UNIVERSITAT DE BARCELONA

DEGREE IN MATHEMATICS
DEGREE IN COMPUTER SCIENCE

Bachelor's Thesis

Article similarities using transformers

Author: **Rafael Beaus Iranzo**

Directors: **Paula Gómez Duran**
Dr. Jordi Vitrià i Marza

Departament de Matemàtiques i Informàtica

Barcelona, June 13, 2022

Abstract

The field of natural language processing is essential in today's data-driven world. In 2017 the Tranformers architecture was introduced based on the concept of attention from 2014. The effects of this new structure were already changing the paradigm when the language processing model BERT marked an inflection point, in 2018.

BERT makes use of the Transformers' parallelization to achieve a network that can be pretrained. In that pretraining, the model is able to learn how a language works on its own: by only feeding it with texts. An improved version came out shortly after, RoBERTa, after which most of the models were based.

In this thesis, we will focus on studying BERTa (a RoBERTa-based Catalan language model) with a dataset from the *Gran Enciclopèdia Catalana*. That analysis will include tasks to assess how does the model perform with real-world data.

The study aims to validate the quality of the resulting embeddings produced by the model in order to further use them to build an article retrieval platform. There, each article query could be related to those with similar information. The semantic textual similarity describes how alike a pair of sentences are and this will be a fundamental target for the designed experiments and development.

Finally, the results will be visualized and interpreted by using a simple front-end tool also created in this work.

2020 Mathematics Subject Classification. 68T50, 68T07, 91F20

Resum

El camp del processament del llenguatge natural és essencial per la societat orientada a les dades en la que vivim. El 2017 l'arquitectura Transformers fou introduïda basada en el concepte d'atenció, del 2014. Els efectes d'aquesta nova estructura estaven ja canviant el paradigma quan l'aparició del model de llenguatge BERT, el 2018, va marcar un punt d'inflexió.

BERT fa ús de la paral·lelització dels Tranformers per tal d'aconseguir una xarxa que pot ser preentrenada. En aquest preentrenament el model és capaç d'aprendre com funciona un llenguatge per sí mateix: només consumint textos. Poc després va sortir una versió millorada, RoBERTa, en la qual es basarien gairebé tots els models que apareguessin posteriorment.

En aquest treball, ens centrarem en estudiar BERTa (un model de llenguatge en català basat en RoBERTa) amb un conjunt de dades de la *Gran Enciclopèdia Catalana*. Aquesta anàlisi inclourà tasques per avaluar quin és el rendiment del model amb dades quotidianes.

L'estudi busca validar la qualitat dels embeddings produïts com a resultats del model i utilitzar-los per tal de construir una plataforma de cerca d'articles, on cada consulta d'article podrà ser relacionada amb aquells que tinguin una informació semblant. La *semantic textual similarity* (similaritat textual semàntica) descriu quan s'assemblen una parella de frases i serà un objectiu fonamental pels experiments i el desenvolupament.

Finalment, els resultats seran visualitzats i interpretats fent servir una pàgina senzilla; també creada en aquest treball.

Resumen

El campo del procesamiento del lenguaje natural es esencial para la sociedad orientada a los datos en los que vivimos. En 2017 la arquitectura Transformers fue introducida basada en el concepto de atención, de 2014. Los efectos de esta nueva estructura estaban ya cambiando el paradigma cuando la aparición del modelo de lenguaje BERT, en 2018, marcó un punto de inflexión.

BERT utiliza la paralelización de los Tranformers para conseguir una red que pueda ser preentrenada. Mediante este preentrenamiento el modelo es capaz de aprender cómo funciona un lenguaje por sí mismo: sólo consumiendo textos. Poco después salió una versión mejorada, RoBERTa, en la que se basarían casi todos los modelos que aparecieran posteriormente.

En este trabajo, nos centraremos en estudiar BERTa (un modelo de lenguaje en catalán basado en RoBERTa) con un conjunto de datos de la *Gran Enciclopèdia Catalana*. Este análisis incluirá tareas para evaluar cuál es el rendimiento del modelo con datos de uso diario.

El estudio busca validar la calidad de los embeddings producidos como resultados del modelo y utilizarlos para construir una plataforma de búsqueda de artículos, en la que cada consulta de artículo pueda ser relacionada con aquellos que tengan una información similar. La *semantic textual similarity* (similaridad textual semántica) describe cuánto se asemejan una pareja de frases y será un objetivo fundamental para los experimentos y el desarrollo.

Por último, los resultados serán visualizados e interpretados utilizando una página sencilla; también creada en este trabajo.

Acknowledgements

En primer lloc, voldria agraïr als meus tutors: la Paula i el Jordi per tot el suport lliurat al llarg d'aquest projecte. La seva ajuda ha estat essencial i les seves orientacions molt encertades. Especialment, voldria destacar del Jordi els seus coneixements dels articles publicats fins a la data; i de la Paula la seva facilitat entenent l'estat el meu treball, a més de la seva capacitat crear-me crisi existencials quan parlàvem del meu futur.

També voldria agrair la feina feta per tots els professors, siguin de la universitat o de l'escola. El seu ensenyament (tant acadèmic com personal) ha estat el que ens ha permés ser com som.

No podria oblidar-me dels meus companys. Sense ells segurament no hauria pogut arribar al punt on em trobo ara.

Finalment, a la meva família per la seva incondicional disponibilitat i ajuda.

Contents

Chapter 1

Introduction

Motivation and objectives

As humans, we tend to communicate a lot with each other. All this communication leads to the generation of vast amounts of information, invaluable in today's data-driven world. In AI, the field responsible for analyzing those interactions is called **Natural Language Processing** (NLP), which actually is the combination of Artificial Intelligence and linguistics (sometimes also called computational linguistics).

The main way humans communicate is via speech and text. NLP develops methods for understanding and processing those types of data. In fact, there are many chances you have already been using some of the most powerful NLP techniques although without really being aware.

Those applications, if done well, are almost always transparent to the end users. Amongst others, they include famous recognition assistants like Amazon's Alexa or Apple's Siri and all the chat bots in web pages, Google's multilingual translation, and even spam filters in your inbox. Stepping away from the end users and concentrating more on the data science approach, NLP is arguably one of the most used machine learning tools.

In 2018, the NLP field was struck with a disruptive language representation model: BERT [5], which used the Transformers [18] architecture. As soon as it was tested, it produced state-of-the-art results with many different NLP tasks. It emerged as a new approach where the model would be pre-trained (basically to understand a certain language) and, later, with specific fine-tuning, it will be capable of creating highly developed models for a wide range of tasks. It implied that you might be able to have your own model for many particular problems with little training. The one caveat is that the pre-training part need loads of documents, making it very resource intensive and time consuming. It only needs to be done once, but it is a mandatory step if you want to have a model in a specific language.

As you might expect, English is the language in which there have been more models trained. In fact, some under-resourced languages are struggling to collect enough clean information to train a language-specific model. Fortunately enough, a team at the Barcelona Supercomputing Centre (BSC) released last year (July 2021) BERTa [2], a Transformer-based model in Catalan. We will focus our thesis on this particular model.

Our first objective with this thesis is to understand how do Transformers (and the Neural Networks behind them) work. Our second goal is then to evaluate how good the Catalan model is on a particular dataset when evaluated in the task where it was trained. After that, we also aim to extract the Semantic Textual Similarities (STS) between different articles. Finally, we would like to retrieve the k-Nearest Neighbors (k-NN) for each article while letting the user interact with the results.

Essentially, we will be testing and applying a Transformer-based model to a specific use case. In our circumstances, the final application will be to use the results obtained from this thesis in order to analyze and visualize relevant articles for a given article query.

Structure

This thesis is structured in the following manner:

- **Related work**: In this section we will cover the main state-of-the-art approaches to finding articles similarities by using transformers.

- **Language representation models**: In this block we will review the theoretical background needed before diving into solving the problem at hand, by covering the formal aspects of neural networks from a mathematical point of view.

- **Development and experiments**: In this section we will look at the articles similarities problem, focusing on the methodologies we have implemented on our practical development with the transformers model. The process is presented, comprehending all the parts of the operations as well as the results obtained.

- **Conclusions and further work**: Conclusions of the project and ideas for further work regarding article similarities and transformers.

Planning

It was at the beginning of the academic year when I first contacted my tutors and agreed on the topic, with no previous knowledge neither of deep learning nor on using transformers. A few weeks later I started to work as an intern and, by chance, the project assigned to me dealt in some way with transformers and other NLP techniques. When the time to begin my thesis finally arrived in February, I had been working for a while with those technologies at a very practical level. However, my lack of theoretical background meant that quite a bit of research had to be done before I could confidently plan and tackle this project.

The methodology followed during this project was SCRUM, where one-week sprints were made. Then, weekly meetings were held to discuss the progress and the next steps.

The conducted experiments were mostly sequential: requiring one step to be finished before initiating the next one. However, some overlapping usually occurred when the final improvements were being made to the current iteration of the subject, while also starting to investigate the next topic.

The development of this thesis took place during 4 months: between February and June of 2022. A broken-down, more detailed view of the timeline can be seen at the Figure 1.1.

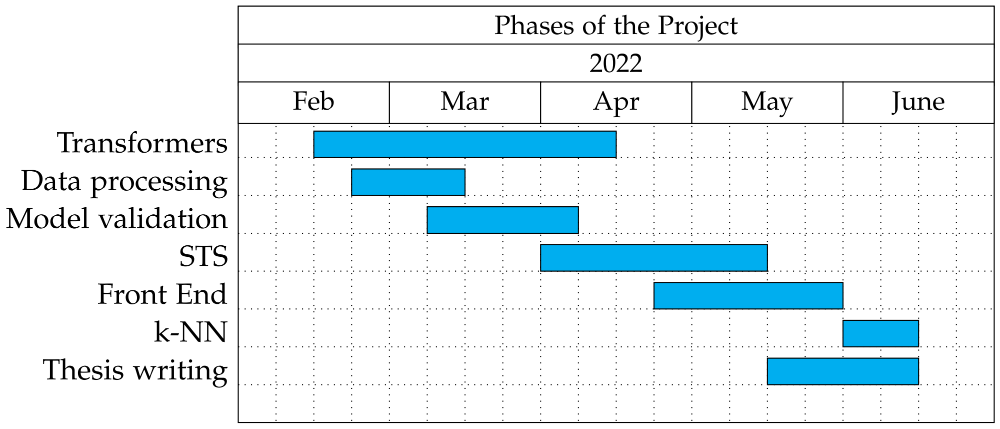


Figure 1.1: Timeline of the project

Chapter 2

Related work

Natural Language Processing (NLP) is a field that has undergone many changes in its history. The latest revolution took place when unsupervised learning algorithms were used. With that approach, the need to have labelled data in order to train the models disappeared and virtually any text source could be used for that purpose.

The current state-of-the-art models follow a clear progression building upon previous research. Focusing on the most relevant work regarding the topic we will cover, it was in 2014 when the introduction of the attention mechanism [3] began.

Despite being understood that attention improved the existing models, the implementations of this mechanism were done on top of a recurrent or convolutional NN until that moment. Attention connected two important parts of the network structure (the encoder and the decoder) hence highlighting what were the most relevant segments of the text: therefore noticing the appearance of complex relationships within the sentences.

It was not until 2017 when the renown paper "Attention Is All You Need" [18] disrupted the machine learning community by introducing the Transformer network architecture. It significantly improved the existing benchmarks and showed the path to follow.

Using the Transformer architecture, a Google AI Language team introduced BERT in 2019 [5]. BERT (Bidirectional Encoder Representations from Transformers) established a new state-of-the-art baseline in machine translation tasks and other baselines. It took the best performing models and kept only the encoder-decoder structure: making the result more parallelizable and, thus, requiring less time to train. Less than a year later, the RoBERTa model [9] appeared as an improvement to the BERT pretraining method, taking it even a step further.

That pre-training step is an essential task. It allows the model to understand a given language by only feeding it with massive amounts of texts. As it could be expected, it is a costly task; but once it has been done, the model can easily and rapidly fine-tuned for the completion of downstream tasks with great results.

The vast majority of these pre-trained models has been done in English. Some other models have multilingual capabilities, although it has been shown that language-specific models outperform them in nearly every task [2]. With that in mind, it comes as no surprise that every language community wants to have their own trained model. And that is exactly what a team at the BSC (Barcelona Supercomputing Centre) did for Catalan language in July 2021, when they released BERTa[2].

Chapter 3

Language representation models

3.1 Embeddings

When working with any kind of object in machine learning, it is useful to have a representation of that object. In our case, we need to find a way of transforming the sentences (groups of discrete words) into a more manageable form. One of the most *manageable form* that we could think of would be a continuous vector.

That is precisely what we call an **embedding**: a mapping of a discrete variable to a vector of continuous numbers. This way, we can describe the words as a real-valued vector of arbitrary dimension rather than having to stick with the huge (thousands or millions) of dimensions that would be needed in the case of a sparse word representation, like a one-hot encoding.

The fact that a word is now represented as an embedding belonging to a vector space gives us a hint on how we can take advantage of it. If done correctly, similar words should end up quite close to each other. Moreover, linear operations can be done such that we are able to infer a relationship between groups of words.

Note that the word embedding could also be referring to a mathematical embedding, where an instance of a mathematical structure is contained within another group. We could see that our usage is merely a less abstract one, but its meaning is still reminiscent of the original one.

Creating embeddings

How those embeddings are created is a really interesting problem, to which there are very different approaches. We will explore some of them, depending on certain parameters.

The main parameter regarding the embeddings is whether or not it takes into account the **context**, i. e. if the same input word will always correspond to the same output vector or if the model will consider the surrounding structure and the rest of the sentence to produce the output vector.

Let us dive into an example just to make it clear. For instance, we will take the word *bank* and use it in two different scenarios:

1. My brother-in-law works at the bank next to the post office.

2. I enjoyed a great run next to the river bank yesterday.

For us, the readers, it is really clear that the two meanings of bank in the previous sentences are not the same. However, when creating each embedding: if we were to simply look at the individual words and replace each of them for a vector, the different meanings would be lost. In the context relies all that information.

As obvious as it may seem, using the words context to create the embeddings is (by far) a trivial task. Some models opt for the simplicity of not caring about that information at all while others make full use of it. Several approaches have been tried since the early days of NLP. Using a variety of techniques, many models have been created. In the Figure 3.1, we can see some of them.

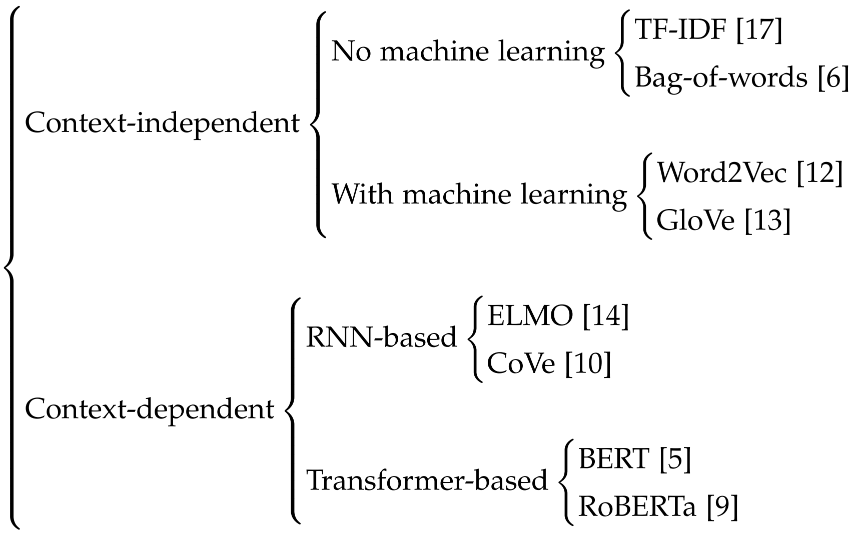


Figure 3.1: Classification of some NLP techniques

Since our aim is to understand how BERTa works, we will first study the model it was based on: RoBERTa.

RoBERTa, in particular, is a retraining of BERT. And, finally, BERT is the model that disrupted the whole community by obtaining state-of-the-art results on eleven natural language processing tasks.

BERT stands for **B**idirectional **E**ncoder **R**epresentations of **T**ransformers. We will begin by understanding what a transfomer is and why it is useful, but since the tranformer is an architecture of a neural network, let us have an introduction on the topic.

3.2 Neural Networks

A **Neural Network** (NN) (sometimes also called Artificial Neural Network) is a serie of algorithms belonging to machine learning, in particular to deep learning. Its aim is to recognize underlying relationships in a set of data and solve common problems.

3.2.1 Structure

NNs are especially well known for having and structure that was inspired by how the human brain works: a collection of interconnected layers of small units that perform mathematical operations. Formally speaking: it forms a directed, weighted graph.

We will begin by defining the different parts that make up the structure:

- **Neuron**: It is the main building block of the NN. It takes in certain inputs and produces a single output. It can also be called *node* or *perceptron*.
- **Layer**: Name received by a collection of neurons operating together at a specific depth within the NN. There are three different types:
  - **Input layer**: First layer of the network. The neurons inputs come directly from the raw data.
  - **Hidden layer**: Internal layers of the network.
  - **Output layer**: Last layer of the network. The outputs of this layer correspond to the final values of the network. They accomplish the task, usually by having classifications or other signals to which input patterns may map.
- **Weight**: Value defining how strong is the connection between neurons of two consecutive layers.
- **Bias**: Value added to the sum of the product between the input values and their respective weights of a single neuron. It is used to accelerate or delay the activation of a given node.

- **Activation function**: Defines how the weighted sum of the input is transformed into an output from a layer of nodes of the network.

Definition 3.1. A neural network **parameter** is a coefficient of the model which is chosen by the model itself. A neural network **hyperparameter** is an element that affects the model architecture or performance, but is pre-set.

On the one hand, the weights and the biases of each neuron of the network are both parameters that the model will try to optimize to better perform at the given task. Meanwhile, the number of layers and the number of neurons on each layer are hyperparameters that need to be chosen by the designers of that specific NN.

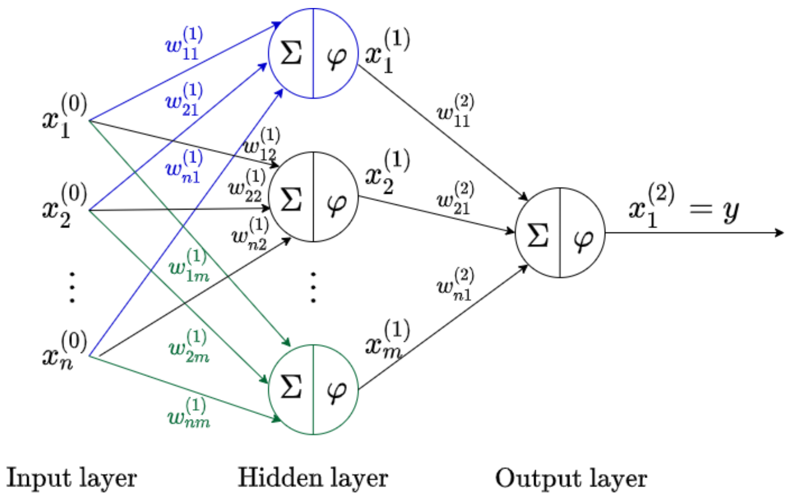


Figure 3.2: Neural network structure

In the Figure 3.2 we can see a neural network structure. The input vector of dimension n is mapped to the input layer. Afterwards, each of the m neurons in the hidden layer is influenced by the weighted sum of the input layer neurons, and slightly modified with a bias. An activation function φ is then used to map the previous result to the output layer, consisting in only one neuron. Ultimately, the output of this final layer is what the neural network will predict.

3.2.2 Activation functions

Activation functions are an essential part of the NN that must not be overlooked. Their choice will control how well the model learns the training dataset. These functions may not be linear, otherwise the network capabilities would be very limited.

Definition 3.2. An **activation function** is a non-linear function $\varphi : \mathbb{R}^n \to A \subset \mathbb{R}^n$.

A few have been used and their complexity has evolved over time. We will show a couple of the most used ones..

Definition 3.3. The **Rectified Linear Unit (ReLU)**. This activation function is linear only for $x \geq 0$, and it is given by

$$ReLU(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases} = \max\{x, 0\}$$

Definition 3.4. The **logistic** or **sigmoid function**. This activation function approaches 1 as $x \to +\infty$ and 0 as $x \to -\infty$. It is given by

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Definition 3.5. The **softmax function** is given by

$$softmax(x) = \left(\frac{e^{x_1}}{\sum_{i=1}^{n} e^{x_i}}, \ldots, \frac{e^{x_n}}{\sum_{i=1}^{n} e^{x_i}} \right)$$

It can be interpreted as the generalisation of the logistic function for n dimensions. In this context, the softmax activation function is frequently used since it acts as a smooth arg max.

3.2.3 Learning

As stated before, the aim of a NN is to capture the relationship between its input and the supposed output. When a NN is initialized it is clueless on how to solve that problem, therefore it must be trained.

For starters, we will need a metric to see evaluate the output of the NN.

Definition 3.6. A **loss function** $\mathcal{L}$ is a function that measures the proximity between the NN predictions and their target value.

Then we can call **learning** to the process of changing the parameters of the network so that they produce the minimal loss.

Loss functions $\mathcal{L}$ can be any metric that measures proximity. However, differentiable functions are typically used to take advantage of interesting properties of functions that allow the usage of methods for minimizing. One of such methods is the Gradient Descent.

3.2.4 Gradient Descent

Let $f : \Omega \to \mathbb{R}$ be a differentiable function. Now, consider the general unconstrained optimization problem:

$$\arg\min_{x\in\Omega} f(x)$$

Definition 3.7. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function. We define **the gradient of** f, $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$, at the point $x = (x_1, \ \ldots, \ x_n)$ as the vector composed by its partial derivatives:

$$\nabla f(x) = \Big(\frac{\partial f}{\partial x_1}(x), \ \ldots, \ \frac{\partial f}{\partial x_n}(x)\Big)^T$$

Remark 3.8. Note that at any point x, the direction $-\nabla f(x)$ is the direction of the fastest decrease of f at x.

Iterative methods try to converge to the solution (the local minimum x^*) by generating a sequence of values $x^0, x^1, \ldots$. **First order methods** must generate that sequence using only the function value and its gradient at different points of Ω.

The Gradient Descent (GD) is a first order method that generates each next point by taking a step of size $\eta > 0$ in the direction of the antigradient $(-\nabla)$. GD is arguably the simplest and most intuitive first order method. [1] In the Figure 3.3 we can see an example of how does it work.

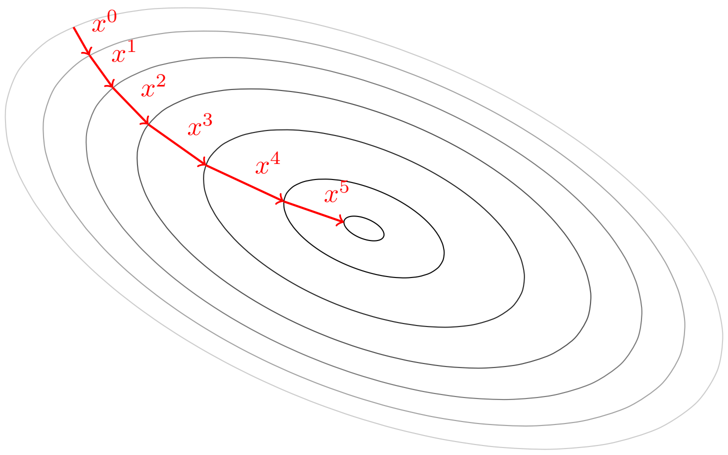


Figure 3.3: A gradient descent example

Formally speaking: We start at a random point, x^0. Then, at iteration k the next point (x^{k+1}) is defined as

$$x^{k+1} = x^k - \eta \frac{\nabla f(x^k)}{\|\nabla f(x^k)\|}$$

The step size η is called the **learning rate**. It has a great impact on the results. Smaller learning rates will take longer to arrive to the minimum. On the other hand, larger rates may be too big and miss the minimum: that is known as the *overshooting problem*.

One of the first improvements that can be made to the algorithm is to progressively change the value of the learning rate: making it larger when starting and decrease it as the solution converges. This technique receives the name of **learning rate decay**.

The GD is a greedy, local algorithm that can only be expected to converge to a local minimum of f. Despite the learning rate decay optimization, the computation of the gradient could still be computationally demanding. Unfortunately, supervised learning is one of the most common scenarios where this problem arises.

Stochastic Gradient Descent

Remember: when learning, the aim of the NN is to minimize the loss function $\mathcal{L}$. We are doing so by calculating the average difference of the actual values and the target ones for all the samples (or a subsample) of size r. Calculating the theoretical loss would mean to run the calculations for each one of the samples, on every step of the gradient descent. That operation is not only impractical but, fortunately, not necessary.

The gradient is a vector that results by summing the r gradient-vectors (one per sample). We can deem reasonable to presume that no particular sample has too much domination in the resulting gradient. It is then not a bad assumption to infer that a good approximation of the gradient could be obtained by taking a random subset of $s < r$ samples. Then, we would need to calculate s gradient vectors, add them, and scale the resulting gradient proportionally.

The process described above is what is called the **Stochastic Gradient Descent** (SGD). It is the most widely used optimization method in the machine learning community.

Convergence of the gradient descent

As it could be expected, some conditions must be met to guarantee that the GD algorithm converges towards a certain point $x^* \in \Omega$.

Remark 3.9. The previously defined concept of transition from one point to the next one

$$x^{k+1} = x^k - \eta \frac{\nabla f(x^k)}{\|\nabla f(x^k)\|}$$

can be interpreted as an update function $G_{f,\eta} : \Omega \to \Omega$

$$G_{f,\eta}(x) = x - \eta \nabla_x f$$

Definition 3.10. Let (M, d) be a metric space. Let $f : M \to M$ be a function. f is a **contraction mapping** if

$$\exists c,\ 0 \le c < 1 :\ d\big(f(x), f(y)\big) \le c \cdot d(x, y), \qquad \forall x, y \in M$$

Proposition 3.11. If the gradient update is contractive, then the GD converges to a stationary point x^*, i.e., $\nabla_{x^*} f = 0$

We can even go a step further and define a couple concepts that can help us determining the rate of convergence.

Definition 3.12. Let X be a convex subset of $\mathbb{R}^n$ space. Let $f : X \to \mathbb{R}$ be a function. f is **convex** if

$$\forall t,\ 0 \le t < 1 :\ t \cdot f\big(x + (1-t)y\big) \le t \cdot f(x) + (1-t) f(y), \quad \forall x, y \in X$$

Definition 3.13. A continuously differentiable function $f : X \to X$ is β-smooth if its gradient, ∇f, is β-Lipschitz. Equivalently:

$$\|\nabla f(y) - \nabla f(x)\| \le \beta \|x - y\| \quad \forall x, y \in X$$

Definition 3.14. Let (M, d) be a metric space. A function $f : M \to M$ is **Lipschitz** if

$$\exists K \le 0 :\ \|f(x) - f(y)\| \le K \cdot \|x - y\|, \qquad \forall x, y \in M$$

Proposition 3.15. If f is convex and β-smooth, and a step of $\eta \le 2/\beta$ is used, then the k-th iterate, x_k, of GD satisfies

$$|f(x_k) - f(x^*)| \le \frac{\|x_0 - x^*\|^2}{2\eta t}$$

Thus, GD has an $O(1/k)$ rate of convergence.

3.3 Transformers

The Transformer is a network architecture that was introduced in the renown paper Attention Is All You Need [18] from December 2017. The distinguishing trait of this model architecture is that it dispensed with recurrence and convolutions, and it was solely based on the attention mechanism.

Using an attention mechanism to connect the encoder and the decoder was something that the best models already did, but the dominant ones were based on complex recurrent or convolutional neural networks. This apparently simple change is what led to having better results, making the model more parallelizable and require less training time.

3.3.1 Attention

In the case of NLP, the idea behind the concept of attention is to have a way of telling the transformer how important is the relationship between the words in the same sentence. Therefore, being able to find what the most important parts are.

An attention function can be described as mapping a query and a set of key-value pairs to an output. A compatibility function of the query with the corresponding key is used to compute the weights assigned to each value, and then the output is computed as a weighted sum of the values.

We will call Q, K, V to the matrices associated with the query, keys and values respectively. Let the dimension of the queries and keys be d_k, and the dimension of the values be d_v.

The steps for obtaining the Scaled Dot-Product Attention are:

1. Compute the dot products of the query with all the keys.
2. Divide each result by $\sqrt{d_k}$.
3. Apply a softmax function to obtain the weights on the values.

Since the vectors will be processed in batches, from now on we will consider the attention mechanism for each batch. Understood as a mapping, it is defined as

$$Attention(Q, K, V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

3.3.2 Model architecture

Now we will dive deep into the concepts that the Transformer model is made of: mainly the Encoder-Decoder structure and its main building blocks: the positional encoding, the multi-head attention and the feed forward. The structure of the architecture can be seen in Figure 3.4.

The encoder is responsible of mapping an input sequence of symbol representations $(x_1, \ldots, x_n)$ to a sequence of continuous representations $z = (z_1, \ldots, z_n)$. Then, given z, the decoder will generate the output sequence $(y_1, \ldots, y_n)$.

For each sequence of words that the Transformer is fed, a series of operation are performed. Let us briefly go over them.

Firstly, using a dictionary (vocabulary) and an embedding matrix, each of those words gets their embedding representation. That embedding has a dimension of 512. Afterwards, they get information added via a positional embedding before being fed to the Encoder-Decoder block.

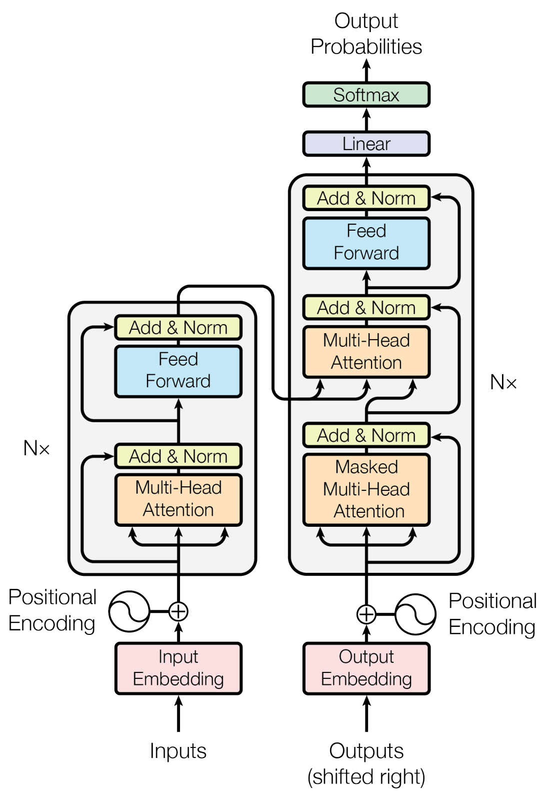


Figure 3.4: The Transformer model architecture [18]

Positional Encoding

One of the main advantages the transformer model has is that it splits the input in what is called **batches**. Batches are processed in parallel and in the same way. Therefore, we must find a way of conveying the original position of the input on each batch.

In the first iteration of the structure, a position vector was added to each batch: with the objective of having the weights matrices take into account that position, in case that was needed [18]. That way of adding a fixed position vector is called **positional encoding**. If, instead of being fixed, the added vector is also trained, we call it **positional embedding**.

Encoder

Instead of using a single self-attention function, numerous self-attention heads are used: each with a distinct weight matrices for the queries and keys. Multi-head attention allows the model to simultaneously attend to information from several representation subspaces at different positions. In the original study, the

Transformer had eight parallel attention heads.

The resulting outputs are then added and normalized and passed to the feed-forward block.

The feed-forward block consists of a fully connected feed-forward network, made up of two linear transformations with a ReLU activation in between.

What we have defined until now is an encoder. The encoding component from the Transformer is a stack of 6 encoders. They are all identical in structure, yet they do not share weights.

Decoder

The decoder section of the Transformer model in composed by mostly the same blocks as those found in the Encoder and previously explained. Like the encoder component, it is also composed by a stack of 6 identical layers.

Since this block is not essential for following rest of the thesis. In case that the reader wanted to keep learning about the decoder block, the original paper does a good job explaining the details: [3].

3.4 Language models evolution

As one might expect, scientific research builds upon itself to keep improving. There is no doubt that we are *standing on the shoulders of giants*. Language models are no exception to that rule.

In this section, we will briefly study the technical developments that built the tools that will be used in the thesis.

3.4.1 BERT

BERT **B**idirectional **E**ncoder **R**epresentations from **T**ransformers is a language representation model introduced in May 2019 [5]. It was the first of its kind and generated such an impact that new models are still produced with the basis layed by BERT.

Prior to the appearance of BERT, the state of state-of-the-art NLP model was **G**enerative **P**re-trained **T**ransformer (OpenAI GPT) [15].

GPT [15], introduced minimal task-specific parameters, and it was trained on the downstream tasks by simply fine-tuning all pre-trained parameters. The reason behind that was because it was unidirectional. Having that property, it forced the authors to use a left-to-right architecture: where every token can only attend to previous tokens in the self-attention layers of the Transformer.

The main improvement BERT does is to change the pre-training objective and being freed from having to be unidirectional.

The pre-training objective used is called Masked Language Model (MLM). It was inspired by the Cloze task: where some randomly selected tokens are masked and the goal was to predict them only from their context. In addition to the MLM, a task where the two sentences were presented to the model and it had to predict if the second followed the first was used.

3.4.2 RoBERTa

After closely studying the process through which BERT had been developed, several improvements were found. Apparently, some design choices had been overlooked and hyperparameter choices had too big of an impact in the final results. Those were the claims that the RoBERTa[9] paper exposed and backed up with data.

On top of that, they were able to refine the detected issues and they developed an improved version of BERT. As the paper and model name suggest: *A Robustly Optimized BERT Pretraining Approach*. With the enhanced training recipe for BERT models (RoBERTa) the performance of all post-BERT methods was matched or exceeded.

It came as no surprise that from then on, RoBERTa became the new language model standard.

3.4.3 BERTa

BERTa training was trained following the RoBERTa [9] base model. The Next Sentence Prediction task, present in the original BERT, was omitted in this case since it was auxiliary. Therefore, only the masked language modeling was used as the pre-training objective.

The paper is also important because in it, high-quality Catalan corpus (the largest to its date) was released with an open licence. On top of that, annotated corpora for multiple NLP tasks was created and released as well. That makes future work in the Catalan language much easier by lowering the barrier to entry.

3.4.4 Sentence Transformers

One of the most important things that BERT is lacking is the ability to compare pairs of sentences. In reality, BERT can compare pairs of sentences but only one at a time: since they must be fed to the model at the same time. It means that to compare n sentences we must use the model n^2 times. In practice, that *small* caveat makes this feature unusable.

That shortage is clearly impractical and it was also the reason why the SentenceBERT (SBERT) model [16] was created. SBERT is a modification of the BERT network able to derive semantically meaningful sentence embeddings. SBERT was built using siamese networks that allow it to process two sentences in the same way, simultaneously. These two twins are identical down to every parameter, which allows us to think about this architecture as a single model used multiple times.

BERT is the base of this model, to which a pooling layer has been added. This pooling layer makes SBERT able create a fixed-size representation for input sentences of varying lengths. The original BERT model already generates a CLS token summarizing the information from its tokens that was also considered in the study.

Another alternative would be to use an aggregation of the individual embedding for each word as the sentence embedding. That is an interesting experiment and different aggregation functions could be studied to find which one captures best the sentence as a whole. Furthermore, that method might be used as a benchmark when comparing it against SBERT.

It was found that SBERT had only minor differences compared to SRoBERTa for generating sentence embeddings. For our needs, we will use BERTa [2] as the starting point with all its vocabulary learnt in the pre-training.

3.5 Hugging Face

Hugging Face Transformers [2] is a library that provides thousands of pretrained models to perform tasks on different modalities such as text, vision, and audio. It first started in 2020 and had a paper follow up the launch [19].

Hugging Face Transformers provides APIs to quickly download and use those pretrained models on a given text, fine-tune them on your own datasets and then share them with the community on our model hub.

Transformers is backed by the three most popular deep learning libraries — Jax, PyTorch and TensorFlow — with a seamless integration between them. In our case, we benefited from using PyTorch. It provides a useful layer of abstraction to the usage of the GPU, very useful for working with large batches of data.

As stated in the introduction the chosen language can be considerate under-resourced.

There is clearly a difference in the number of models if we group them by language. Bear in mind that the multilingual models are also counted towards each of the languages it comprehends.

[2] `https://huggingface.co`

The top 4 languages (and their respective number of models) are: English (6568), Spanish (602), French (496) and German (415). Catalan sits in the 23rd position by having 107.

Chapter 4

Development and experiments

4.1 Problem definition

The aim of these experiments is on the first place to evaluate how does the BERTa [2] model for Catalan perform with our dataset. Then to use the contextualized sentence embeddings to represent a set of articles from which we will extract the approximate nearest neighbours. In the following sections, a detailed explanation on how each objective was planned and resolved is given.

4.2 Experiments environment

The entirety of the code of this project has been written in Python, in a Jupyter notebook environment. For the NLP processing, the transformers library has been used. For the neural network testing, we took advantage of the Tensorflow library. Moreover, we used the scientific computing library NumPy and Pandas, as well as Plotly for data visualizations.

Neural networks training is a computationally expensive and time-consuming process. Therefore, deep learning models are generally run on GPUs, with which we can achieve much shorter performance times. In our particular case: we did not train any model, but when doing certain calculations we made good use of the free GPU instances offered by the Kaggle environment.

The code of this project can be found at the following GitHub repository: `https://github.com/t1emp0/tfg-transformers`.

4.3 Gran Enciclopèdia Catalana dataset EDA

For the whole experiment, we used the Gran Enciclopèdia Catalana (GEC) dataset. It consists on 33 files, each one with different articles inside. The files provided from the GEC database, but it is not fundamental because they could also have been scraped from the web, since all the information is publicly available.

Each file in the dataset contained 107 columns and a certain number of rows. Since the only columns that we were interested in were the Title and the Body (text), we reduced each dataframe to only those two, and we looked at how many articles are there inside each file: before and after dropping all the empty ones.

Un any de COVID-19

```
    Antecedents i inicis de la pandèmia\n\nLa pandèmia de la
COVID-19 causada pel coronavirus SARS-CoV-2\xa0és filla de la
globalització. La mobilitat de persones, ...
```

Figure 4.1: A sample of a title and article (just a few words of it)

Figure 4.2 shows a plot describing that information. The blue bars represent the number of articles before filtering, the red bars stand for the number of articles after filtering the empty ones, and the green bars depict the difference between those two numbers. The red line is the mean number of filtered articles per file (5187). The total number of filtered articles is 171186.

Please note that the number of articles (y axis) is in logarithmic scale, in order to better show the wide range of the article numbers on each file ($\sigma = 18840$).

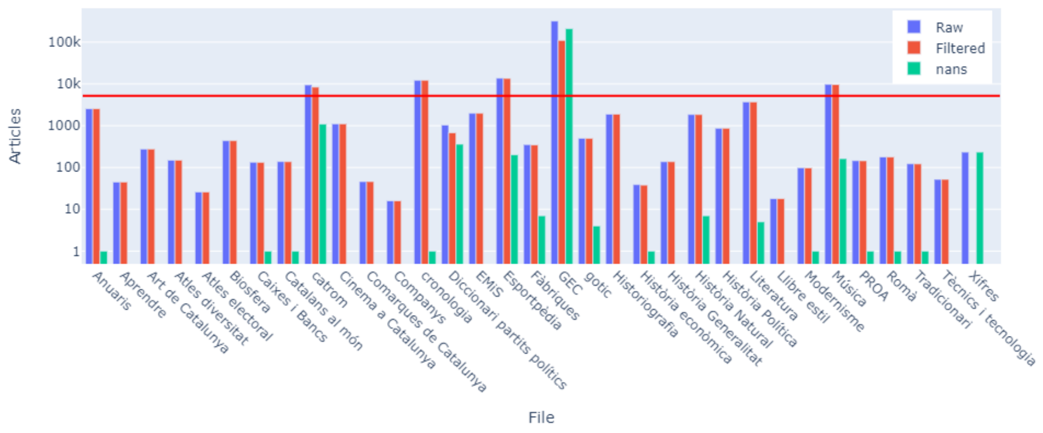


Figure 4.2: Articles per file before and after filtering, with the difference

Then we proceeded to explore the number of words per article. Now we don't care anymore of which file do the articles come from, we joined all of them and proceeded to count. We display the histogram obtained from those calculations in the Figure 4.3. The red horizontal line is showing the mean value (341.5). Note that, again, the y axis has a logarithmic scale.

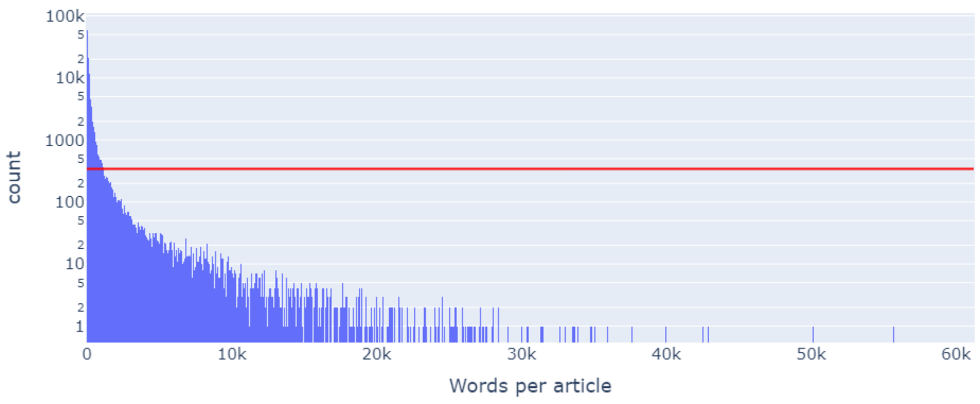


Figure 4.3: Number of articles grouped by words per article

From looking at the graph, it is obvious that most of the articles are in the shorter range of the spectrum. Just to give some more evidence of that fact, we show a description of the distribution. In the Table 4.1, a summary can be seen.

Measure	mean	std	min	25%	50%	75%	max
Value	341.5	1389	1	34	78	178	61221

Table 4.1: Number of articles grouped by words per article

Dataset corrections

Since we are working with real world data and not any kind of competition prepared datasets, we need to make sure the compatibility is right. That includes checking that the text is harmonized and that there are no encoding problems.

If you were to carefully read the article quote shown in Figure 4.1, you might see that there are some strange characters. The **\n** character represents a line break, but something else is encoded in the **\xa0** sequence. It represents a non-breaking space. To deal with it, and any other similar problem that may arise, we will need to understand how Unicode normalization works.

Unicode normalization

Unicode is a standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems. It is responsible of defining the UTF-8 encoding, among others.

Some Unicode characters can be written in different ways, but in practice they are the same one. That phenomenon is called **equivalence**. It can happen when two characters combined form an existing one (ç ↔ c+◌̧), or when the ordering of the combining marks is altered (q+◌̇+◌̣ ↔ q+◌̣+◌̇). There also exists the equivalence when characters or sequences represent the same abstract character but may have different visual appearances (① → 1).

There are a few ways of dealing with this issue. The standard practices are brought together in the Unicode Normalization Forms.

Unicode Normalization Forms are "formally defined normalizations of Unicode strings which make it possible to determine whether any two Unicode strings are equivalent to each other".[3] There are four Unicode Normalization Forms. In order not to extend ourselves too much, we will just focus on **NFKC**, the recommended form for our use case.

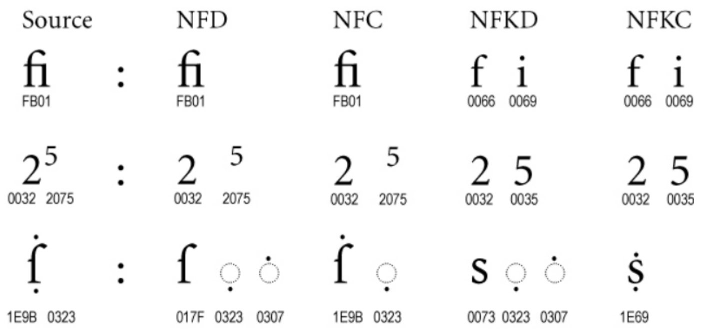

Source
NFD
NFC
NFKD
NFKC
ﬁ : ﬁ ﬁ f i f i
FB01 FB01 FB01 0066 0069 0066 0069
2⁵ : 2 ⁵ 2 ⁵ 2 5 2 5
0032 2075 0032 2075 0032 2075 0032 0035 0032 0035
ẛ̣ : ſ ◌̣ ◌̇ ẛ ◌̣ s ◌̣ ◌̇ ṩ
1E9B 0323 017F 0323 0307 1E9B 0323 0073 0323 0307 1E69

Figure 4.4: How the diferent Unicode Normalization Forms act [3]

NFKC is the 4th Normalization Form and stands for Compatibility Decomposition, followed by Canonical Composition. In the NFKC form, many formatting distinctions are removed, as it can be seen in the Figure 4.4 . The "fi" ligature changes into its components "f" and "i", the superscript formatting is removed from the "5", and the long "s" is changed into a normal "s".

By applying the NFKC to each one of the texts, we obtain the homogenization we were looking for. And, therefore, we do not need to worry anymore by those characters like **\xa0** since they will all have been replaced by their canonical equivalents.

[3]Obtained from the Unicode website: `https://unicode.org/reports/tr15/#Norm_Forms`

Apostrophes

Another issue that emerged was that some apostrophes were **straight apostrophes** while others were **smart apostrophes**: they curl in the direction of the accompanying text. The smart ones are represented by the character U+2019 *right single quotation mark* ’ while straight ones use the character U+0027 *apostrophe* '. There is no problem when using one or the other, but in our case they were mixed.

An easy fix we found was to replace all of them for straight apostrophes. This is due to the fact that smart apostrophes might also be used to quote some texts, and this way we prevented any confusions. Moreover, by doing so, the model will have an easier time dealing with them since they belong to the vocabulary and can be interpreted as a single token.

4.4 Model evaluation

The aim of the evaluation is to find out how does the BERTa model [2] perform with the GEC dataset explored in the previous section.

4.4.1 Task

The assignment with which the model was evaluated was an open cloze test. It is the same task that the model undertook in its training and, if done properly, should give quite good results.

The open cloze test is one where we hide a word from a sentence and ask the model to supply what word goes in that place as a test of its ability to comprehend text. The model will then return a series of predictions with its guesses. By default, it will return 5 predictions of words and their accompanying confidence scores.

Those confidence scores on their own are not enough to measure how well the model is performing. With the object of correctly determining the quality of the predictions, some particular metrics were used.

4.4.2 Metrics

The following measurements were used to assess the quality of the desired transformers model detectors:

- **Hit rate @ k.** It measures the fraction of words correctly predicted in the top k results. Since there are 5 predictions, the hit rate at 1, 3 and 5 will be taken.
- **Confidence.** The confidence score given by the model to the first prediction (the most likely one).

- **NDCG score.** Stands for Normalized Discounted Cumulative Gain: the result of the sum of the position of the predicted word in the order of predicted words, after applying a logarithmic discount. It is frequently used in recommender systems but a little unknown outside of that niche. We will dedicate a brief section to fully understand how it works.

After each individual guess has been evaluated, all the previous metrics will be calculated. Afterwards, an average across all the results from the predictions in the sentence will be considered. Finally, the global average between all the sentences in the run will be performed and that value will be our output.

As you can see, all metrics will return a value between 0 and 1, being 1 the best score and 0 the worst.

NDCG

The Normalized Discounted Cumulative Gain (NDCG) metric is a measure of ranking quality. As its name implies, it is the normalized version of the Discounted Cumulative Gain (DCG). DCG, at its turn, originates from an earlier (more primitive) measure called Cumulative Gain (CG). Where, using the relevance of each result, it computes the relevance of a query. [7]

Definition 4.1. Given a list of recommendations where each item has an associated relevance, we define the **Cumulative Gain** (CG) as

$$CG := \sum_{i=1}^{n} rel_i \ ,$$

where rel_i is the relevance of the result at position i.

CG is only considering the relevance of the list as a single whole, not taking into account the order in which the results are appearing. If we assume that the earlier a relevant result appears, the more useful it will be; a discount should be introduced to improve CG.

Definition 4.2. Using the same premises and notation as in the Definition of CG, we define the **Discounted Cumulative Gain** (DCG) as

$$DCG := \sum_{i=1}^{n} \frac{rel_i}{log_2(i+1)} = rel_1 + \sum_{i=2}^{n} \frac{rel_i}{log_2(i+1)}$$

As shown in the formula, the discount penalizes if the relevant results appear in later positions on the list. It can be seen that for a list with only one relevant result, it would penalize iif that document is not in the first position.

Finally, NDCG is calculated by dividing the DCG by the best possible score. The optimal score is obtained by ordering the given query: forming a list in which all the relevant results are monotonically decreasingly sorted.

In our particular case, we only have one relevant document with an associated relevance of 1. Therefore, some simplifications can be done to optimize this metric.

- The ideal DCG score will always be 1: no need to calculate it nor divide in the NDCG. Meaning that NDCG will be equal DCG.
- We only have one relevant result: no need in the summation of DCG.

Bearing the above deductions into consideration, we present the optimized formula used in our experiments. By calling the result we are looking for *item* and its position in the list *index*, we have:

$$NDCG = \begin{cases} log_2(index+1)^{-1}, & \text{if } \textit{item} \text{ in predictions} \\ 0, & \text{if } \textit{item} \text{ not in predictions} \end{cases}$$

4.4.3 Pipeline

In this section we will briefly explain how were the experiments conducted.

In the first place, we processed the articles coming from the dataset explored in the previous section. This action is necessary to homogenize all the apostrophes and, also, fix any codification errors.

Afterwards, we downloaded the model, tokenizer and fill-mask pipeline from HuggingFace. They are an essential part of our system, as demonstrated in Chapter 3.

At this point, we built a custom dataset class using PyTorch, to optimize the performance time. This way, some operations can be parallelized; even more when taking advantage of the GPU. Fortunately, we do not need to own a compatible GPU since we can use one online. In our case, we profited from Kaggle offering some free usage of an environment with a built in GPU. The only thing we had to do was to upload our notebook and dataset and run it there.

The code execution was completed without any major setback. The notebook had all the above mentioned preparations, as well as the main method and some helper functions. The helper functions were used to clarify the process and give a richer structure to the notebook. Those support functions primarily contained the operations needed to predict and, then, to evaluate the results. The main function, on its part, was responsible for sequentially calling all the steps.

Once the code had ran, the final results were downloaded. Computing them was the hard part of the work. But, considering that we only keep the metrics, their final size is not too big (almost 2MB for 10000 predictions).

4.4.4 Results

The previously explained pipeline was ran with sentences that had between 13 and 33 words: corresponding to the mid-half of the distribution. In that region, we had a little more than one million sentences.

As you might expect, the experiments were performed with only a small sample of that dataset. Our election was to take 10k sentences and evaluate their results. Just to make sure our process was not faulty, we had two trials with different partitions.

Measure	NDCG score	confidence	hit@1	hit@3	hit@5
1st run	0.664	0.675	0.567	0.698	0.746
2nd run	0.666	0.676	0.570	0.700	0.748

Table 4.2: Model evaluation results

In the Table 4.2 the metrics for each one of the trials are portrayed. It is quite clear that the two results basically coincide with each other. Therefore, we might be able to deduce that the random samples we took were representative of the dataset.

After examining the obtained results, we can conclude that the model performed quite well. Interestingly, we might interpret the differences between the various hits as a generalization of what might be happening behind the scenes.

We can see that more than half of the times it got the prediction right at the first try (that can be understood as a one-hot measure). Also, we can observe that the hit@5 is less than 0.2 larger that the hit@1. From there, we can derive that if the result was not immediate, the model was not very likely to improve in the following attempts.

That deduction is backed up by the relatively high confidence that the model has on its first guess: around two thirds. So, although we cannot be completely assured, we can say that the attributed behaviour is quite likely to be happening.

4.5 Semantic Textual Similarity

For the second part of our experiments, we wanted to get the approximate nearest neighbors of a given article. That task will use the embeddings generated by each article and locate the closest ones. Taking a step back, we first needed to investigate how well could the model embeddings relate different sentences: a process which goes by the name of **Semantic Textual Similarity**, STS. For that matter, we used a labelled dataset and evaluated the predictions of our model using it as a reference.

4.5.1 Catalan Language Understanding Benchmark

We took advantage of the *Semantic Textual Similarity in Catalan* resource released as part of the Catalan Language Understanding Benchmark (CLUB), accompanying the BERTa model [2]. That dataset was published as an open resource. It includes over 3000 pairs of sentences written in Catalan and evaluated by 4 different annotators. Each of them gave the pairs a score ranging from 0 (no similarity at all) to 5 (semantic equivalence).

That methodology is a pretty standard practice in this type of problems. It is based on the work done in the SemEval challenges. The SemEval challenges come each year as a workshop in semantic evaluation. The paper [20] explains in detail what is being evaluated at SemEval, but we will not go into the technicalities in here.

What we did was to closely examine the dataset with the aim of getting an idea of the contents. In the Figure 4.5 we can see two examples of the dataset. Effectively, the more similar sentences get a better result that the ones that differ in their meaning.

```
Score: 4.67 - Id: TE2_21
No se sap res més de la seva vida.
De la seva vida, però, no se'n sap gaire res.

Score: 1.25 - Id: Oscar2_619
Digues un llibre que no hagis llegit.
No passa res si t'has llegit el llibre.
```

Figure 4.5: A couple of sentence pairs with their scores and ids

When studying the contents of the files, it was seen that they could have been slightly better organized. That is because all the outputs of the steps of the pipeline were present in the file, with the purpose of being able to analyse the process. However, some inconsistencies were found regarding the number of row in the different datasets. Although the analysis were looked over the smallest one was chosen, just to be on the safe side.

4.5.2 Model predictions

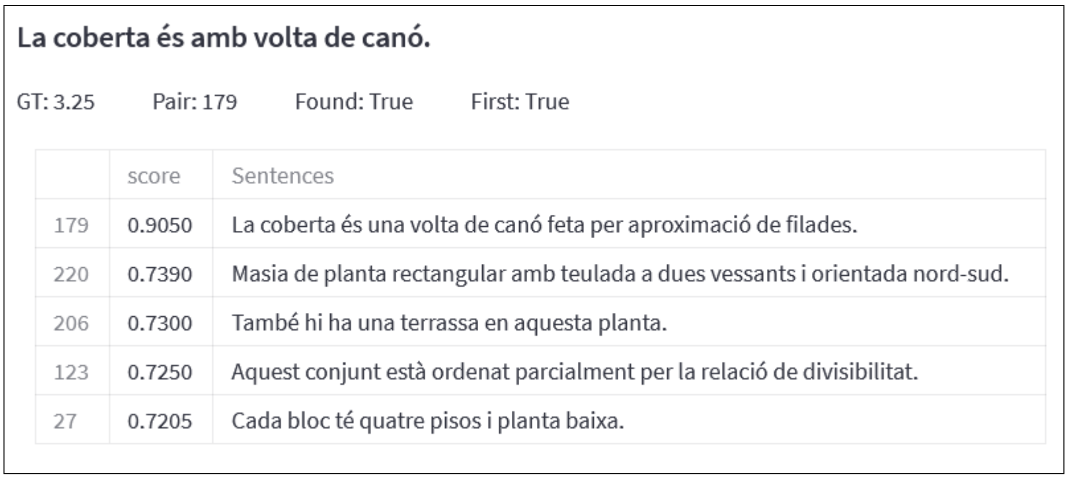
As stated in the previous section (4.5.1), the CLUB was analysed and used as a reference. The task now at hand was to determine the quality of the transformers model predictions. This time, we did not use Kaggle and we only processed a sample of 200 sentence pairs.

The steps taken were the following ones:

1. Use a transformers model to **encode** each sentence (individually) and obtain the corresponding embedding.
2. Obtain the **nearest** 5 sentences from the embeddings, using the cosine similarity.
3. Check if the matching pair was present and calculate the **metrics**.

With the aim of having a baseline with which being able to compare the Catalan only BERTa model, the steps were repeated with the **multi-qa-MiniLM-L6-cos-v1** model [4]. Both of them needed to be used in the sentence-transformers environment, as explained in the introductory part.

In the Figure 4.6 we can see an example of a sentence, its pair, the Ground Truth, and its predicted matches along their cosine similarity scores (using the BERTa model). That interface is result the development of a Streamlit app. More on that in the Section 4.6.2.


La coberta és amb volta de canó.

GT: 3.25 Pair: 179 Found: True First: True

	score	Sentences
179	0.9050	La coberta és una volta de canó feta per aproximació de filades.
220	0.7390	Masia de planta rectangular amb teulada a dues vessants i orientada nord-sud.
206	0.7300	També hi ha una terrassa en aquesta planta.
123	0.7250	Aquest conjunt està ordenat parcialment per la relació de divisibilitat.
27	0.7205	Cada bloc té quatre pisos i planta baixa.

Figure 4.6: Sentence number 379 and its predicted matches

[4] `https://huggingface.co/sentence-transformers/multi-qa-MiniLM-L6-cos-v1`

In the Table 4.3 we can see the results obtained by each one of the models. The metrics are the same as in the previous evaluation, with the only difference being the scores: they correspond to the cosine distance evaluation. The mean between all the sentence predictions was made.

Model	hit@1	hit@5	max score	other scores	diff
BERTa	0.948	0.988	0.912	0.800	0.112
multi-qa	0.870 (348)	0.943 (378)	0.742	0.553	0.189

Table 4.3: STS results

As you see, they are quite good results. The BERTa model correctly assigned 95% of the sentence pairs to the one with the highest value. It can also be seen that the scores calculated with the Catalan-only model are higher and, thus, have a closer relationship with the original sentences.

The difference between the predicted sentence score (column *max score*) and the next 4 scores (*other scores*) appears in the column *diff*. They are similar in both models, having a larger gap between the two *max* and the two *other* scores.

It is interesting to also study when does the model fail to make the correct predictions (miss). Since not all the sentences have exactly the same similarity, we could check if the model fails with a higher frequency in the pairs that have a more distinct meaning. Those metrics are reflected in the Table 4.4, where the Ground Truth is the value assigned by the (human) classifiers in the CLUB. For each case, it is averaged among the corresponding predictions.

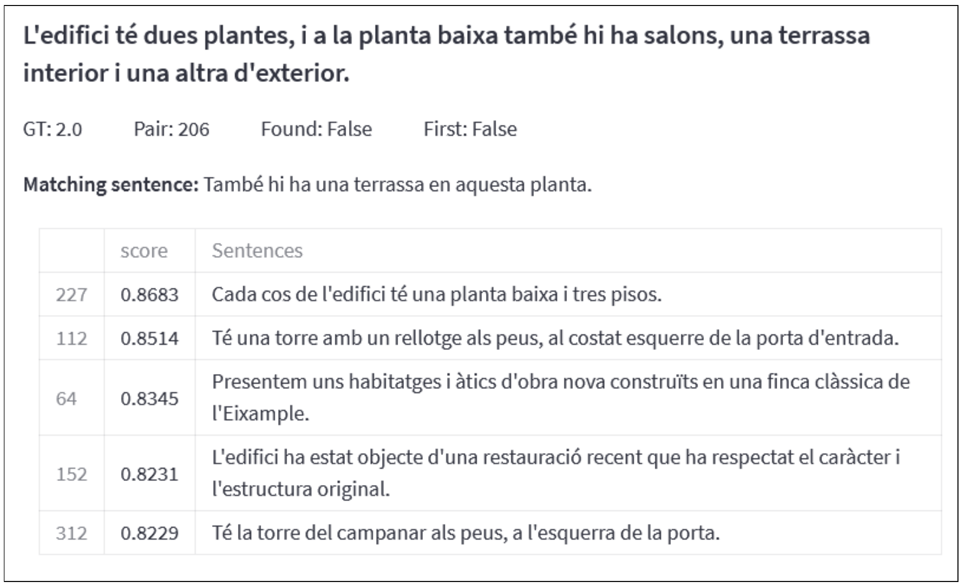
Model	hit@1	hit@5	miss@1	miss@5
BERTa	2.677	2.653	2.044	1.884
multi-qa	2.722	2.666	2.122	2.279

Table 4.4: STS results Ground Truth

A higher value on these predictions means the original sentences were considered more similar by the taggers. As expected, the earlier was a sentence retrieved, the higher the score; and similarly with the missing ones. Even though there is an exception: the *miss@5* of the multi-qa model was actually greater than the *miss@1* of the same model. This anomaly might be caused by the random properties of the subsample or may be an underperformance of the model. Either way, the issue could be further investigated in an effort to detect and understand the cause.

On the upside, this evidence reinforces the claim made by the authors of BERTa, that their model outperforms general purpose ones.

In most of the cases where the prediction did not match the original pair, the model's guess was actually more similar to the provided one than the original's pair. Also, the scores tend to be in quite a narrow range. An example of this phenomenon can be seen in the Figure 4.7.


L'edifici té dues plantes, i a la planta baixa també hi ha salons, una terrassa interior i una altra d'exterior.

GT: 2.0 Pair: 206 Found: False First: False

Matching sentence: També hi ha una terrassa en aquesta planta.

	score	Sentences
227	0.8683	Cada cos de l'edifici té una planta baixa i tres pisos.
112	0.8514	Té una torre amb un rellotge als peus, al costat esquerre de la porta d'entrada.
64	0.8345	Presentem uns habitatges i àtics d'obra nova construïts en una finca clàssica de l'Eixample.
152	0.8231	L'edifici ha estat objecte d'una restauració recent que ha respectat el caràcter i l'estructura original.
312	0.8229	Té la torre del campanar als peus, a l'esquerra de la porta.

Figure 4.7: Sentence 6 - An example where the pair is **not** found

4.6 GEC Similarities

After the work done in the previous sections, we can state that the Sentence Transformer along with the BERTa model form a good pair for working on the Semantic Textual Similarity problem. Now we can proceed to use that on the original dataset.

For that matter, we reproduced the pipeline explained when making the model predictions at 4.5.2. The output of the pipeline was the list of the sentences (now articles) embeddings.

Unfortunately, we are working with an unlabelled dataset and we do not have any quantitative metric for measuring how good are the embeddings relationships. However, since we are dealing the vector representation of the articles, we are able to create some visualizations from of the outputs.

4.6.1 Visualization techniques

If we recall the theoretical part, the output embeddings dimension is 768. It is impossible for us to conceive those magnitudes. Therefore, some procedures have been developed to project those vectors into a space we can more easily understand: 2D or 3D.

We will take a brief look at two of the most used ones: PCA and UMAP.

PCA

Principal Components Analysis (PCA) is a method used for dimensionality reduction via feature extraction. It is one of the simplest and most well known techniques. In order to calculate PCA the idea is to find the principal components to obtain lower-dimensional data while preserving as much of the data's variation as possible.

It does that by combining the input variables in a specific way, then keeping only the most valuable parts of all of the variables, while dropping the least important ones. This process has the added benefit of producing all the variables as linearly independent to each other.

In the Figure 4.8 we can see how data from a 3D space gets mapped to a 2D component space via PCA.

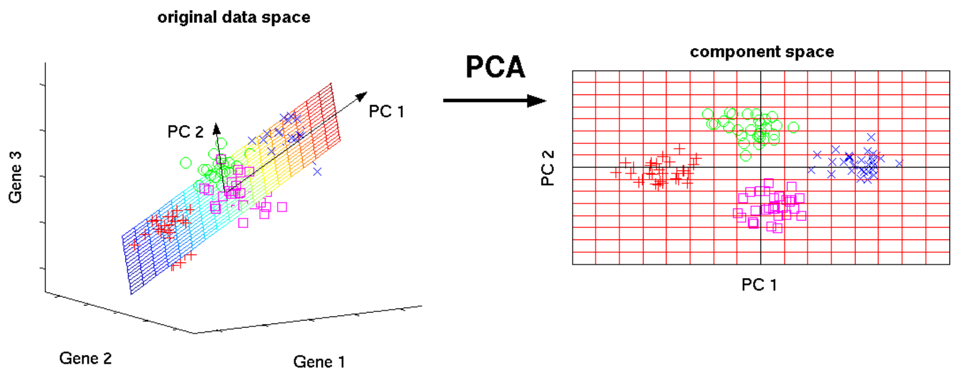


Figure 4.8: PCA Example [5]

UMAP

UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction [11]. Taking the input data, UMAP builds a high dimensional graph repre-

[5]Image obtained from `https://www.analyticsvidhya.com/blog/2016/03/pca-practical-guide-principal-component-analysis-python`

sentation. Then, it optimizes a low-dimensional graph to be as structurally similar as possible.

To build the initial, high-dimensional graph, it joins neighboring points together by creating a n-dimensional ball around each one and connects those whose radii overlap. It is able to overcome the challenge of the radius selection by doing it locally. The resulting graph is weighted, with the values representing the likelihood that two points are connected. Formally speaking, it is called a **fuzzy simplicial complex**.[4]

Dataset visualization

There are multiple implementations of those techniques from which we could benefit. We will use TensorFlow Projector [6], a tool specifically designed with that purpose: to visualize embeddings.

The process is really simple: we only need to correctly format the embeddings that we want to represent and upload them to the online visualizer. On top of the points, we can add some metadata to have a certain context to guide our intuitions.

TensorFlow Projector allows users to select the projection method between UMAP, T-SNE (UMAP's predecessor) and PCA. The latter falls short to capture any meaningful clustering information. T-SNE is more complex and allows for more hyperparameter tuning, including a temporal timeline; one of the reasons why we have opted to stay with the simpler of the representations: UMAP.

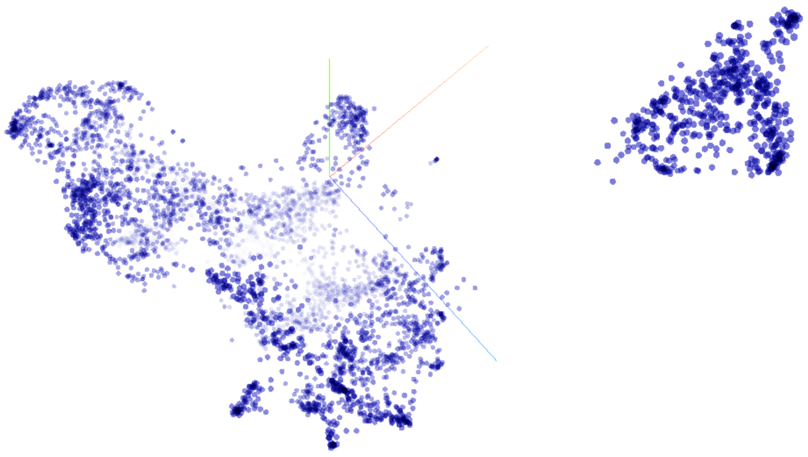


Figure 4.9: GEC Embeddings

[6]`https://projector.tensorflow.org`

The projections you will see have been produced using UMAP with the number of neighbors = 20.

You can see the construction obtained in the Figure 4.9. After the construction and exploration, we reach some interesting findings:

- The tiny dense island in the upper-middle is exclusively composed of political parties.
- The right cluster is mainly formed by people, with the exception of some sports clubs in the lower regions.

From this angle is hard to see, but by looking at the big cluster located on the left from the opposite side we see some more groupings. Those are shown in the Figure 4.10.

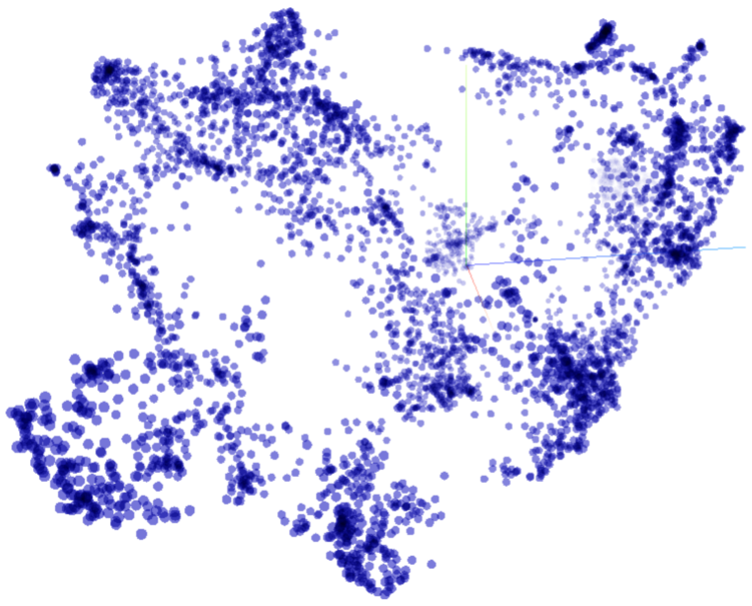


Figure 4.10: GEC Embeddings from the opposite side

Some more comments on the groups:

- The aggregation in the lower left corner corresponds to places from around the world.
- The dense cluster in the bottom of the image are sites belonging to Catalonia.
- The narrow column on the left and the upper section it connects to mainly contain articles related with science and medicine.
- The upper right cluster and the dense region below it are all articles about people.
- The middle sparse sector contains some news articles and other categories.

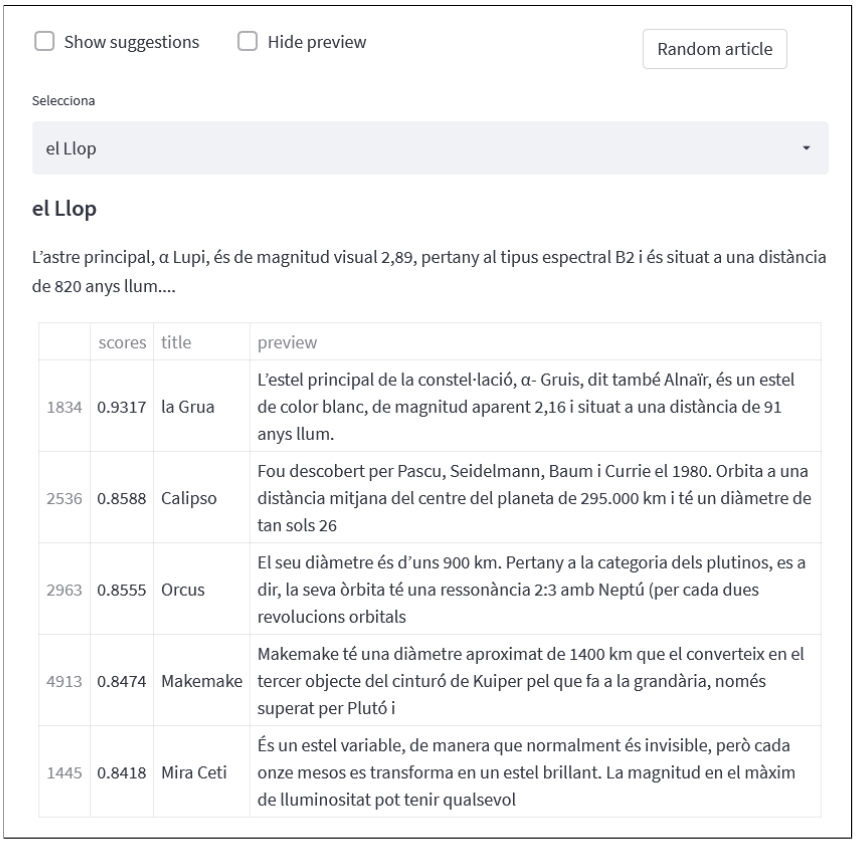
4.6.2 Front End

The final step we wanted to take was a way for the users to interact with the articles' database and show those that the model considered the most similar.

Our main goal was for the front-end to be simple and, if possible, quick to develop in Python. For those reasons, the Streamlit library [7] was used.

Streamlit is intended for the creation of simple interfaces, mainly with data in mind. The development is done via a script; in which you call the API components that will turn into the front-end components displayed in the screen. It automatically updates as you save the source file, what helps with the process.

Since the source file is written in Python, we can directly import data from an exported file with pandas or pickle. That data is easily visualized in the given widgets. Instantiating them is as simple as calling a function with the contents: saving the need to deal with the hassle associated with creating a back-end, a front-end and linking them.



el Llop

L'astre principal, α Lupi, és de magnitud visual 2,89, pertany al tipus espectral B2 i és situat a una distància de 820 anys llum....

	scores	title	preview
1834	0.9317	la Grua	L'estel principal de la constel·lació, α- Gruis, dit també Alnaïr, és un estel de color blanc, de magnitud aparent 2,16 i situat a una distància de 91 anys llum.
2536	0.8588	Calipso	Fou descobert per Pascu, Seidelmann, Baum i Currie el 1980. Orbita a una distància mitjana del centre del planeta de 295.000 km i té un diàmetre de tan sols 26
2963	0.8555	Orcus	El seu diàmetre és d'uns 900 km. Pertany a la categoria dels plutinos, es a dir, la seva òrbita té una ressonància 2:3 amb Neptú (per cada dues revolucions orbitals
4913	0.8474	Makemake	Makemake té una diàmetre aproximat de 1400 km que el converteix en el tercer objecte del cinturó de Kuiper pel que fa a la grandària, només superat per Plutó i
1445	0.8418	Mira Ceti	És un estel variable, de manera que normalment és invisible, però cada onze mesos es transforma en un estel brillant. La magnitud en el màxim de lluminositat pot tenir qualsevol

Figure 4.11: Streamlit app screenshot

[7] https://streamlit.io

As can be seen in the Figure 4.11, the first element of the web page is the article we have chosen (or the default one): made up of the title and a preview of the text. Then, the predominant component is the table showing the 5 nearest articles, their scores and a small preview too. We deemed the preview a necessary feature because it is difficult to guess the contents of some articles only by their title. That being said, if the user prefers not to see that preview there is the option to hide it.

Be noted that in the drop-down menu, the user can begin typing and the titles list will be filtered accordingly. That is useful if the user is looking for a particular article.

Thinking into the user experience when using this app, we considered essential some hints to help with exploring the results, which is the main objective of the app. Bearing that in mind, a button to random article is permanently shown. On top of that, the user can toggle some suggestions that will show them 4 article titles and the possibility to explore their nearest results when clicking on them. The resulting header can be seen in the Figure 4.12.

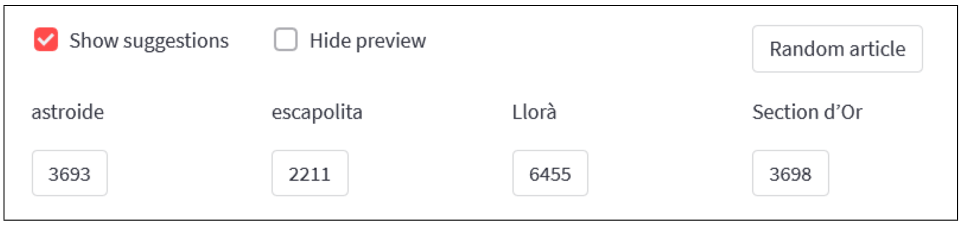


Figure 4.12: Streamlit app header

An added advantage when using Streamlit was the ease with which you could deploy the app costless. In a few minutes (and completely hassle-free) we had our page up and running. The initial load time could be slightly improved but you could not ask for a better service.

You can try out the application for yourself. It is deployed in the following url: `https://share.streamlit.io/t1emp0/tfg/main/GEC_streamlit_app.py`

4.6.3 Approximate Nearest Neighbours

The *k*-Nearest Neighbours (*k*-NN) is an optimization problem in which given a point from a set, the *k* closest points need to be found. This problem arises quite frequently in different fields of application.

In our particular case, we are only interested in finding the articles that are nearer to the given one. Therefore, we could use this method in order to save the need of calculating the cosine similarities between all the articles embeddings.

There are several algorithms that resolve this problem. The quality of the outputs depends on the time and space complexities of their search data structures.

A common relaxation of the problem is what is known as the Approximate Nearest Neighbours search. The aim of this thesis was not to implement these algorithm nor to compare them. For this reason, we took advantage of the work done by some libraries. After testing a few of them, Facebook's FAISS [8] was chosen due to its simplicity and speed. A proof of concept was made using some data from the previous sections. The pipeline started by indexing the dense vectors. Afterwards you could launch a query with another vector and it would return the closest ones. Despite no formal evaluation taking place, different trials were ran and the final results were satisfactory.

Using the proof of concept as a starting point, the ideal scenario would be to build a web service with that use case. Such web service would have a running instance of the FAISS pipeline. The first time it was executed, the indexing would need to take place but afterwards it will be as easy as making a query to obtain the results.

This would greatly expand the usability of the preview app by allowing users to query the database with their own texts. First, those texts will need to be passed through the sentence transformer to obtain their embeddings. Then, the embedding would be sent to the web service and the response would be shown to the user.

Conceptually it is a simple pipeline. However, due to the architectural constraints of this implementation, it ended up out of the scope of the thesis.

Chapter 5

Conclusions and future work

The first objective of this thesis was to understand how do neural networks work: in particular, the attention mechanism and the transformers architecture as well as the language models and their training. Our aspirations were to not only get the basic idea but also to be able to formulate those concepts from a mathematical perspective. In the beginning, it looked like a daunting task, but (with quite a bit of research) it was successfully achieved in Chapter 3.

Afterwards, in Chapter 4, we explain the pipeline of how a model should be tested by experimenting with some real-world data. Once we had checked the Semantic Textual Similarities from a labelled dataset and seeing that our model performed in a reasonable way, we proceeded to retrieve the k-Nearest Neighbors (k-NN) from a sample of the articles. With all the data obtained, some visualizations were produced which confirmed that the embeddings were behaving in the way we were expecting them to do so. In addition, a simple front-end application was built so that users could interact with the articles and get a grasp of the model's inner workings.

Bearing all the above in mind, we can claim that the thesis was fulfilled as intended and the objectives that were set in the start have been fruitfully accomplished. The development was quite smooth, which brought us great joy and left us with new knowledge that, for sure, will be proven useful in our future adventures.

However, we need to acknowledge some limitations of our study. Firstly, the results obtained from the GEC dataset articles lacked quantitative measures. Those metrics could have been obtained via user validation. That way, we could assess how good the predicted neighbours are and give some insights on their relationship. Nonetheless, it was quite clearly out of the scope of this project due to the magnitude difference.

Moreover, if we had more time (or resources) we could have aimed to process the whole dataset and not just a subset, meaning that the relationship between articles from the same collection might have been made more evident with more samples. Also, we could have been able to make some more tests even probably it would have only affected lightly to our results. We state that because, as seen in the Table 4.2, the data was already consistent and the returned *k*-NN were already the closest ones.

Another possible expansion of the thesis would have been to explore topic modelling: a collection of techniques that allow the users to group the articles by topic. It could be useful to see how articles are consumed along timeline segregated by themes.

Finally, it would have been positive to compare the BERTa model with some other Catalan model, as for example Julibert [8], another Roberta-based model. The pipeline is adapted for using Hugging Face's API, so it would be trivial to perform this experiment (once having the resources needed).

In conclusion, we can state that is very useful to have a language-specific model for Catalan language with all the associated benefits it carries.

[8] `https://huggingface.co/softcatala/julibert`

Bibliography

[1] Panos Achlioptas. *Stochastic Gradient Descent in Theory and Practice*. 2019. URL: https://scholar.google.com/citations?view_op=view_citation&citation_for_view=Z975RkYAAAAJ:IjCSPb-OGe4C.

[2] Jordi Armengol-Estapé et al. *Are Multilingual Models the Best Choice for Moderately Under-resourced Languages? A Comprehensive Assessment for Catalan*. 2021. DOI: 10.48550/ARXIV.2107.07903. URL: https://arxiv.org/abs/2107.07903.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. DOI: 10.48550/ARXIV.1409.0473. URL: https://arxiv.org/abs/1409.0473.

[4] Andy Coenen and Adam Pearce. *Understanding UMAP*. 2019. URL: https://pair-code.github.io/understanding-umap.

[5] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805.

[6] Zellig S. Harris. "Distributional Structure". In: *WORD* 10.2-3 (1954), pp. 146–162. DOI: 10.1080/00437956.1954.11659520. eprint: https://doi.org/10.1080/00437956.1954.11659520. URL: https://doi.org/10.1080/00437956.1954.11659520.

[7] Kalervo Järvelin and Jaana Kekäläinen. "Cumulated gain-based evaluation of IR techniques". In: *ACM Transactions on Information Systems (TOIS)* 20.4 (2002), pp. 422–446. URL: https://faculty.cc.gatech.edu/~zha/CS8803WST/dcg.pdf.

[8] Jeff Johnson, Matthijs Douze, and Hervé Jégou. *Billion-scale similarity search with GPUs*. 2017. DOI: 10.48550/ARXIV.1702.08734. URL: https://arxiv.org/abs/1702.08734.

[9] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. DOI: 10.48550/ARXIV.1907.11692. URL: https://arxiv.org/abs/1907.11692.

[10] Bryan McCann et al. *Learned in Translation: Contextualized Word Vectors*. 2017. DOI: 10.48550/ARXIV.1708.00107. URL: https://arxiv.org/abs/1708.00107.

[11] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2018. DOI: 10.48550/ARXIV.1802.03426. URL: https://arxiv.org/abs/1802.03426.

[12] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. DOI: 10.48550/ARXIV.1301.3781. URL: https://arxiv.org/abs/1301.3781.

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

[14] Matthew E. Peters et al. *Deep contextualized word representations*. 2018. DOI: 10.48550/ARXIV.1802.05365. URL: https://arxiv.org/abs/1802.05365.

[15] Alec Radford and Karthik Narasimhan. "Improving Language Understanding by Generative Pre-Training". In: 2018. URL: https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf.

[16] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2019. URL: http://arxiv.org/abs/1908.10084.

[17] Gerard Salton and Christopher Buckley. "Term-weighting approaches in automatic text retrieval". In: *Information Processing & Management* 24.5 (Jan. 1988), pp. 513–523. DOI: 10.1016/0306-4573(88)90021-0. URL: https://doi.org/10.1016/0306-4573(88)90021-0.

[18] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[19] Thomas Wolf et al. *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. 2019. DOI: 10.48550/ARXIV.1910.03771. URL: https://arxiv.org/abs/1910.03771.

[20] Oskar Wysocki, Malina Florea, and Andre Freitas. *What is SemEval evaluating? A Systematic Analysis of Evaluation Campaigns in NLP*. 2020. DOI: 10.48550/ARXIV.2005.14299. URL: https://arxiv.org/abs/2005.14299.