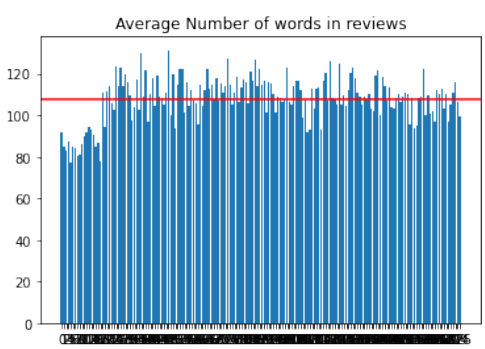


Figure 42: Number of words on reviews

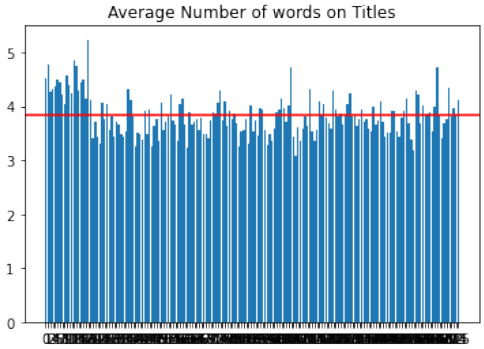


Figure 43: Number of words on titles

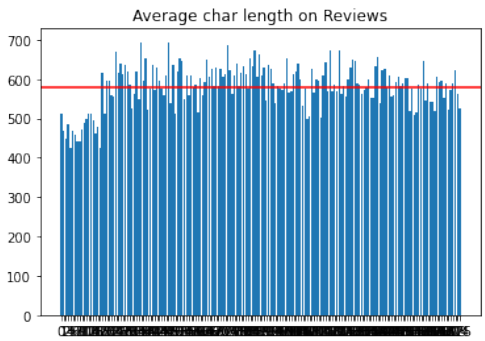


Figure 44: Charac. length on reviews

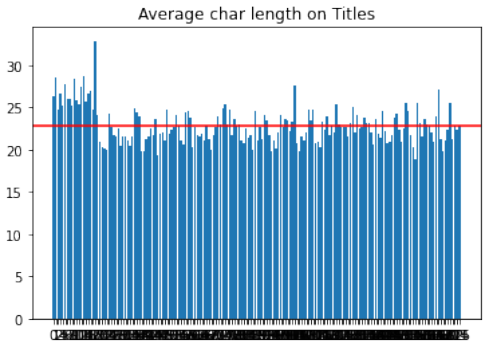


Figure 45: Chars title length

frequent words, appear bigger). We will go more in depth in this aspects, on the Exploratory Data Analysis section.

4.3 Data pre-processing and preparation

This section will focus on the manipulations that have to be done on the reviews from our scrapping process. It is an essential step because, in order to be able to implement our models, we need our data to be clean, simple, and with a particular structure. Since we will work with BERT and GPT-2, we will change the tokenization method both times that we will apply this process so that we meet the requirements of the architectures.

Figure 46: Top 100 words in our dataset

4.3.1 data pre-processing for sentiment analysis

Data pre-processing in NLP is a complex step as dealing with other unstructured data tends to be. It is first because of how complex the data we deal with is. Secondly, because we are juggling between simplicity and information, simplicity will help us gain insight, perspective and better understand what we are doing. However, we might be losing details that would help our model perform way better. Without going further, there are many times that one decides to lowercase all the text in the pre-processing steps. Which sometimes will lead to a loss of context. Therefore, having a simple and more interpretative model is more important than an almost insignificant improvement. Nevertheless, we thought that it is vital that we always consider this trade-off during our pre-processing, and all we have done is justified under the 'keep it simple' frame.

In our particular case, our text will follow the below pipeline of modifications that we will explain in each section. Some of these operations could be performed differently, but there are some restrictions between operations. Without going further, if applied late in our pipeline, the text correction starts to behave worse (which makes sense since it will have less contextual information).

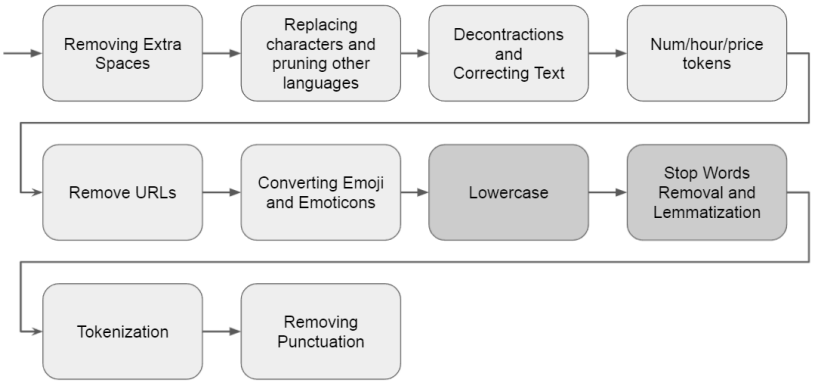


Figure 47: Text processing pipeline

There are two steps (lowercase and stop words + lemmatization) that we are going to apply, but we will also make a version that will not consider them (so we will have three versions). It is because we want to see if it helps to improve the result, or it does not. We think that it will help for the sentiment analysis.

However, we want to emphasize that getting an immaculate text is impossible. There are very contextual specific challenges, and many dictionaries can be customized to gain that bit more accuracy. For example, without going further in our domain, people will tend to name the hotel or use a Spanish term (remember, our properties are located in Spain). It could be patched by adding more words to a dictionary, translating them, etc. Nevertheless, even we did all that, there would still be scenarios that we would only be able to see if we read all our comments from typos that a corrector system will not see as what they should be corrected, to words that are just way too badly written to be understood, etc. Therefore, we need to work with domain knowledge but not overfit our approach to our corpus because we will always have this "humanization" on our corpus that will have to be ignored.

Once mentioned, in our particular case, we have applied the following transformations to our data:

- **Eliminating URLs:** They do not add any meaning to our text. We are doing this by using regular expressions. However, there are also instances where the users do not include a URL but the branding

name. One example would be, instead of saying: "We booked through https://www.booking.com/", they mention "booking.com". Common sense can happen, but most libraries do not cover this case, and in terms of regex, it is not easy to generalize. Because we can find scenarios where a phrase does not include a space once finished the sentence, it complicates the regex. We have created a dictionary with websites that are most likely included in comments, together with all possible domain terminations so that we can do a regex with that.

- **Pruning other languages comments:** Another one of these "humanization" problems and that also comes from the TripAdvisor site, is that there are some comments that are bilingual. They usually work like this, first explained in English and later explained slightly differently to their fellow citizens with detailed advice. This issue is complicated because we can have people who write in any language and English. However, we can find that this issue mostly happens with people that write in other alphabets such as Cyrillic or Hangul. To solve this, we will do later on our pipeline to stripe all characters that are not ASCII letters.

- **Converting emojis and emoticons to text:** In this particular case, we decided not to eliminate emojis and emoticons because we believe that they add meaning to a sentence. Especially in the form of sentiment. However, keeping them as what they are, is not the best solution either. So, what we are going to do is transform them into the text to get better interpretability. To achieve so, we will be using the emoji library[16] that basically, has a big subset of emojis included so we are not going to miss emojis or emoticons which, are not so easy to remove with regex expressions due to its variability, in both aspects: lexically and as Unicode characters.

- **Removing extra white spaces:** We are going to remove extra white spaces and in general all kinds of spaces that are misplaced in the text. It is done to keep the text as light as possible, avoid redundancies, and help when applying the tokenization process.

- **Lowercasing text:** The main goal of this step is to make words that most likely are the same, look the same. For instance, in the case of the word 'Pizza', 'pizza' and 'PIZZA'. The three words are the same (and any lowercase you can imagine). It is not always the case if we

think of other examples (e.g. 'Apple', 'apple', 'APPLE'). However, we are losing sentiment by doing this step. Therefore, as we mentioned, we will be having two subsets, one where we do it and another one where we do not.

- **Substitution of numbers by a token:** Again in the same fashion as in lowercasing text, we are aiming to simplicity. On the overall of a text, and in the context that we are, it will not matter at all, if the number is 3, 7 or 20 it will be substituted by the token 'NUM'. As a particular case in this scenario, we have included a particular case when there is a currency next to the number, where we tag it as by the token 'PRICE'. We also do something similar with hours (e.g. 8 pm) to 'TIME'.
- **Removal of punctuation:** Punctuation does normally not add much meaning to the overall of a text. Therefore, it makes sense to eliminate it.
- **Apostrophe's removal (expanding contractions):** Apostrophe's expansion it is useful in order to diminish the dimensionality of our corpus. Performing this step is not trivial since doing it correctly is not as straightforward as it seems. However, some libraries help in the process. In particular, we are going to use the contractions[14] library. If we use the library straightforward, it does not work so well as one would expect. Many reviews use accents such as "‘" or "´" instead of the quote character. So what we did is to apply a regex to avoid cases where "I´ve read this thesis" would not translate it to "I have read this thesis".
- **Spelling correction:** Specially in a social media context where one is most likely using a phone, or is multi-tasking while writing, there might be some typos that will make that our words are treated as different when they are not. Furthermore, in future processes of the pipeline, this error might mean that the given the word ends up being considered another one, removed, etc. This step, could be very complex depending on which solution we want to approach. For example, we could work with distance algorithms to see which is the closest letter to the one spelled, from calculating the distance with Levenshtein to finding the most likely sequence with the Viterbi algorithm. In particular, we are going to use a library that will do a better job than us in this particular

case since it is tested. A sound library with years of development, this library that we will use is the TextBlob[15] library.

This process has been very computer demanding, taking days on the computer used to run the code.

- **Removing Stop Words and some custom words:** Stop Words are those words that are known not to contribute to the deeper meaning of the phrase. So it is generally a good idea to remove them to give more importance to those words that might be relevant. Some examples of stop words are 'the', 'is', 'a', 'I', etc. In this particular case, we will take the following approach: Since we are working in a system that uses opinion mining and summarization, it is good for us to remove them for the opinion mining aspect. However, this does not happen with summarization, where we should have those stop words. Therefore, we will give our opinion mining system the vocabulary without the stop words, and for the case of the text summarization system, we will provide it with them. For this task, we will take NLTK standard stop words list and remove them. Aside from removing only stop words, it is also a known practice to remove other common words on the corpus (basically customizing the stop words dictionary). It is also common to do the opposite, removing very particular words in the corpus. In our particular case, we decided not to remove rare words.

- **Tokenization:** Tokenization refers to the task of splitting a text, into smaller pieces (tokens), based on some criteria. Generally at a space basis. This process can be done on several levels: character, word, and even sentence. Each of these methods has its problems: word-level faces a problem out-of-vocabulary words (OOV) that later appear. These words will need to be tagged as UNK tokens to emphasize that it is an unknown word. It is a problem in terms of meaning because that unknown word will not provide any meaning to the phrase where it belongs. This problem does not happen at a character level, and the vocabulary size is small. However, the length of input-output sentences increases dramatically, and learning relationships between characters to form meaningful words is very difficult.

  The solution to this problem relies on subword tokenization algorithms. One of the most important ones is the byte pair encoding algorithm (BPE) widely used method among transformer-based models. BPE

tackles OOV effectively by segmenting them as subwords and representing the words in terms of these subwords, making the input and output sentences shorter than in the character tokenizations.

For the particular case of BERT, we are going to introduce its unique tokens [SEP], to mark the ending of a sentence, [CLS] to the start of each sentence, so BERT knows we are doing classification, [PAD] since we will be padding all the sentences to the same size. Finally, we will be using [UNK] for those not in the training set.

- **Lemmatization:** We are going to use lemmatization over stemming. We believe that lemmatization is way more potent than stemming since stemming works by cutting the suffix or prefix from the word. In contrast, lemmatization takes into consideration the morphological analysis of the word.

4.4 Exploratory Data Analysis

Once we have cleaned and pre-processed our textual data, we can now perform some visualizations, which will help us understand our data a bit more.

4.4.1 Data Splitting

Before beginning with the modeling phase, we decided to split the dataset. Our dataset is pretty big; therefore, we will split it as 70% train, 15% dev, and 15% test. The partition was done so that all the information of a property is in a given subset. We are working with a multi-document summarization, and we need all of our documents to be working with them.

So in total for our dataset we have the following sets:

Training Set (70%)	**Dev Set (15%)**	**Test Set (15%)**
123	26	27

Table 4: Dataset Composition

We think that it is worth mentioning that we took the subsets at random. So we could not pick reviews at random because it would merge properties, but we are picking which properties belong to each subset at random. So for the

particular case of sentiment classification, it would not impact, but it would impact the summarization results.

4.4.2 Word frequencies

First, if we see the frequencies of our reviews and titles, we can see that the most recurrent words are mainly stop words such as "the", "and", "is", etc.

Figure 48: Top 20 words in reviews before stop words

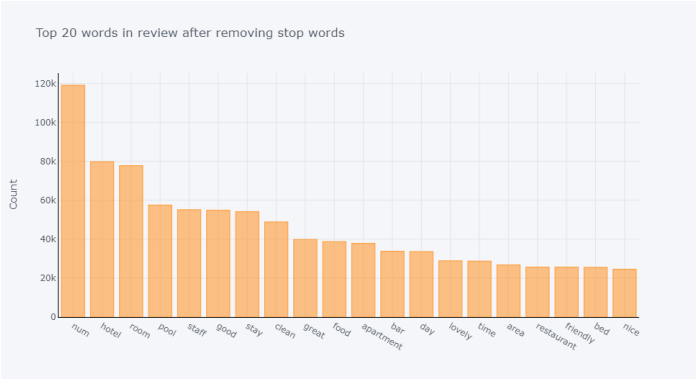


Figure 49: Top 20 words in reviews after stop words removal

When we remove stop words and lemmatize, we already see that the graphic completely changes and that we have other words that, as one can guess, will provide more inside into the contents of the review. For instance, we can see that some of the most common words are: hotel, room, good, stay, staff, clean,

friendly, etc. Whereas we had: the, and, to, was, of, in, etc, before removing stopwords and lemmatize. We can see that num is the most common one. That is the token we used to substitute the numbers in the text.

If we pay attention to the titles, we can see that the most common words change significantly regarding the review ones. When writing a review, it will have multiple phrases, and we will need to use connectors. It does not happen to a title where we already saw that they are much shorter. Moreover, when we compare titles with and without stop words and lemmatization, we see that some words appear but are not the most common ones in a title. It is basically due to the brevity of the titles that we can find.

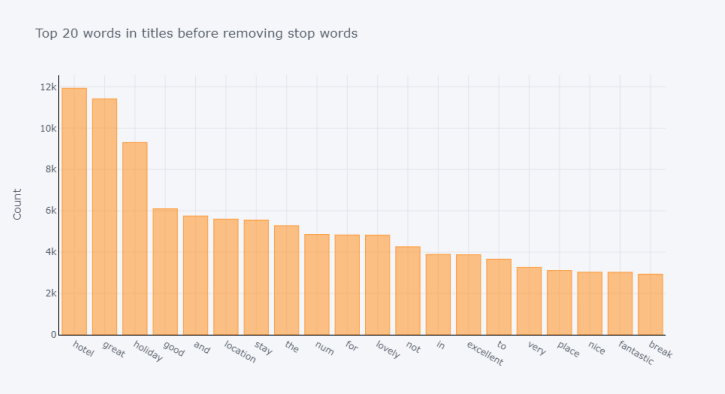


Figure 50: Top 20 words in titles before stop words

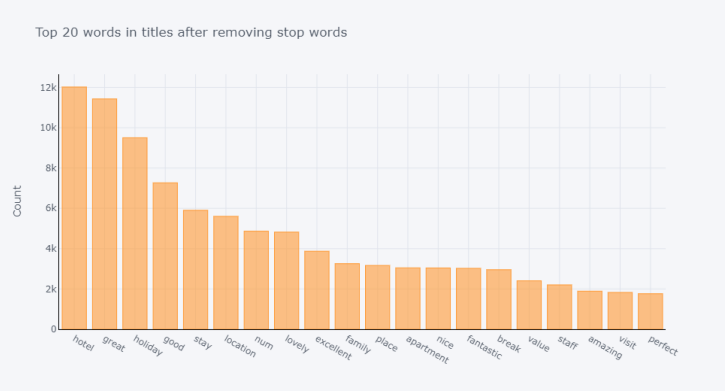


Figure 51: Top 20 words in titles after removing stop words

Furthermore, aside from the unigrams before and after removing stop words and lemmatizing, we can do the same for bigrams and trigrams (sets of two and sets of 3 consecutive words). Bigrams and trigrams, unlike unigrams, allow us to see common word associations in our corpus. Both cases (without considering the ones with the stop words, which are not so relevant). We can see that there are recurrent aspects that are important such as: "friendly staff", several days/nights/room, "bar-restaurant", "sea view", etc. Titles provide a better inside in general, where we can see that directly with only those bigrams, we could already see if that person is happy about their experience. We can see bigrams such as: "value money", "great hotel", "great location", etc. We see many repetitions between good and great pairs, but they are still insightful.

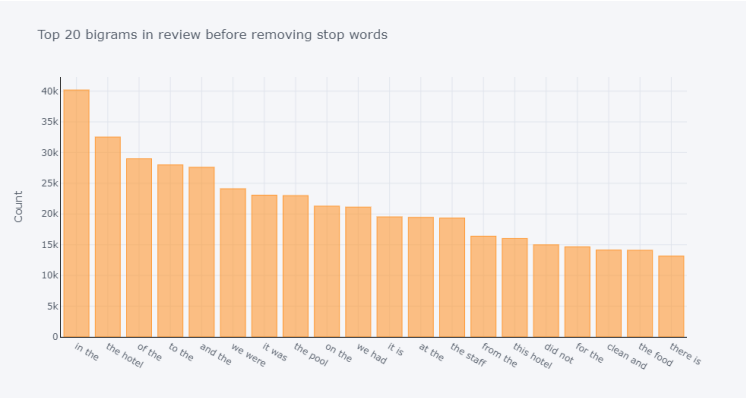


Figure 52: Top 20 bigrams in reviews before stop words

Figure 53: Top 20 bigrams in reviews after stop words

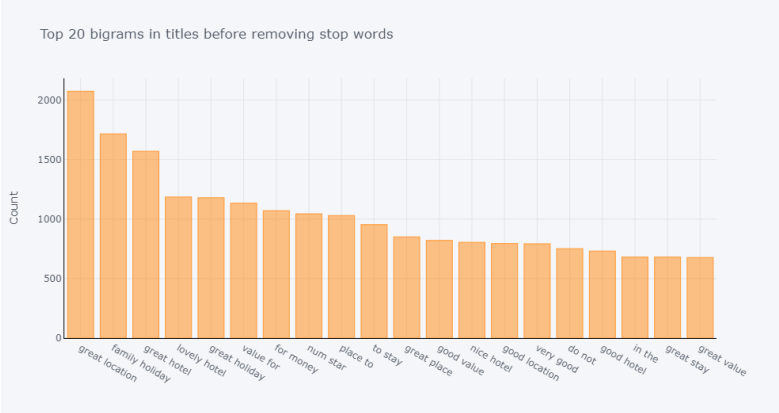


Figure 54: Top 20 bigrams in titles before stop words

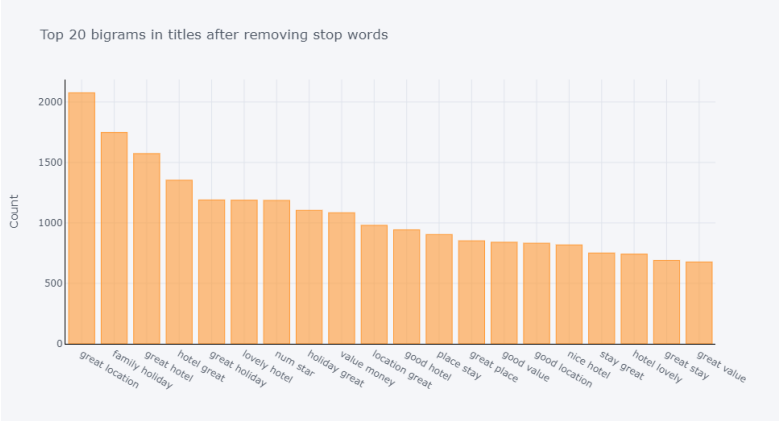


Figure 55: Top 20 bigrams in titles after stop words

If we pay attention to trigrams, they are not very interested in the case of once removed the stopwords and lemmatized because the results are not very relevant. Nevertheless, let us consider the case with stop words. We obtained similar results to the bigrams after removing them, making sense since those bigrams most likely had a stopword as a connector between them.

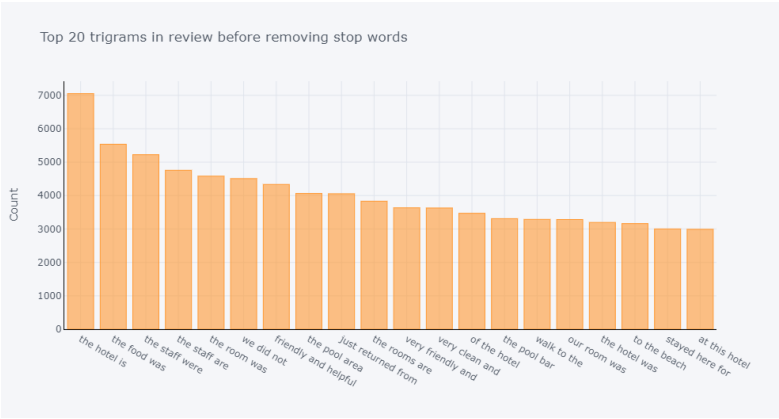


Figure 56: Top 20 trigrams in reviews before stop words

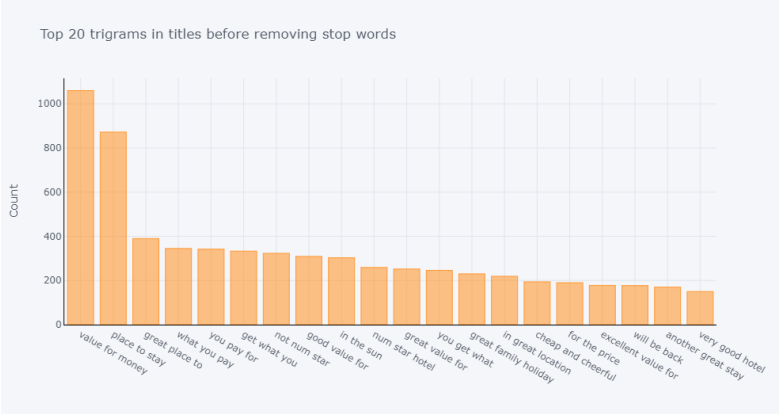


Figure 57: Top 20 trigrams in titles before stop words

4.4.3 Sentiment and Ratings

In order to analyse the sentiment on our corpus, and to have a grasp on how our dataset is composed by, we are going to use the textblob library which has a sentiment function that will return in a range of [-1,1] where 1 stands for the maximum positive sentiment and -1 a most negative one. Moreover, we will use the rating that we have from each review to see if a positive sentiment.

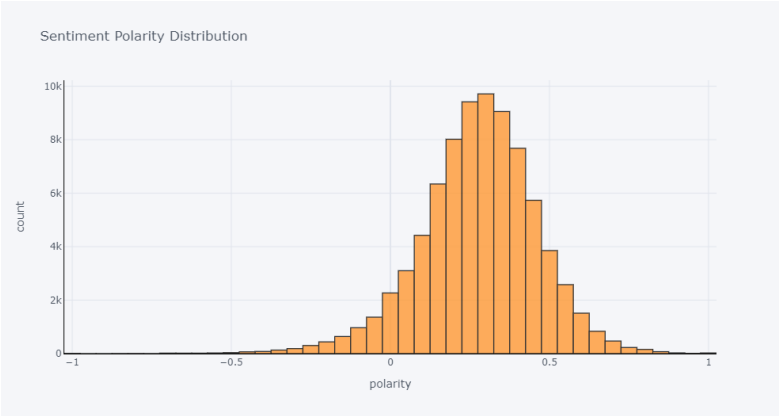


Figure 58: Sentiments across our corpus

As we can see, most of the sentiment polarity scores are more significant than zero, which means most of them are pretty positive. However, when we

are grouping them from their rating. We can see that they do not match as much as one would think. For instance, in the graph below, we have "Recommended" with reviews with 4 stars or more and "Not Recommended" for those with 1, 2 and 3 stars.

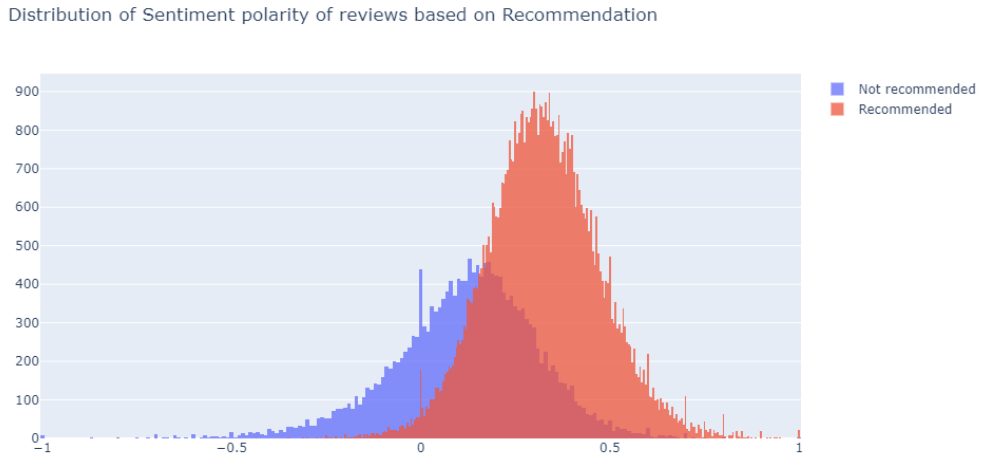


Figure 59: Sentiments across our corpus

So as we can see, our data, even in the not recommended cases, is not centered on the negative side of the corpus, which considering that we are dealing with 1 and 2 stars reviews, can be shocking. It happens because if we remember our corpus, around 70% of our data is represented by 4 and 5-star reviews. Therefore, we thought of giving it a more profound look by filtering reviews by stars and plotting again:

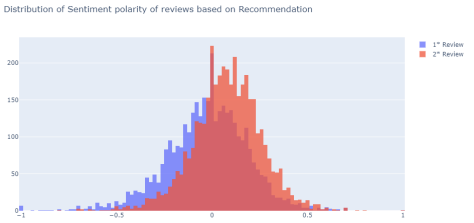


Figure 60: Sentiments on 1-2*

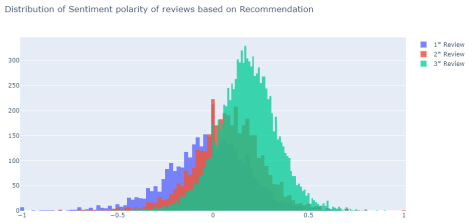


Figure 61: Sentiments on 1-3*

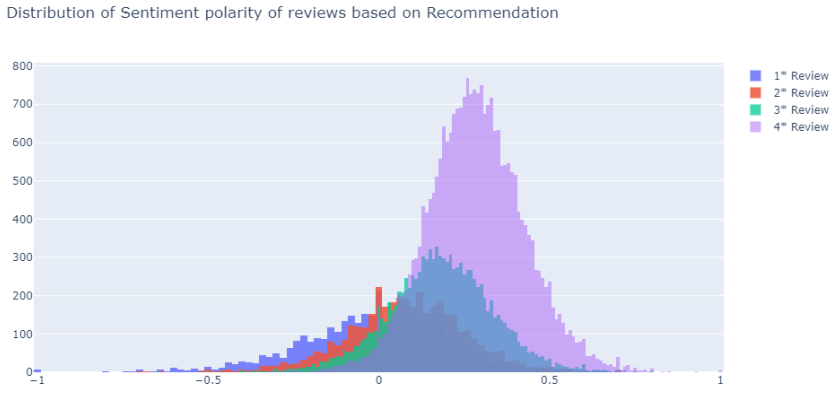


Figure 62: Sentiments on 1-4*

So as we can see, once we split in several plots everything, we can see that each review has a different center. This is good because it shows that there is some relationship between stars and the sentiment on the words used in the review. However, even the 1-star reviews are centered around the 0 mark, as we can see. Which is a multi-class approach would be the "Neutral". For 2 and 3 stars would be something between 0.10 and 0.25, and even in the best cases, a 5* review is centered around 0.4. But the spread of each review that the Gaussian does tells us that, for instance, there are also quite many 4-star reviews that could be interpreted as a lower or upper category that they are rated. This is partly because the 1-5 stars system is not so good if you aim to provide an accurate review.

Nonetheless, if we pay attention to the context of reviews, we can empirically

observe a tendency: when everything is all right, we keep it short, we do not over-praise. However, if we pay attention to the negative ones instead, we can see that they do tend to be longer, since the consumer tends to rant and expose why they are not satisfied, in a way, to take revenge from the property so other clients, do not go there. Therefore, we decided to see if this is rule meets and plot the ratting distribution together with the average number of letters and the average number of words.

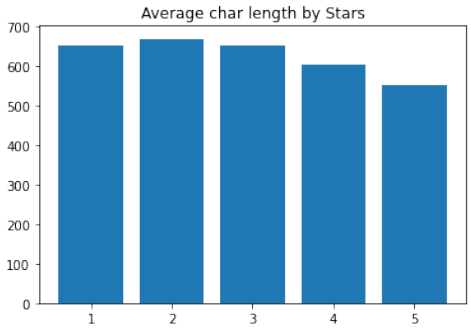


Figure 63: Characters vs ratings

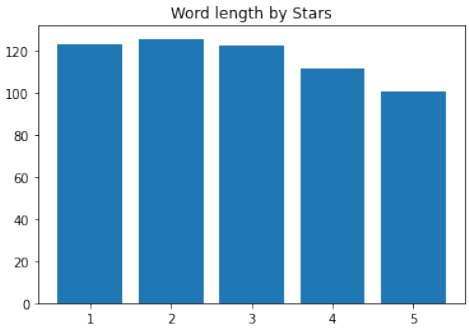


Figure 64: Words vs ratings

So once we see the graphic, we can see that there is up to a 20% of difference between, for instance, 2* reviews v. 5* reviews and indeed, we can see that 5* reviews on average, are the shortest. Whereas 1*-3* are the longest. We assume that this is because when you are complaining about something you tend to explain it as we mentioned earlier. We can also see that 4* reviews are significantly shorter shan 1-3* but longer than 5*. So we can really see that 1-3* is a group and 4-5* is another group. To put more evidence on the topic, we plotted a correlation matrix on different aspects of our dataset, we can see that both ratings and sentiment have a negative correlation with the length of a review and the number of words. This would explain the inverse relationship as the count of letters and words per review increases the overall rating and sentiment decreases.

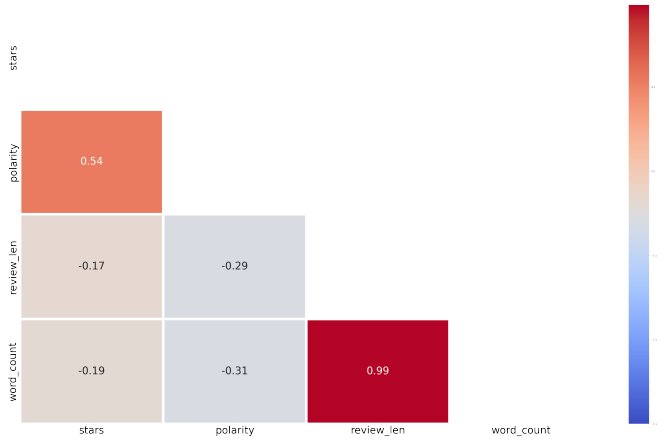


Figure 65: Sentiments on 1-4*

In the same fashion as the previous plots, we can see if some of the variables we get from each property might be useful. For instance, does being on the center somehow influence the reviews? Maybe hotels that tend to be more central have worse reviews because they tend to exploit that feature, and their quality is not as good as their location, which leads to a not so good review:

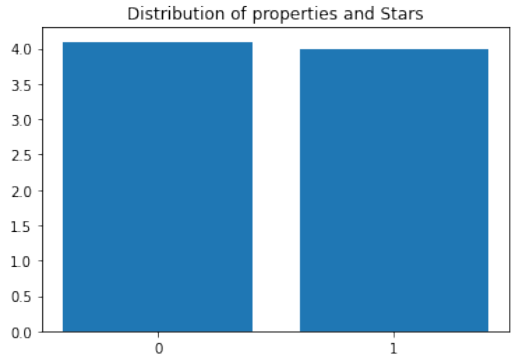


Figure 66: Ratings vs central location

Regarding the type of accommodation with reviews, we can see that there are differences. However, we cannot tell if it's a matter of luck with the properties we got or is a regular pattern that higher-end properties. For instance, a hostal tends to be worse than a hotel and will have worse reviews. Also,

we have much imbalance in our data, which does not help to evaluate the situation. Lastly, regarding the reviews they got vs the reviews they get, we can see a small difference again:

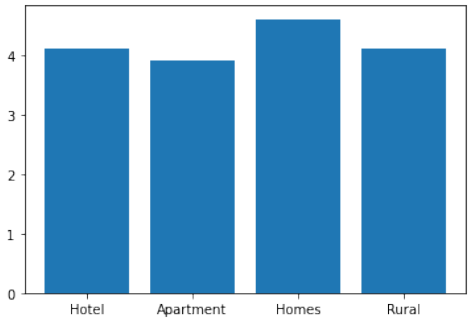


Figure 67: Ratings vs type of property

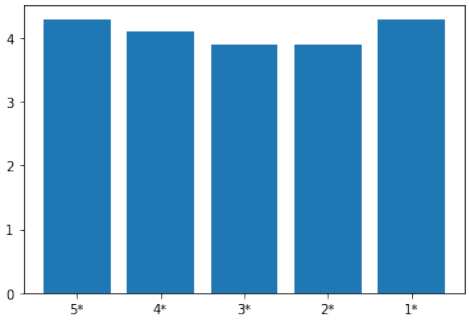


Figure 68: Stars vs reviews stars

Also, as we have seen earlier, we can see the top words in each category of reviews. We can observe that most common words are present across ratings, but some other words change (we removed from the clouds the top 3 from each because they were the same)

Figure 69: Top 50 words classified by Rating

we can see for our corpus phrases, their polarities. So we can have an idea of what a bad sentiment and a good sentiment review looks like and all the process in between:

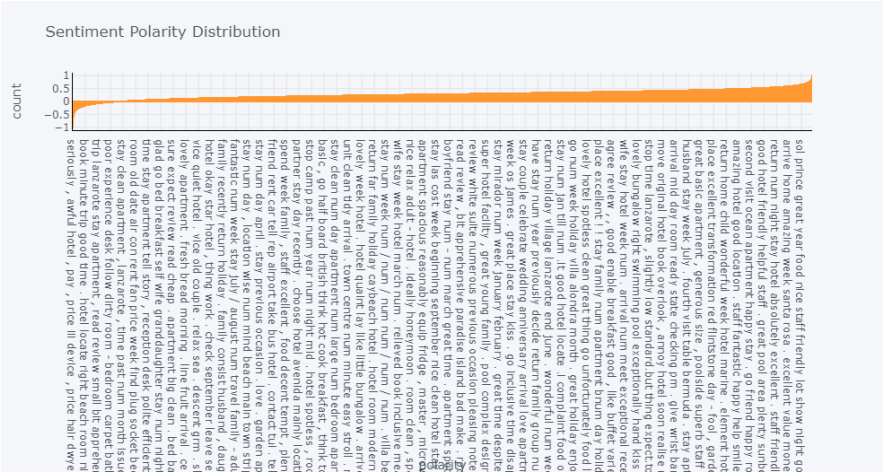


Figure 70: Sentiment analysis phrases from polarity

4.4.4 Other Insights

More in a lexical approach, we can also apply POS to our corpus. POS is a process of assigning parts of speech to each word, such as noun, verb, adjective, etc. With the TextBlob library, we can perform a POS analysis and plot the most frequent POS in our corpus before stop word removal and lemmatization:

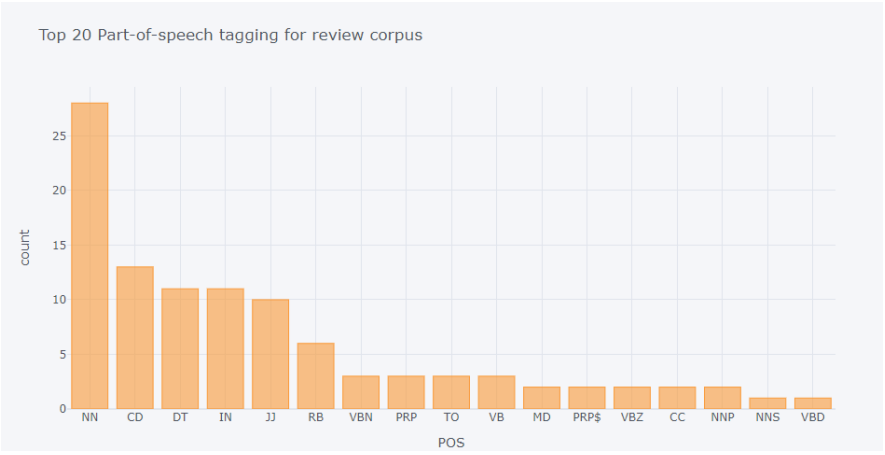


Figure 71: POS in our corpus

As we can see, the most prominent POS tags are singular nouns, "CD" that stands for cardinal numbers, "DT" that means determiner, and "IN" that means preposition/subordinating conjunction, and finally adjective (JJ) and adverbs (RB). However, something that seems surprising to us is the VB (verb) tags, which seem small, and how high is CD, as we have seen in previous plots.

Another interesting topic to cover is topic modeling. What does our corpus talk about? Which are the different topics that are covered? What topics do our low-rated reviews mention the most? We thought this would be an interesting approach, so we decided to model the topics from our dataset. We used the LDA approach and the NMF approach: both did not provide fascinating topics where the only exciting topic was around **food**.

4.5 Sentiment Analysis

As we have mentioned earlier, Sentiment analysis can be treated as a classification problem and in our case. As we mentioned earlier, 1-2-3 stars will be considered a negative review and 4-5 stars will be considered positive. We will be focusing on a binary classification with two possible outcomes: a positive review or a negative review.

As we have mentioned in earlier steps, our text is already pre-processed and cleaned. In summarization of the previous steps, we will say what our input is going to be: a text that has been removed from extra white spaces, removed foreign languages, lowercased, removed punctuation, expanded apostrophes, applied some spelling correction, emojis and emoticons have been converted to text, stemmed, removed common English stop words and tokenized according to BERT. However, we will work with two subsets to see how much improvement we have when removing stop words and lowercased or when keeping them. This is because removing them is not the best for summarizing, but it tends to be for sentiment analysis. Therefore, we will have a trade-off that we will have to evaluate. Also, in the case of lowercasing, some context gets lost but also some expressivity. It is not the same to be 'real good' as being 'REAL GOOD'.

4.5.1 Architecture

Regarding to the model, BERT has two model sizes:

- **BERT Base:** comparable in size to the OpenAI Transformer in order to compare performance.
- **BERT Large:** is a way bigger model. Is the implementation that achieved the state-of-the-art results reported in the paper.

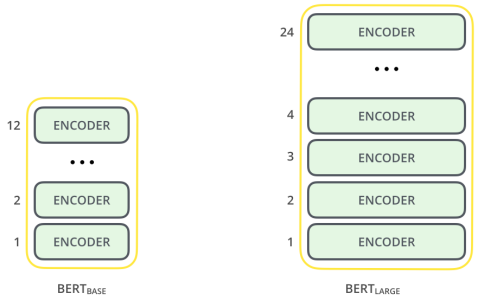


Figure 72: Bert model sizes

The difference between them is that even though both have many encoder layers, we have that BERT base has 12 versus 24 in the large version. Which also have more extensive feedforward networks (768 vs 1024 hidden units), more attention heads (12 vs 16). Which is more than the default configuration in the reference implementation of the Transformer in the initial paper (6 encoder layers, 512 hidden units, and 8 attention heads)[55].

In our particular case, we will work with BERT base because according to the volume of our data, it is more than enough. Therefore, we are going to use Huggingface[56] library in order to load and deal with our BERT transformer easier.

4.5.2 Pre-training

One of the strong points of BERT and from other transformers is that we do not need to train the models from scratch because there are available versions of these models trained. In particular, BERT was pre-trained on the English Wikipedia and the Toronto Book Corpus, which constitute a vast corpus. This is relevant because, thanks to transfer learning, we can then fine-tune them on downstream tasks. As mentioned, it is a widespread practice, but BERT was trained on a large corpus, which does not necessarily mean

that they will be coming from a specific domain. For instance, if BERT was trained on an electric domain, "current" would most likely mean "electricity" whereas in another context, "current" will mean "actual". Therefore, because our domain is specific, we are going to pre-train the model to improve our performance. Because otherwise, our model would be using the pre-trained embeddings that will tend to favor the generalized word representations.

In order to pre-train, we will need to run MLM once more but specifically for the data so that BERT will change the language distribution that is expected to the one from our domain-specific language. Therefore, we will need to choose a sequence length. In this particular case, what we will do is to choose the maximum length from our corpus. This is similar to what we did earlier counting the length, but we have to consider the tokens introduced:

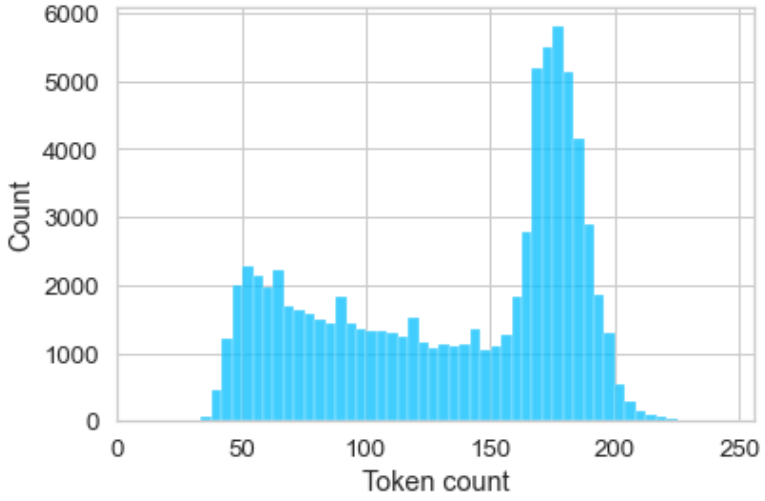


Figure 73: Token count from our corpus

As we can see in the plot, most of the reviews are around 175 tokens, and we have a maximum of around 225. Since the maximum can be 512, we will let the maximum be 250, which leaves a margin for a bit longer sequence.

4.5.3 Dealing with data heterogeneity

One of the problems we can think of is: how do we put other variables on the BERT model? Commonly BERT is just used to deal with textual data.

One approach that is used, which is quite naive, is to put everything as text. This can be a problem because there is a limit on the maximum token length that a transformer can handle. We want to ensure that there is an attention or weighting mechanism between the different modalities. The variables that we are going to work with are the following:

- **Title:** Text from the title review.
- **Review:** Text from the body of the review.
- **Title Length:** Length of the title.
- **Review length:** Length of the review.
- **Accommodation type:** Which as we have seen earlier. It is categorical and might have more types, but in our data we have: "hotel", "apartment", "homes" and "rural"
- **Stars:** It is a categorical variable that goes from 1-5 stars.

So if we were not going to be using these variables, we would typically apply a simple linear classifier on the CLS embedding. However, here we have additional features outside of BERT, so we need something that has the expressive power to combine them adequately.

After doing some research, we have seen that there is a library that deals with this issue. In particular, we have found MultiModal[57]. This library enables us to model for text and tabular data in combination with HuggingFace and basically, offers us a variety of MLP that otherwise, we would have to implement ourselves.

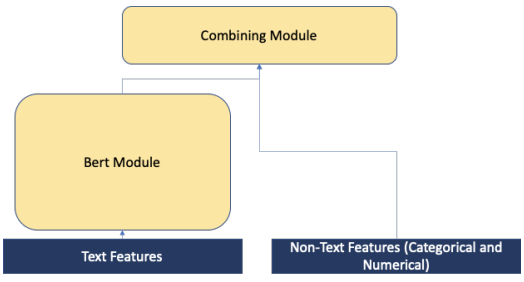


Figure 74: Multimodal schema

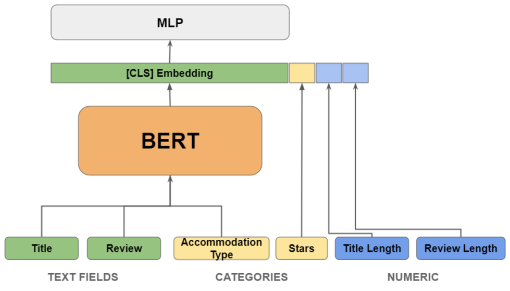


Figure 75: Multimodal own features

The approach that we have just seen, as explained in the blog of the author:[58]

> *At a high level, the outputs of a transformer model on text data and tabular features containing categorical and numerical data are combined in a combining module. Since there is no alignment in our data, we combine the text features after the transformer's output. The combining module implements several methods for integrating the modalities, including attention and gating methods inspired by the literature survey.*

So we will follow this approach, and we will treat the data as mentioned earlier as described in exception of the accommodation type, which we think that BERT can take a better result from it than just by treating it as a categorical feature. We could be dealing with the number of stars to convert it from category to number as an integer encoding, but that would be an error. This is because, as we already explained early in our state-of-the-art, integer encoding is generally wrong. After all, if we encode it as an integer, the meaning is not accurate if we calculate their similarity. It might be possible to just integer encode it in stars, but we prefer not to and to one-hot encoding instead.

The multi-modal toolkit offers a variety of methods, which are the following:[57].

Combine Feat Method	Equation
text_only	$\mathbf{m} = \mathbf{x}$
concat	$\mathbf{m} = \mathbf{x} \Vert \mathbf{c} \Vert \mathbf{n}$
mlp_on_categorical_then_concat	$\mathbf{m} = \mathbf{x} \Vert h_{\mathbf{\Theta}}(\mathbf{c}) \Vert \mathbf{n}$
individual_mlps_on_cat_and_numerical_feats_then_concat	$\mathbf{m} = \mathbf{x} \Vert h_{\mathbf{\Theta_c}}(\mathbf{c}) \Vert h_{\mathbf{\Theta_n}}(\mathbf{n})$
mlp_on_concatenated_cat_and_numerical_feats_then_concat	$\mathbf{m} = \mathbf{x} \Vert h_{\mathbf{\Theta}}(\mathbf{c} \Vert \mathbf{n})$
attention_on_cat_and_numerical_feats	$\mathbf{m} = \alpha_{x,x}\mathbf{W}_x\mathbf{x} + \alpha_{x,c}\mathbf{W}_c\mathbf{c} + \alpha_{x,n}\mathbf{W}_n\mathbf{n}$ where $\alpha_{i,j} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top[\mathbf{W}_i\mathbf{x}_i \Vert \mathbf{W}_j\mathbf{x}_j]\right)\right)}{\sum_{k\in\{x,c,n\}}\exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top[\mathbf{W}_i\mathbf{x}_i \Vert \mathbf{W}_k\mathbf{x}_k]\right)\right)}.$
gating_on_cat_and_num_feats_then_sum	$\mathbf{m} = \mathbf{x} + \alpha\mathbf{h}$ $\mathbf{h} = \mathbf{g_c} \odot (\mathbf{W}_c\mathbf{c}) + \mathbf{g_n} \odot (\mathbf{W}_n\mathbf{n}) + b_h$ $\alpha = \min(\frac{\Vert\mathbf{x}\Vert_2}{\Vert\mathbf{h}\Vert_2} * \beta, 1)$ $\mathbf{g}_i = R(\mathbf{W}_{gi}[\mathbf{i} \Vert \mathbf{x}] + b_i)$ where β is a hyperparameter and R is an activation function
weighted_feature_sum_on_transformer_cat_and_numerical_feats	$\mathbf{m} = \mathbf{x} + \mathbf{W}_{c'} \odot \mathbf{W}_c\mathbf{c} + \mathbf{W}_{n'} \odot \mathbf{W}_n\mathbf{n}$

Figure 76: Different methods that we can use in the combine method

Where:

- m denotes the combined multimodal features.
- x denotes the output text features from the transformer.
- c denotes the categorical features.
- n denotes the numerical features
- h_Θ denotes a MLP parameterized by Θ
- W denotes the weight matrix
- b denotes the scalar bias

We will be opting for the weighted feature sum on transformer cat and numerical features.

4.5.4 Fine tuning

In order to properly fine-tune, we will be working with several hyperparameters, which we must configure adequately. If we pay attention to the BERT paper, we can find that we should put our attention on the learning rate and the batch size since those are the most critical parameters[61]. What BERT authors recommend is fine-tuning for four epochs over the following hyper-parameter options:

- **Batch sizes:** 8, 16, 32, 64, 128
- **Learning rates:** 3e-4, 1e-4, 5e-5, 3e-5

Therefore, we put our efforts into calculating which of those hyper-parameters were the best. We found that the best learning rate for us was 3e-5 and 16 as the batch size.

- **Optimizer:** We are going to be using the AdamW optimizer. The AdamW optimizer is a version of the Adam algorithm that has a fix weight decay[60] and when we execute it, we will not perform warm-up steps.
- **Learning rate:** We are going to work with a learning rate of 3e-5. Also, we are not going to correct the bias.
- **Batch size:** As it is known, the bigger the batch size, the lower the accuracy. In our case, 16 was the best result.
- **Loss function:** Since we only have two outputs, we are going to be using the binary cross-entropy function (also known as log loss): $H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$

Moreover, we used gradient clipping with normalization of 1.0. Gradient clipping is a technique used in case the exploding gradient occurs, which is the case where the gradients are too large, and as a consequence, our training process would not be stable.

4.5.5 Training and model evaluation

There are several topics to discuss and evaluate:

- We have four different train models: the one with the entire pipeline,

the one without lowercasing, the one without the stopwords and lemmatization, and one without both steps.

- Does adding tabular data on our BERT model suppose improvement respect not adding it?
- Finally, from the model that it works best on our training/validation. How well performs on the test set?

In summarization of our findings on the training-validation set:

- We found that lowercasing was a bad idea. Many approaches directly use a pre-trained version that is best uncased, and it is familiar not to lowercase the text. This gets confirmed by the results, where our results are worse when we lose this context.
- Removing stopwords improve the results, but very slightly that it can be considered insignificant. It could have been a good idea to separate the steaming process and the stopword removal.
- Regarding the tabular data on the BERT model, we saw that adding the tabular data improves the results.

We kept a specific configuration for the different datasets so that there were no differences between the results, which was done with the BERT without the tabular data.

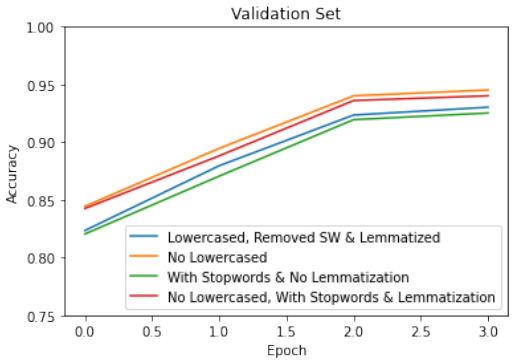


Figure 77: Version comparison

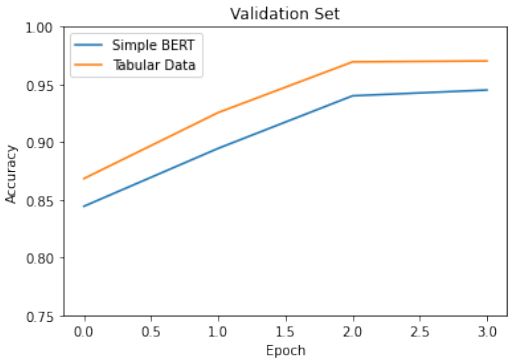


Figure 78: BERT comparison

As we can see, our best result is the BERT with the tabular data, obtains in the validation set around a 97% of precision, this is an excellent result. This

is partly because we are classifying only in two classes. If we were working in a neutral class, it is in the class where we would be having the most problems. Also, we had a lot of unbalance on our data, so we have to see if it generalizes well on the test set or because of the subset we have on the validation set, and we are overfitting.

In regards with the model evaluation, we applied our model to the test set, we can see that the model generalised well. The precision diminished up to 96% which is still a very good result. Below it can be seen the confusion matrix:

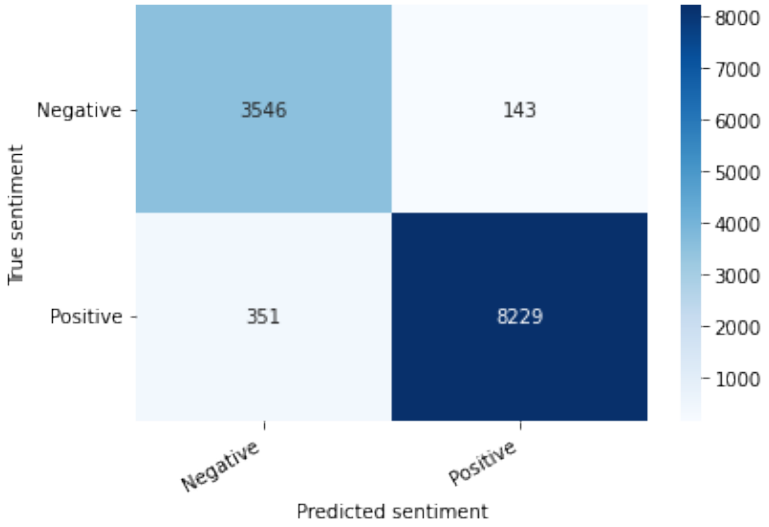


Figure 79: Confusion Matrix of our model on the test set

4.6 Text Summarization

As we mentioned in our proposal, once we filter our data by the review's feelings, we will summarize those texts. As we have seen in our state-of-the-art, there are multiple parameters to choose from when defining a summarization system:

- **Input:** Reviews from TripAdvisor for a same property
- **Output:** Our output is a text of around 100 words (on average, our input has 110 words). This is something similar to the length of a tweet.

- **Purpose:** Informative
- **Form:** Extractive summarization
- **Dimensions:** Multidocument
- **Context:** Generic, we want to treat all aspects treated on the text as much as possible. However, it is in a way, somewhat query-specific because we are using the filtering by sentiment before we reach to this system.

4.6.1 Framework & Architecture

Multi-document summarization typically has a set of steps that can see in the below image. We will be working on these steps as a baseline so that we make as straightforward as possible which decisions we take at every step and explain which other plausible decisions there were.

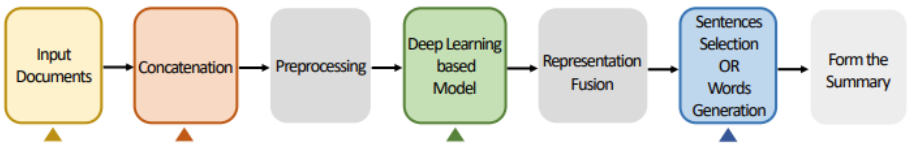


Figure 80: Processing framework of Multi-document summarization[62]

First, our input type, as we have just specified, is composed of many short documents of short length. This is something prevalent in reviews. A large number of input documents may contain a contradiction, redundancy, and complementary information. Therefore, we need to think well about which concatenation method will work well for us. There are two types that are very common:

- **Flat concatenation:** This method is very simple: we concatenate everything in a single document. So the task we have to do would be the same as a single document summarization. However, it has a drawback: the model that will have to deal with the document needs a solid capacity to handle long input sequences.

- **Hierarchical Concatenation:** This method is able to preserve cross-document relations, which leads to a better model, since it will be richer and will affect in its effectiveness. This is normally done by applying clustering and graph techniques, which are undoubtedly helpful with redundancy (one of the significant problems on extractive summarization)
  However, many deep learning models do not make full use of it.

In our particular case, we are going to go with a flat concatenation. This is because of several reasons:

- We want to use transformers. Transformers are capable of dealing with large inputs; since we can batch process them, we do not have the vanishing gradient problem (also can be parallelized)

There are many different kind of architectures[62] that we can consider. If we consider that we are going to use transformers, we have that we can use the following network approaches:

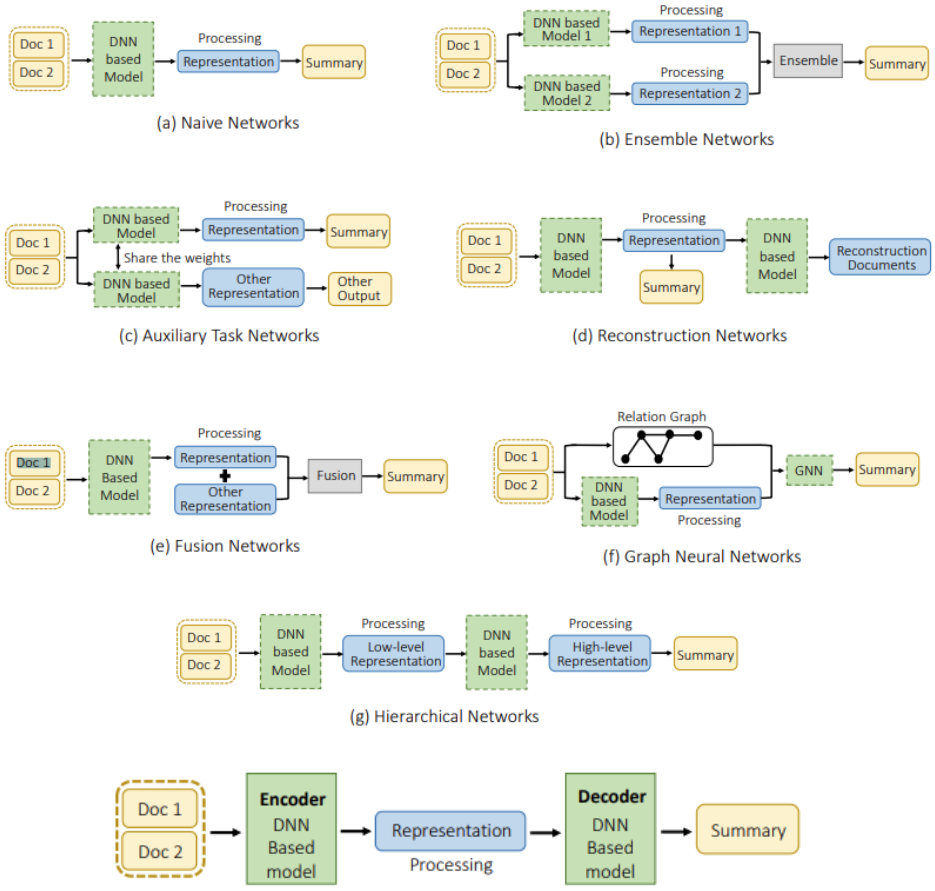


Figure 81: Different types of architectures

From the ones that we can see, we think the best approach is a hybrid encoder-decoder framework. Where we will have two encoders: one for the titles and the other for the reviews. We will be encoded in a document direction. Then we will have a middle process where we will finally work towards the best solution to decode the solution to have our summary.

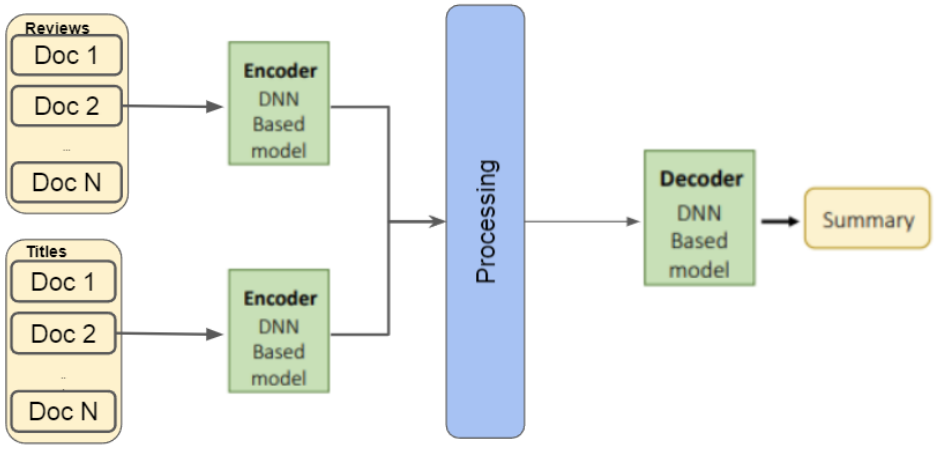


Figure 82: Summarization system architecture

In our particular case, we will be concatenating for each of the reviews and titles, all of the titles and reviews from the same property. Then what we will do is to encode those two texts in the model that we decided that we will use, that is GPT-2 medium.

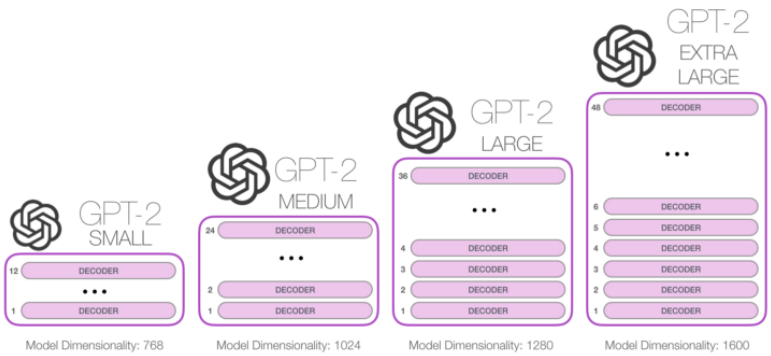


Figure 83: GPT-2 model sizes

One could wonder why we will use GPT-2 and not, for instance, BERT again. The answer to that question is that BERT is composed of encoders, and GPT-2 is composed of decoders. This leads in that BERT lost its auto-regressive nature in exchange for gaining the ability to "reading" from both

sides and get better context. Again, this comes from the way that attention is handled.

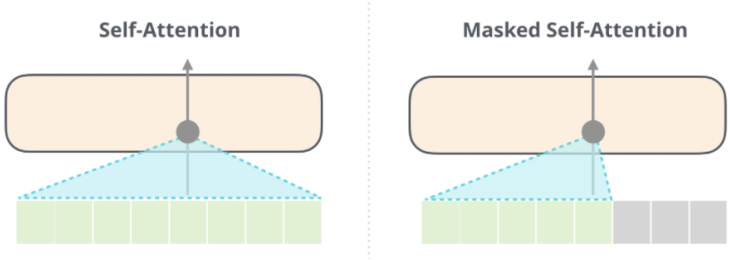


Figure 84: Self-Attention vs Masked Self-Attention

The image below is very descriptive and gives an intuition why GPT-2 can generate text. Because once we have one word, the attention is directed to what we already have, and it will be, which will help us to know which is the most likely word.

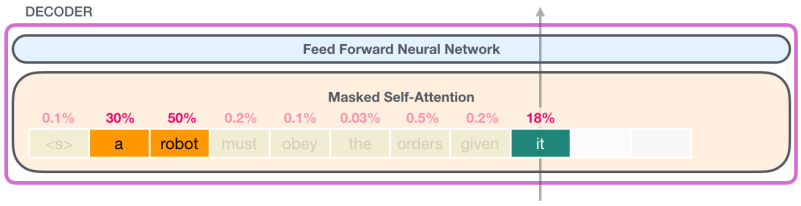


Figure 85: Illustrated Self-Attention

Then regarding to the Processing block, what we will have is a MLP. This MLP what basically will do is to work with the Cross-entropy loss $L_{CE} = -\sum_{i=1} y_i \log(\hat{y}_i)$ where y_i is the target salience score that is usually calculated with ROUGE. We could use many ROUGE variants which may or not may make more sense. However, in this particular case, we opted to use simply ROUGE.

4.6.2 Evaluation

As we have mentioned earlier across the project, we are going to be using the ROUGE-F1 metric. For that purpose, we needed to do a "golden summary"

for each property for both: positive and negative reviews for the 27 different properties chosen as a test. We have done this process manually according to the data that we dispose of on our dataset.

However, to be more readable, we are going to be doing first to calculate the precision and recall for each answer and then calculate this metric. We will do the following to calculate the recall as the number of n-grams found in the model and references among the total of n-grams in reference. Then we will calculate the precision as the number of n-grams found in the model and reference among the number of n-grams in the model. Once we have calculated both, we can calculate the final F1-Score as: $2 * \frac{precision * recall}{precision + recall}$

Below, we provide to evaluate the metrics that Python ROUGE module returns: ROUGE-1, ROUGE-2 with the F1 score for each. The results can be seen in the plots below:

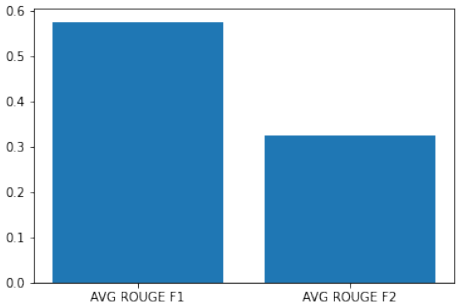


Figure 86: Average ROUGE F1 and ROUGE F2

As we can see, our final results, on average, are 0.575 in the case of ROUGE F1 and 0.325 for ROUGE F2.

If we pay attention to each of the subsets, the positive and negative reviews, we can see that the positive reviews are being generated better than the negative ones. This is most likely because of the data unbalance. Also, the vocabulary and the complaints are likely to introduce different aspects, which we do not find in the positive ones.

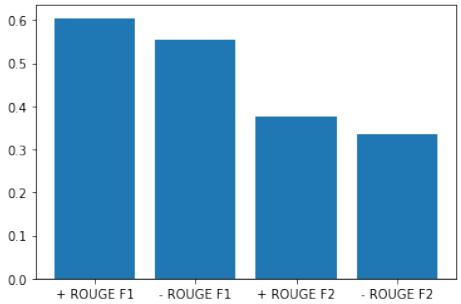


Figure 87: ROUGE F1 and ROUGE F2 for positive and negative reviews

5 Conclusions

When we started this thesis, we stated four major objectives: Performing deep state-of-the-art work, obtaining our dataset to work with, defining a methodology, and obtaining good results. Among the four proposed objectives, we can say that all of them have been successfully satisfied. Below we are going to explain why we believe each of them has been satisfied:

- **Get to know the NLP ecosystem:** With all the work that we have done and by working in an NLP project from beginning to end, I believe that we got a deep understanding of what is going on in this field. We have also talked about several techniques that have been used and why they are worse than other methods, so we do not only understand what can be done, but also we know most of the tools that are being used and why.
- **Obtaining our dataset to work with:** We have found a way to retrieve properties from StayforLong and learned how to link it with TripAdvisor reviews. Once we knew how to get a property from StayforLong to their reviews on TripAdvisor, by using the BeautifulSoup library, we could scrap the content from TripAdvisor. This implied learning how the links fluctuate on pagination, the language of reviews, and parsing the HTML content per page.
- **Defining a methodology:** We have foreseen the different methodologies possible on data science and opted to use CRISP-DM. We have also worked and emphasized the importance of finding an appropriate evaluation. Because it is essential, we have also studied different

evaluation methods, where we have finally opted for ROUGE.

- **Obtaining good results** We are happy with the results obtained, especially with the classification process, that it is easier to see if it was done correctly or not. However, in the case of multi-document summarization, it is more complicated. As we mentioned, we had to do our golden summaries, which, to begin with, introduce a bias and are known to be challenging to generate. Also because even if we have been cautious when doing them, we are dealing with extractive summarization. As humans, we tend to generate abstractive summarization, so it can be a factor that directly affects our results. Also, it is not very clear how good a model is performing just by considering ROUGE. There is another consideration to be done that it is a human validation.

Finally, I have to say that I did not expect NLP to be so advanced in terms of complexity and knowledge, and I am thrilled that I chose this topic because I have learned a lot on neural networks, which I have always wanted to learn.

5.1 What I would do differently

As we mentioned, we are happy with the results. But as it is common, once you end something, you know how you would do it next time. In this particular case, I would change the following steps:

- We are happy with our results, and we did the best we could. However, when we chose the topic, there were more standard options, which had research behind them, highly used on benchmarks, and we could compare our results.
- Also related with the previous argumentation, dealing with a more standard dataset would have most likely helped have golden standards in evaluating the summarization. That would have been very helpful and removed some bias that we might have introduced involuntarily in the evaluation process.
- The scope was too big. One of the two systems that we have worked with would have been already enough to be a thesis, and I found it extremely difficult to cope with both problems simultaneously. Furthermore, most of the topics we are dealing with are very recent, and there is a lack of high-impact papers (without even talking that not all the papers

published are necessarily available for me).

- In the same fashion as the previous, I would not scrap again from a public website. I know that I would not have been given access to the API in this particular case because it is restricted. However, dealing with the scrapping process has been tedious. Basically, the time it took to compute and gather the data most likely also goes against the ToS of TripAdvisor.

6 Future Work

Although we are satisfied with the work done, many things can be done in the future:

- As it has been mentioned earlier on, crawling data has been a costly process. If that was not the case, we could have first enlarged the data by crawling more information (by removing the 600 reviews capping), and we could have also removed the restriction of comments only on English. This would have enlarged our dataset but would have also added another problem to the translation scope.

  This could have been solved by using some libraries (or the Google translator API) to translate the text. However, we all know that not even Google is proficient when translating, mainly depending on the language. This would have potentially blurred our results. Since we have an opinion mining system on our solution, we believe that would have biased the results since linguistic precision is essential in that scenario.

- We can expand our scrapping process to get some other information. There is some information that we are missing, but it could have been interesting, for instance, to get the overall rate that the user gives and specific ratings from each aspect the user is asked to. We did not do this initially because it does not happen for all reviews. However, we think that it can be insightful.

Figure 88: Example of additional fields

- I believe that there is some room for improvement in the pre-processing of the text. We did a good job, but I think that some extra personalized stop words, for instance, would probably help. I believe that the auto-corrector that is being used could be something that we can expand

and build our auto-corrector. This way, we would not be relying on a library that is not a state-of-the-art level of performance.

- It would be good to put the system to work on the cloud with GPUs. First, this would help to the speed of the system. Then it would also be able to deal with more data.
- There is room for improvement in the summarization system. We could introduce other data structures like a discourse tree. We could add methods to avoid redundancy, etc.

References

[1] StayForLong website
https://es.stayforlong.com/

[2] TripAdvisor website
https://www.tripadvisor.es/

[3] The Secret Ratio That Proves Why Customer Reviews Are So ImportantIt's what you don't know that hurts you
https://www.inc.com/andrew-thomas/the-hidden-ratio-that-could-make-or-break-yo
html#:~:text=A%20customer%20who%20has%20a,customers%
20leaves%20a%20good%20review

[4] Liddy, E.D. 2001. Natural Language Processing. In Encyclopedia of Library and Information Science, 2nd Ed. NY. Marcel Decker, Inc.

[5] Turegun, Nida. (2019). Text Mining in Financial Information. 1. 18-26.

[6] Radev, D. R., Hovy, E., McKeown, K. (2002). Introduction to the Special Issue on Summarization. Computational Linguistics, 28(4), 399–408.

[7] S. F. Esuli A, "Determining the semantic orientation of terms through gloss classification," presented at Proceedings of the 14th ACM international conference on Information and knowledge management, Bremen, Germany 2005.

[8] Mani, I., Maybury, M. (1999). Advances in Automatic Text Summarization. MIT Press.

[9] Coursera Natural language Processing Specialization:
https://www.coursera.org/specializations/
natural-language-processing

[10] Pramita Widyassari, A., Rustad, S., Fajar Shidik, G., Noersasongko, E., Syukur, A., Affandy, A., Rosal Ignatius Moses Setiadi, D. (2020). Review of Automatic Text Summarization Techniques Methods. Journal of King Saud University - Computer and Information Sciences. doi:10.1016/j.jksuci.2020.05.006

[11] Gradient descent image:
https://interactivechaos.com/sites/default/files/inline-images/tutorial_ml_optimizer_04.png

[12] Learning rate image resource:
https://www.educative.io/edpresso/learning-rate-in-machine-learning

[13] Oliveira, Nicollas Pisa, Pedro Andreoni, Martin Medeiros, Dianne Menezes, Diogo. (2021). Identifying Fake News on Social Networks Based on Natural Language Processing: Trends and Challenges. Information. 12. 38. 10.3390/info12010038.

[14] Contractions library:
https://github.com/kootenpv/contractions

[15] TextBlob library:
https://textblob.readthedocs.io/en/dev/

[16] Emoji library:
https://github.com/carpedm20/emoji

[17] Scientific method:
https://i.pinimg.com/736x/b5/68/5c/b5685cc40f529f890dc08117746c2de7--scientific
jpg

[18] su, Yuanhang Kuo, C.. (2018). On Extended Long Short-term Memory and Dependent Bidirectional Recurrent Neural Network. Neurocomputing. 356. 10.1016/j.neucom.2019.04.044.

[19] Chained RNN diagram
https://www.codeproject.com/Articles/1272354/ANNT-Recurrent-neural-networks

[20] Cross Industry Standard Process for Data Mining on Wikipedia:
https://es.wikipedia.org/wiki/Cross_Industry_Standard_Process_for_Data_Mining

[21] Heck, Petra Cruz, Luís. (2021). Engineering Machine Learning Applications. 10.13140/RG.2.2.33897.83041.

[22] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena,Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," 2019.

[23] Yang Liu. Fine-tune BERT for extractive summarization. arXiv preprint arXiv:1903.10318, 2019.

[24] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In Text summarization branches out, 2004.

[25] Radev, D., Tam, D. (2003). Summarization Evaluation Using Relative Utility. In Proceedings of the Twelfth International Conference on Information and Knowledge Management (pp. 508–511). Association for Computing Machinery.

[26] Ani Nenkova and Rebecca Passonneau 2004. Columbia University. Computer Science Department Evaluating Content Selection in Summarization: The Pyramid Method

[27] Bing Liu, Department of Computer Science, University of Illinois at Chicago

[28] Word2vec brief explantion from Tensorflow documentation: https://www.tensorflow.org/tutorials/text/word2vec

[29] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation

[30] FastText repository: https://github.com/facebookresearch/fastText

[31] Hochreiter, S., Schmidhuber, J"urgen. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780.

[32] Cho, Kyunghyun; van Merrienboer, Bart; Gulcehre, Caglar; Bahdanau, Dzmitry; Bougares, Fethi; Schwenk, Holger; Bengio, Yoshua (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". arXiv:1406.1078

[33] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, ; Polosukhin, I. (2017). Attention is All You Need. In Proceedings of the 31st International Conference on Neural Information Processing Systems (pp. 6000–6010). Curran Associates Inc..

[34] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners. Journal is required!.

[35] Baxendale, P. B. (1958). Machine-Made Index for Technical Literature—An Experiment. IBM Journal of Research and Development, 2(4), 354–361. doi:10.1147/rd.24.0354

[36] Luhn, H. P. (1958). The Automatic Creation of Literature Abstracts. IBM Journal of Research and Development, 2(2), 159–165. doi:10.1147/rd.22.0159

[37] Edmundson, H. (1969). New Methods in Automatic Extracting. J. ACM, 16(2), 264–285.

[38] Automatic Text Summarization Horacio Saggion Department of Computer Science University of Sheffield England, United Kingdom

[39] Radev, D. R., Jing, H., Budzikowska, M. (2000). Centroid-based summarization of multiple documents. NAACL-ANLP 2000 Workshop on Automatic Summarization -. doi:10.3115/1117575.1117578

[40] Conroy, J., O'leary, D. (2001). Text Summarization via Hidden Markov Models. In Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (pp. 406–407). Association for Computing Machinery.

[41] Carbonell, J., Goldstein, J. (1998). The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries. In Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (pp. 335–336). Association for Computing Machinery.

[42] Kupiec, J., Pedersen, J.O., Chen, F.R. (1995). A trainable document summarizer. SIGIR '95.

[43] Radev, D.R., McKeown, K. (1998). Generating Natural Language Summaries from Multiple On-Line Sources. Comput. Linguistics, 24, 469-500.

[44] Lin, C-Y. and E.H. Hovy. 1997. Identifying Topics by Position. In Proceedings of the Applied Natural Language Processing Conference (ANLP-97), 283–290. Washington.

[45] Osborne, M. (2002). Using maximum entropy for sentence extraction. Proceedings of the ACL-02 Workshop on Automatic Summarization

[46] Erkan, G., Radev, D. R. (2004). LexRank: Graph-based Lexical Centrality as Salience in Text Summarization. Journal of Artificial Intelligence Research, 22, 457–479. doi:10.1613/jair.1523

[47] Daniel Marcu. From discourse structures to text summaries. In Workshop on Intelligent Scalable Text Summarization, ACL97 and the EACL97, pages 82–88, July 1997.

[48] Barzilay, M. (1997). Using Lexical Chains for Text Summarization. In Intelligent Scalable Text Summarization.

[49] Summarizing similarities and differences among related documents I Mani, E Bloedorn Information Retrieval 1 (1), 35-67

[50] Multi-document summarization by graph search and matchingI Mani, E Bloedorn arXiv preprint cmp-lg/9712004

[51] Gerard Salton, Amit Singhal, Mandar Mitra, Chris Buckley (1997). Automatic text structuring and summarization. Information Processing Management, 33(2), 193-207.

[52] Aone, C., Okurowskit, M. E., Gorlinskyt, J., et al. A Scalable Summarization System Using Robust NLP. In Proceedings of the ACL'07/EACL'97 Workshop on Intelligent Sclalable Text Summarization, pages 66-73, 1997

[53] Brandow, R., Mitze, K., Rau, L. F. Automatic condensation of electronic publications by sentence selection. Information Processing Management, 31(5):675–685, 1995.

[54] Orundefinedsan, C. (2003). An Evolutionary Approach for Improving the Quality of Automatic Summaries. In Proceedings of the ACL 2003 Workshop on Multilingual Summarization and Question Answering - Volume 12 (pp. 37–45). Association for Computational Linguistics.

[55] Bert illustrated guide
https://jalammar.github.io/illustrated-bert/

[56] Hugginface library:
https://huggingface.co/

[57] MultiModal library:
https://github.com/georgian-io/Multimodal-Toolkit

[58] How to Incorporate Tabular Data with HuggingFace Transformers:
https://medium.com/georgian-impact-blog/how-to-incorporate-tabular-data-with-huggingface-transformers-b70ac45fcfb4

[59] Hotel-Review Datasets:
https://www.cs.cmu.edu/~jiweil/html/hotel-review.html

[60] AdamW HuggingFace documentation:
https://huggingface.co/transformers/main_classes/optimizer_schedules.html

[61] Hyper-parameter tuning, what to put our efforts in:
https://wandb.ai/jack-morris/david-vs-goliath/reports/Does-Model-Size-Matter-A-Comparison-of-BERT-and-DistilBERT--VmlldzoxMDUxNzU

[62] Congbo Ma, Wei Emma Zhang, Mingyu Guo, Hu Wang, Quan Z. Sheng. (2020). Multi-document Summarization via Deep Learning Techniques: A Survey.

A Pre-NN State-of-the-art on summarization

One of the first and most influential papers on automatic summarization, is from Baxendale (1958)[35]. In his paper, he introduces the positional method which works particularly well for some kinds of text such as scientific papers. He concluded from an analysis of 200 paragraphs, that the most influential information (on that type of genre) comes from picking the first and last sentences of a given paragraph, 85%. This happens because in those positions is were usually the topic sentences are usually located.

On 1958 also we have a paper from Luhn (1958)[36]. He was one of the first papers in applying automatic stemming techniques were words were stemmed to their root forms, he also removed stop words and he used as features the frequency of content terms.

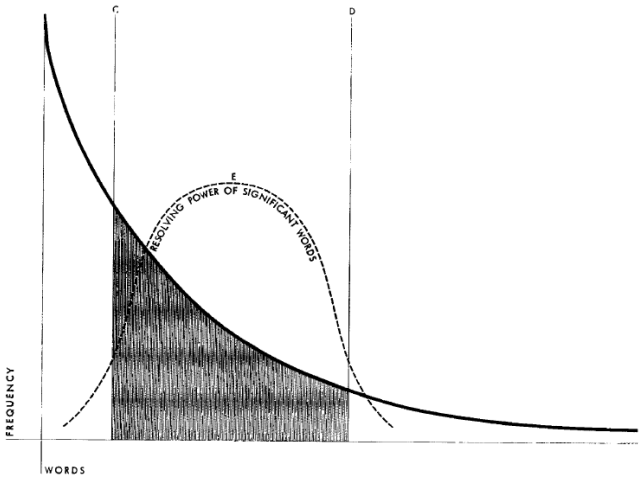


Figure 89: Stop words removal

According to his methodology, the words that are the most frequent ones, are stop words and those that appear only once, are the most informative ones. Therefore, as can be seen on the image, the E metric that we can see on the image, reaches its highest point for words that are on the middle frequency range. So he concludes that sentences that have those words, are those phrases that are the most relevant and that should be in the summary. To measure this, he introduces a sentence-level significance factor which basically looks

for concentrations of salient content terms.

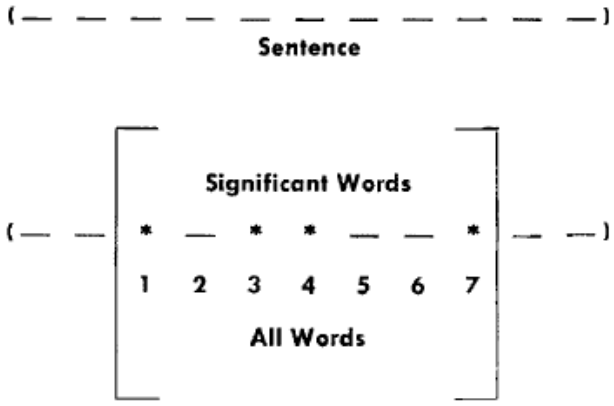


Figure 90: Sentence level significance

Edmunson (1969)[37] worked on technical documents and also takes as features the position and the frequency. Achieving 44% of the auto-extracts matched the manual extracts. However, he adds other features:

- **Cue words** (bonus and stigma words). defined as a connective expression that links spans of discourse and signals semantic relations in a text. One example of bonus words would be "significant" and one example of stigma words would be "hardly" or "impossible".
- **Document structure** as a feature (for example, if the text is the heading, the title, etc).

With all that together, he does a linear combination on those 4 factors to do choose the most relevant factors.

deJong (1979-1982)[54] proposed a knowledge-based approach named Frump which was a slot-filling based on UPI new stories. Inputs matched to scripts based on manually selected keywords. This paper was highly influential although it was not really easy to translate it to other domains because it was difficult to apply. Since the proposed scripts turned to be missing for many inputs.

Paice (1990)[51] lists things that mostly failed. Such as syntactic criteria (Earl 1970), indicator phrases. He also mentions different problems which

extractive summarization methods can find. He states two main problems which are lack of balance, and cohesion. Which have been covered and studied across literature.

Brandow et al. (1995) [53] Worked with 41 publications from commercial news rather than scientific papers, which was the most common source used in literature until that point. His studies, shows that "lead" achieves acceptability of 90% whereas 74,4% "intelligent" summaries which basically implied that a human could do better. However, it was later found that this results are only given on certain genres of news were the journalist, gives in the first few words of a sentence.

In order to find out which were the most relevant words Brandow et al. (1995), used the term frequency * inverse document frequency ($tf*idf$). Note that lower $tf * idf$, implies least important words (stopwords). In order to generate the summarization, they used the following sentence base features: signature words, location, anaphora words, lenght of abstract. Regarding to their evaluation, they generate 3 different outputs: 60, 150 and 250 words length in a non-trask-driven evaluation, by asking users if those summaries are acceptable or not.

Kupiec et al. (1995) [42] First use of trainable techniques for sentences extraction, with a Naïve Bayes classifier to categorize each sentence as worth or not of extraction. Their objective was to get a 20% extraction from their 188 scientific journals. The features that they presented are:

- Length of the sentence, where if the sentence length is minor than 5 words is not included, ($|S| > 5$).
- Presence of uppercase words (except common acronyms).
- Use of thematic words.
- Set of 26 manually fixed phrases.
- Sentence position in paragraph.

The best combination that he found was using only position, cue and length with a performance of 84% precision for the 25% of the summaries. And for smaller summaries, 74% of improvement over the base Lead. Later on, Aone et al (1999)[52] also incorporated a naive-Bayes as but he used richer features

- Signature words

- Named-entity tagger
- Shallow discourse analysis
- Synonyms and morphological variants were also merged (accomplied by WordNet)

Summons (McKeown & Radev 1995) [43] is the first work on multi-document summarization. The input set in this case, comes from a different sources regarding to a same topic. His approach was a knowledge-based one, using MUC templates as information extraction to later apply text generation that contain the slot fillers from the MUC templates. It is quite extensive to explain everything that they performed. However, the following schema summarizes is it quite well

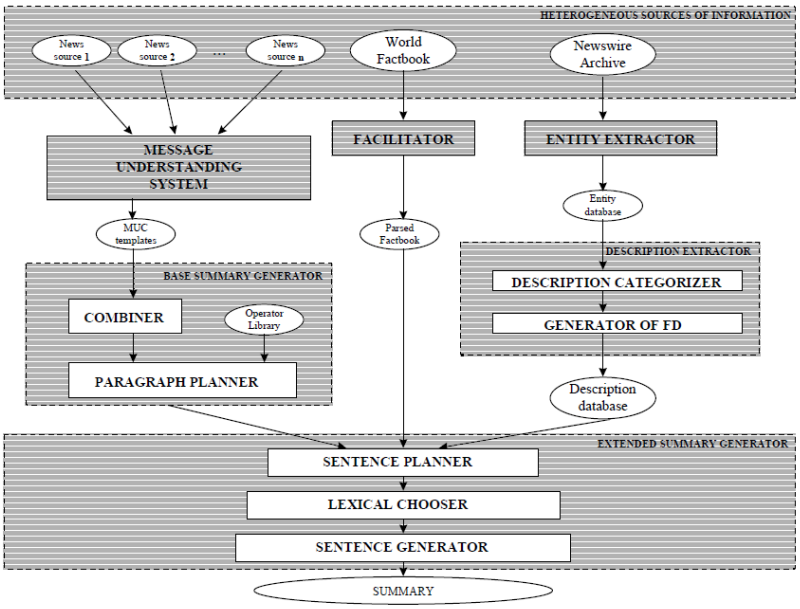


Figure 91: Multidocument schema

Lin and Hovy (1997) [44] studied the importance of sentence position feature, which significantly changes across domains. However, by investigating techniques in order to tailor the position method depending on the genre, they were able to produce the optimal position policy. This was possible since

they first measured the yield of each sentence against the topic keywords and then, they ranked the sentence positions by their average.

Mani/Bloedorn (1997,1999)[49][50] used a graph-based method for identifying similarities and differences between documents based on a single or a sequence of events. In their approach, nodes from the graph are content entities and relationships are the edges. The different kind of relationships that can be described were: same, adjacent, alpha, coref. To highlight the most important nodes, they use spread activation.

Barzilay M. Elhadad (1997)[48] they based their approach on lexical chains. The relationships from the chains were extracted from WordNet and they described 3 types of relations: extra-strong (e.g repetitions of the same words), strong (e.g WordNet relationships), medium-strong (link between synsets $>$ 1)

Marcu (1997-1999) [47] who took a different approach on summarization by basing his work on rhetorical structure theory (Mann and Thompson 1988). This approach basically focuses in that a phrase is composed by two parts: nucleus + satellite. The nucleus is the most important and relevant part, and the one that gives meaning to the second, while the second is the one that re-emphasizes and makes stronger that nucleus. By following this approach, the author splits a text to the point that he can get the K most relevant phrases by iterating thought a three of nucleus, satellites and a set of rules.

Carbonell and Goldstein (1998) [41] introduced the idea of maximal marginal relevance (MMR). Which explains that even we have two queries that are relevant to a same query, even that the two documents are very similar to each other, you do not want to show both to the user because the value that is added when showing the second one, the marginal value, is very small (diminishing returns). Therefore, what is prioritized is documents that even are not as relevant, but are diverse. To do so, they proposed a greedy method.

Lin (1999) broke away from the assumption that features are independent of each other by instead of using a naive-Bayes classifier, using decision trees. Some of the novelties from his paper are:

- **Query Signature:** normalized score given to sentences depending on number of query words that they contain.
- **IR signature:** score given to sentences depending on number and

scores of IR signature words included (the m most salient words in the corpus)

- **Average lexical connectivity:** the number of terms shared with other sentences divided by the total number of sentences in the text
- **Numerical data:** value 1 when sentences contained a number Proper name, Pronoun or Adjective, Weekday or Month, Quotation (similar as previous feature)
- Sentence length and Order

Moreover, Lin (1999) proves that the feature analysis suggested that the IR signature was a valuable feature, corroborating the early findings of Luhn (1958).

Mead [39] described a salience-based extractive summarization that could process multiple languages at the same time. The presented approach is a centroid-base method. The centroid score is calculated as the similarity between a given sentence and the centroid. It also combines the centroid score with features such as position. Detects and eliminates sentence redundancy using a similarity metric.

Conroy and O'Leary (2001) [40] which used hidden markov model in order to take into consideration the local dependencies between sentences. This is because when you are selecting phrases, it is required to know if the previous and the following ones, are also relevant so they will be included in the summary or not. In this particular case, the attributes used were position, number of terms and similarity to the document terms.

Osborne (2002) [45] used log-linear models to obviate the assumption of feature independence. Because the model tends to reject too many sentences, they decided to add a non-uniform prior to the model. The features that they included were: word pairs, sentence position, sentence length and native discourse. By native discourse, they refer to for instance: "inside introduction", "inside conclusion", etc.

Lexrank (Erkan and Radev 2004) [46] first method base on random walks, focused in lexical centrality. Therefore, if a sentence is likely to be visited during a random walk process, when the similarity graph corresponding to the sentences on the set processes then that sentence is worth to be included on the summary. To achieve so, they first represent the text as a graph, and then

they use graph centrality algorithms to check their most relevant sentences, then they apply clustering to segment text into units that correspond to certain themes.