Gonçalo Mascarenhas

A Semantic Search System for the Legal Domain

UNIVERSIDADE DO ALGARVE

Faculdade de Ciências e Tecnologia

2023

Gonçalo Mascarenhas

A Semantic Search System for the Legal Domain

Master in Informatics Engineering

Trabalho efetuado sob a orientação de:
Prof. João Dias
Prof. Pedro Santos

UNIVERSIDADE DO ALGARVE

Faculdade de Ciências e Tecnologia

2023

A Semantic Search System for the Legal Domain

Declaração de autoria de trabalho

Declaro ser o autor deste trabalho, que é original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências incluída.

Gonçalo Mascarenhas

Acknowledgments

I would like to express my deepest gratitude to my supervisors, whose expertise, understanding, and patience, added considerably to my graduate experience. Their mentorship was paramount in providing a well-rounded experience consistent with my long-term career goals.

I would also like to thank my committee members for their insightful comments and suggestions, which have greatly enriched this work.

To my family, thank you for always encouraging my pursuits and for providing unconditional love and support throughout my life. Your belief in me has always been a source of strength.

Lastly, I would like to acknowledge the Competition on Legal Information Extraction / Entailment (COLIEE) for their invaluable datasets. It has made this research possible.

This accomplishment would not have been possible without you all. Thank you.

Abstract

This thesis focuses on the critical issue of accessing and comprehending legal documents within the context of digital gazettes. Traditional keyword-based search systems have often proven insufficient in providing precise and relevant results when applied to complex legal texts. In response to these limitations, our research presents an innovative information retrieval system that capitalizes on the power of semantic search techniques.

Our approach is designed to address the inherent gap between the formal language used in legal documents and the more informal and diverse nature of user queries. By doing so, our system strives to significantly enhance the accuracy and relevance of search results, ensuring that users can access the legal information they require more efficiently and effectively.

To assess the practical effectiveness of our proposed system, we have subjected it to rigorous evaluations using the Competition On Legal Information Extraction and Entailment (COLIEE) dataset. These evaluations have provided us with valuable insights into the system's capacity to accurately extract information from legal texts and establish meaningful relationships within them.

By introducing a specialized semantic search approach customized for legal texts, our aim is to provide users with the tools to navigate complex legal documents more effectively. This research has the potential to enhance legal literacy, transparency, and accessibility, benefiting not only legal professionals but also the wider audience utilizing digital gazettes.

Keywords

Natural Language Processing; Legal Information Extraction; Lexical Search; Semantic Search; Information Retrieval System; Contextual Word Embeddings; Neural Network; BM25; BERT.

Resumo

Esta tese foca-se na questão crítica do acesso e compreensão dos documentos jurídicos no contexto dos diários eletrónicos digitais, como é o caso do Diário da República Eletrónico (DRE). Os sistemas tradicionais de pesquisa baseados em palavras-chave são frequentemente insuficientes na oferta de resultados precisos e relevantes quando aplicados a textos jurídicos complexos. Em resposta a estas limitações, apresentamos neste projeto um sistema de pesquisa inovador que aproveita o poder das técnicas de pesquisa semântica.

A nossa abordagem procura diminuir a lacuna existente entre a linguagem formal usada em documentos jurídicos e a natureza mais informal e diversa das consultas dos utilizadores. Desta forma, o nosso sistema melhora significativamente a precisão e relevância dos resultados da pesquisa, garantindo que os utilizadores possam acessar as informações legais de que precisam de maneira mais eficiente e eficaz.

Para avaliar a eficácia prática do sistema proposto, submetemo-lo a avaliações rigorosas usando o conjunto de dados da Competição de Extração/Inferência de Informações Jurídicas (COLIEE). Estas avaliações proporcionaram-nos resultados notáveis sobre a capacidade do sistema de extrair informações com precisão a partir dos textos jurídicos e em estabelecer relacionamentos semânticos significativos entre eles.

Ao introduzir uma abordagem de pesquisa semântica especializada, adaptada para textos jurídicos, tentamos fornecer aos utilizadores as ferramentas necessárias para navegar de maneira mais eficaz os documentos jurídicos mais complexos. Este projeto tem o potencial de melhorar a literacia jurídica, a transparência e a acessibilidade, beneficiando não apenas os profissionais

de direito, mas também o público em geral que utiliza diários eletrónicos digitais.

Palavras Chave

Processamento de Linguagem Natural; Extração de Informações Jurídicas; Pesquisa Lexical; Pesquisa Semântica; Sistema de Recuperação de Informações; Embeddings Contextuais de Palavras; Rede Neuronal; BM25; BERT.

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
BERT	Bidirectional Encoder Representations from Transformers
BM25	Best Matching 25
BPTT	Back Propagation Through Time
COLIEE	Competition On Legal Information Extraction and Entailment
DRE	Diário da República Eletrónico
FNN	Feedforward Neural Network
IR	Information Retrieval
LeSSE	Legal Semantic Search Engine
LIR	Legal Information Retrieval
LLM	Large Language Model
LSTM	Long Short Term Memory
MNLI	Multi-Genre NLI
NL	Natural Language
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
PBT	Population Based Training
RNN	Recurrent Neural Network
SBERT	Sentence-Bidirectional Encoder Representations from Transformers
SNLI	Stanford Natural Language Inference
STS	Semantic Textual Similarity problem

TF-IDF Term Frequency-Inverse Document Frequency Algorithm

UKP Ubiquitous Knowledge Processing lab

USE Universal Sentence Encoder

1

Introduction

Contents

All laws and norms that make up a nation's legislation are part of a vast portfolio that is published by the official gazette of the nation. Usually, these documents are made available in electronic format by a digital gazette such as the Diário da República Eletrónico (DRE) for the portuguese legislation. This provides all citizens free and universal access to the legislation's articles, including laws, decree-laws, regulations, decisions by the Constitutional Court and other relevant texts. However, unfortunately it is not always easy to understand or find which legislative articles apply to a given situation.

Since all citizens have rights and duties they must fulfill, understanding laws and legislation becomes an important part of citizenship, and therefore it is inevitably required for the digital gazette's access to be easy, fast and precise. Nowadays, this access is typically possible through a search system that is based on literal search. This means that the legal articles presented to the user by the search engine are selected based on a comparison made between the literal keywords in their texts and the input provided by the user.

This type of search system may lead regular citizens to face difficulties when accessing digital legal documents, since most citizens are not accustomed to the formal language present in legislation's articles. Fortunately, important research has been conducted within the information retrieval, the artificial intelligence, and the legal communities.

An example of such community is the Competition On Legal Information Extraction and Entailment (COLIEE)[1]. This is an annual competition held since 2014, with the aim of establishing the state-of-the-art approaches for information retrieval and entailment using legal texts. It also provides access to the statute law competition data corpus, a dataset constituted by articles from the Japanese Civil Law that is well suited to train and validate the search system model proposed in this thesis.

1.1 Objectives

As mentioned, due to the level of language that is used in the legal domain, the process of finding relevant information for a given query might reveal to be quite challenging, especially for regular citizens. This may conduce the search system to show results that are not relevant to the user or it may not include all the articles considered to be relevant to the specific instance introduced in the query.

Any search system that relies only on exact matches between queries and texts suffers from this problem. More recent work has reported on significantly better results with semantic search. This is an alternative solution, where we look for a semantic connection between the legal text

[1] https://sites.ualberta.ca/~rabelo/COLIEE2021/

(formal language) and the user's query (informal language). This method involves looking up all the articles that share the same semantic meaning as the one extracted from the query. Based on this reasoning, our objective in this thesis is to developed a information retrieval system for the legal domain with semantic search abilities capable of presenting results that are relevant to a query introduced in Natural Language (NL).

1.2 Document Structure

This thesis is divided in six chapters. In the next chapter we summarize all the concepts needed for understanding the developed system. In Chapter Three we introduce state-of-the-art approaches to problems similar to ours. Chapter Four explains the properties of the statute law dataset from COLIEE 2021 and introduces the architecture of the developed solution. Chapter Five involves an analysis of the system's performance, examining various influencing factors and benchmarking it against a baseline information retrieval system. Lastly, Chapter Six presents the conclusion, offering a succinct recap of the research undertaken, the key accomplishments attained, and potential enhancements that could be pursued in the future to boost the system's performance.

2

Background

Contents

This chapter will explain the concepts necessary for understanding the state-of-the-art approaches detailed in the next chapter and the solution to our problem.

2.1 Information Retrieval

Information Retrieval (IR) is a fundamental task in many real-world applications, such as search engines, digital libraries, media search, spam filters and so on. There is an enormous amount and diversity of information that is available to us now, making this field a continuously evolving area of research.

More specifically, IR is the procedure through which a system retrieves information resources relevant to a query from a collection of available resources [4]. A user expresses its information need as a request and it is then formalised into a search statement for the retrieval system. The system will respond by matching information resources relevant to the given query.

2.1.1 Information Retrieval System

A IR system is a computer system that stores and manages information on documents, typically textual documents. Its objective is to assist users in finding the information they seek by informing on the existence and location of the desired documents, referred to as relevant documents. Hypothetically speaking, a perfect IR system would be able to fetch only the relevant documents and no irrelevant documents. But such systems do not exist, since different users may interpret the relevance of the same retrieved documents differently.

There are three basic functionalities that any IR system has to support [4]: the representation of both the documents and the query, and the comparison of the two representations.

The representation of the documents takes place offline, which means the user of the IR system won't be directly involved. This functionality is often referred to as indexing. During the indexing process documents may be stored in the system, but typically they are only stored partly, for example the title and the abstract, plus information about the location of the document. The representation of the user's information need is called query formulation process. The resulting representation is the query.

The comparison of the two representations is called the matching process [4]. The outcome of the matching process is a ranked list of documents. Usually the ranked retrieval puts the relevant documents on top of the ranked list, minimising the time the user needs to waste reading the documents. The theory behind ranking algorithms is a essential part of IR.

2.1.2 Legal Information Retrieval (LIR)

Legal Information Retrieval (LIR) is a subset of IR, which involves handling legal information such as case laws, legislation, legal opinion, and other legal academic works. Its relevance has increased in past years due to the vast and quickly increasing amount of legal documents available through electronic means [5].

Since the legal field is prone to jargon, polysemes and constant change, LIR requires different approaches to the way text is searched. The contrast between the formal text present in legal documents and the informal text written by regular citizens in the search queries creates an imbalance that may lead to a decrease on the effectiveness of legal searches. Some typical approaches are: boolean search, manual classification, and Natural Language Processing (NLP) of legal text.

In a boolean search system both the user request and the documents are regarded as a set of terms, which are combined with logic symbols like AND, OR, NOT. By applying these operators, along with the set of terms, it is possible to create a huge range of search operations. The system will then return documents containing the terms in the request, which often results in the retrieval of a large amount of documents. Boolean search is widely implemented in IR systems but does not overcome the problems discussed above.

Manual classification is a method that overcomes most of the problems inherent in LIR systems. It attempts to link texts on the basis to their type, value, and topic area. The only drawback behind manual classification is that it would need to be done by legal professionals and this process would take time, and subsequently, the increasing amount of legal documents makes this technique unsustainable [6].

2.1.3 NLP applied to LIR

Another way to look at this problem, that would exclude the need of manual text classification, would be to search for a connection between legal text (formal language) and the user's query (informal language). This is referred to as semantic search. This type of search generally employs NLP techniques to return results that match a user's query, even when there are no words in common.

NLP is a subfield of Artificial Intelligence (AI) that studies fundamental technologies for the meaning expressions of words, phrases, sentences, and documents, and for syntactic and semantic processing. It aims to equip computers with the tools to grasp and make sense of language, facilitating operations like summarizing content and translating languages. Whether the language is spoken or written, NLP takes real-world input, processes it, and makes sense

of it in a way a computer can understand. This has made the legal sector interested in the development of specific methodologies based on NLP for the management, storage, indexing and retrieval of legal documents [5].

2.2 Semantic search

Semantic search is a task that involves finding the sentences that are similar in meaning to a target sentence. For example, a semantic search model would return the top sentence pairs from a paragraph that are closest in meaning to each other.

Comparing the similarity between two natural language texts can be quite challenging, due to the nuances of natural languages where sentences can a have a similar semantic meaning, even though they have no words in common. This challenging comparison is referred to as the Semantic Textual Similarity problem (STS) problem. Advancements in NLP have opened up many techniques to address this problem. The ones that achieve better results tend to first convert each text into a vector, a technique called Embedding, and then compare the texts by computing the distance between the vectors that represent them.

2.2.1 Word and Sentence Embeddings

Embedding is a representation of natural language text as real-valued vectors, where texts that have similar meaning have a similar representation. This technique is extremely relevant to the STS problem, since embeddings are able to capture not only the syntax value but also the semantic meaning of texts. Word embeddings can be seen as the mapping of words in natural language to points in a multi-dimensional space. The similarity between the words can be obtain by calculating the distance between the points using a text similarity metric. The greater the proximity of the points, the higher the similarity between the words they represent. Figure 2.1 illustrates multiple examples of word embeddings in a multi-dimensional space.

Advancements in word embedding techniques have led to the discovery of improved methods for representing increasingly comprehensive information about words, extending these techniques to encompass not just individual words but entire sentences. There are multiple ways to create a sentence embedding. A possible and straightforward way is to average the word embeddings of all the words in a sentence in a weighted form, according to their frequency. This method is simple and fast, but it does not take into consideration the interaction between the words in a sentence, neither the word order. A better and more sophisticated method

[1]Image from https://corpling.hypotheses.org/495

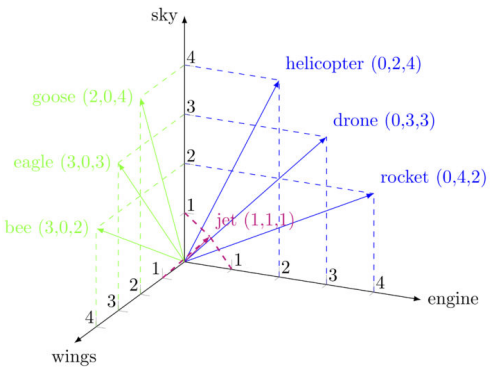

Figure 2.1: Word Embedding Examples.[1]

that addresses these problems is to use Bidirectional Encoder Representations from Transformers (BERT), a state-of-the-art model to create sentence embeddings. This model is presented later in this chapter.

2.2.2 Semantic Similarity Metric

A commonly used metric to measure the similarity between embeddings is the cosine similarity. Since a embedding is represented in vector form, we can use this metric to measure the similarity between two embeddings using the cosine of the angle between the two vectors. The closer the similarity score is to 1, the greater match between the two embeddings, if it is -1 than they are the opposite and if it is 0, than this indicates the two embeddings are completely independent.

The cosine similarity between two vectors A and B can be calculated using the following formula

$$CosineSimilarity = \cos\theta = \frac{A.B}{||A|| \cdot ||B||} = \frac{\sum_{i=1}^{n} A_i \cdot B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

2.3 Neural Networks applied to NLP

Neural networks have achieved state-of-the-art results in many NLP tasks, which led them to become a fundamental tool widely used in NLP. In this section, several neural network models will be explained, starting with the most simple ones and ending with the newer and more sophisticated that are build on top of the older ones.

2.3.1 Recurrent Neural Network

The order of the words in natural language texts is a very important factor in its semantic value, and thus an approach able to manage sequential data is required. Recurrent Neural Network (RNN) is a type of neural network that excels at processing this kind of data, because, unlike traditional neural networks where all the inputs and outputs are independent of each other, the output of RNNs depend on the prior elements within the sequence. This type of network can also be seen as multiple Feedforward Neural Network (FNN) that feed information from one network to the other.

The most important feature of RNNs is the hidden state vector, that allows them to remember some information about a sequence, by passing information from the previous state to the next. The hidden state vector, a_t, depends on the previous vector, a_{t-1}, and the current input x_t. It is defined as

$$a_t = g_1(W_{aa}a_{t-1} + W_{ax}x_t + b_a)$$

where $g1$ is an activation function that typically is either a hyperbolic tangent function (tanh), a sigmoid function (σ) or a rectified linear unit (ReLU). W_{aa} and W_{ax} are the weight matrices for the hidden state vectors and for the inputs, respectively, and the added b is bias.

The output vector (prediction), y_t, is expressed as follows

$$y_t = g_2(W_{ya}a_t + b_y)$$

where g_2 is another activation function, usually $softmax$. In figure 2.2 we have an illustration of the RNN architecture.

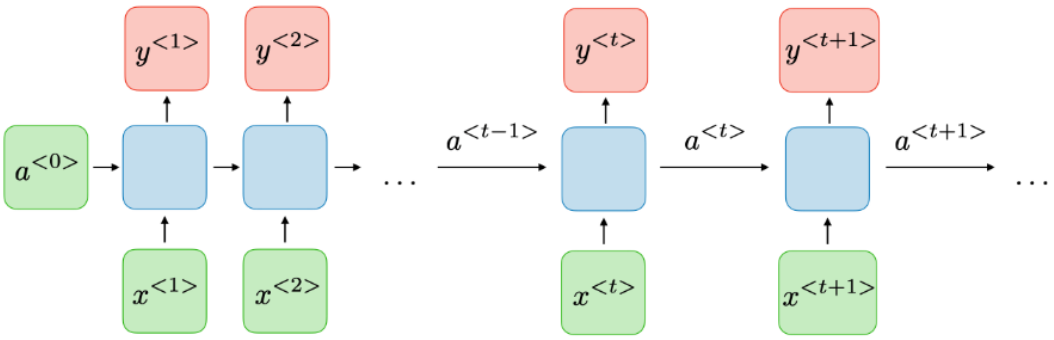

Figure 2.2: RNN architecture.[2] The output vector 'y' at timestep 't' is determined through the utilization of the hidden state vector 'a_t' in conjunction with the input 'x_t'.

[2]From https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

The weight matrices are initialised with random elements, and then, during the training of the RNN, they are adjusted using the error from the loss function - the error between the predicted and the actual observations. This adjusting is done using a back-propagation algorithm that updates the weights according to the loss function.

Applying back-propagation in RNNs is called Back Propagation Through Time (BPTT). Throughout the training of the RNN, this algorithm will calculate and accumulate the errors of each time-step, and a gradient of the loss function is calculated for each weight matrix. This gradient of each matrix is then used to update the matrix, according to a learning rate. Whenever the error is below a fixed threshold, the process ends.

RNNs face a major problem when dealing with sequences that have long-term dependencies. This problem is called Vanishing/Exploding Gradient Problem. It happens because of the multiplicative gradient, which will make the back-propagated errors diminish throughout the states becoming very small and tending to zero, or they will increase exponentially becoming to large. This means that the network experiences difficulty in memorising words from far away in the sequence and makes predictions based on only the most recent ones. One method that may help solving the exploding gradient problem, called gradient clipping, is to put a threshold on the gradients being passed back in time.

Since we want to extract the semantic value of a sentence, or even of whole paragraphs, these long-term dependencies must be learnt, and consequently, RNNs are not a viable choice for our problem. A better option capable of solving this issue is a variant of RNNs, the Long Short Term Memory (LSTM) network.

2.3.2 Long Short-Term Memory

LSTM [7] is a special variant of the standard RNN, specially designed to learn long-term dependencies. In addition to the hidden state vector present in RNN cells, the LSTM cells have a cell state vector. This vector is the key to LSTMs, since it runs straight through the entire chain with only a few minor linear changes, allowing information to flow along it practically unchanged. This means that the errors can back-propagate without being affect by the Vanishing/Exploding Gradient Problem.

In figure 2.3 we have an overview of the LSTM architecture. Similar to standard RNNs, LSTMs have a chain like architecture. The main difference is found in the cell structure. LSTM cells have three vector gates that decide what information will be updated and what will be forgotten from the cell state. These three vectors are: the forget gate, f_t, the input gate, i_t, and the output gate, o_t. They are composed of a sigmoid neural net layer and a pointwise operation.

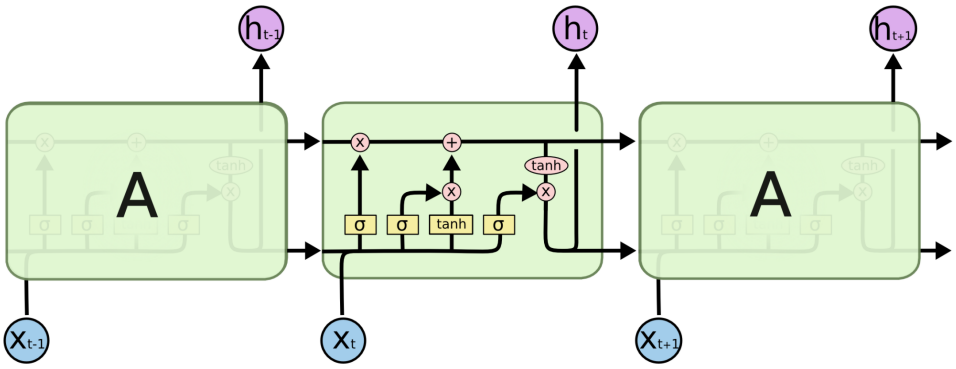

Figure 2.3: LSTM architecture. In this diagram, each line carries an entire vector, from the output of one cell to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.[3]

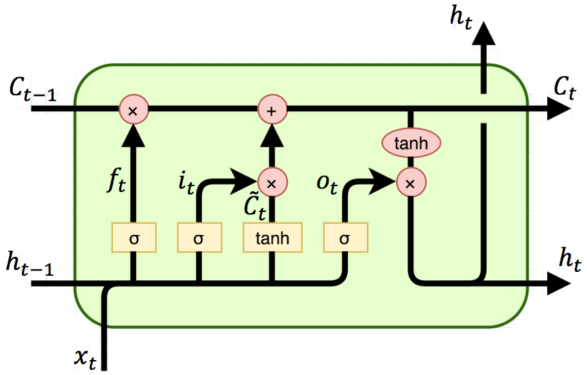

Figure 2.4: LSTM cell architecture.[3]

In figure 2.4 we have the structure of a single cell. The forget layer decides what information from the previous cell state, C_{t-1}, should stay in the current one, C_t. It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 that represents how much information will be preserved. the vector is given by

$$f_t = \sigma(W_f \cdot |h_{t-1}, x_t| + b_f)$$

The input layer decides what values will be updated. To decide what this values will be, the input layer is combined with a tanh layer, responsible for the creation of a new vector, $\tilde{C}_t$, containing

[3]From http://colah.github.io/posts/2015-08-Understanding-LSTMs/

the candidate values. The vectors are calculated as follows

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

To update the old cell state, C_{t-1}, into the new one, C_t, we do as follows

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

The output layer will filter the cell state vector before it is outputted. It is combined with a tanh layer to calculate the hidden state vector, h_t, that will be outputted and transmitted to the next cell.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \cdot \tanh(C_t)$$

2.3.3 Transformer

Even though LSTMs are able to learn long-term dependencies, they face another challenge when dealing with long input sequences (such as long text documents). During the BPTT process, due to the recurrent nature of this type of neural network, the gradients have to travel all the way from the end to the start of the network, which leads to long training times, even longer than standard RNNs because of the added complexity.

Modern computer processors are able to perform parallel computations, which would speed up the training process of neural networks significantly. However LSTMs and RNNs can not make use of this feature, because of their recurrence constraint. To overcome this limitation, a group of researchers at Google invented the Transformer [1], a new neural network model capable of training on sequential data using parallel computation.

The Transformer follows an encoder-decoder architecture (figure 2.5) and it has demonstrated remarkable success in various NLP applications. Consider the translation of the English sentence 'The cat is on the mat' to French. The process begins with the encoder, responsible for encoding the input sequence. Each word in the English sentence is embedded into a high-dimensional vector, capturing its semantic meaning and weighing the importance of each word in the context of the entire sentence.

As the encoder processes 'The cat is on the mat,' it produces a context vector that encapsulates

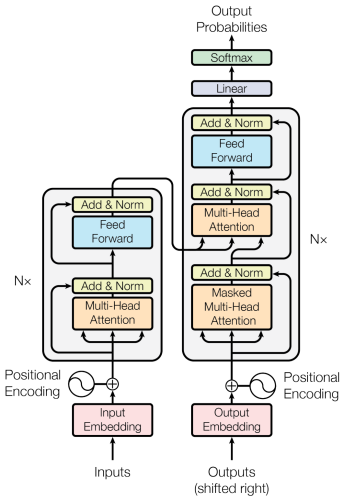

Figure 2.5: Transformer architecture [1]. Consider the translation of the English sentence 'The cat is on the mat' to French. The Transformer's encoder (left block) processes the English sentence 'The cat is on the mat' by embedding each word and capturing contextual relationships. The resulting context vector is then used by the decoder (right block) to generate the French translation step by step.

the essence of the input sentence, considering relationships between words and their contextual significance. The decoder then takes this context vector and generates the corresponding French translation step by step.

During the decoding phase, the model has the ability to focus on different parts of the context vector as it generates each word in the French translation. For example, when producing the first French word, the attention mechanism may emphasize 'The' and 'cat' in the context vector. As the decoder progresses, it attends to different parts of the context vector, effectively translating the entire sentence. This dynamic attentional process enables the Transformer to handle diverse sentence structures and capture the nuances of language, resulting in accurate and contextually appropriate translations.

We will now analyse how each component works. The first component is the Input Embedding, where the sequences of words received as input are converted into scalar vectors. To these vectors is then added the Positional Encoding, a vector that provides context depending on the position of a certain word in a sequence. The process behind the Output Embedding is similar to this one.

After the input goes through the Input Embedding and the Positional Embedding, the resulting vector is passed to the Encoder. This block contains a Multi-Head Attention layer and a Feed-Forward layer (figure 2.6).

The first layer in the Encoder block is the Multi-Head Attention layer. This layer will calculate

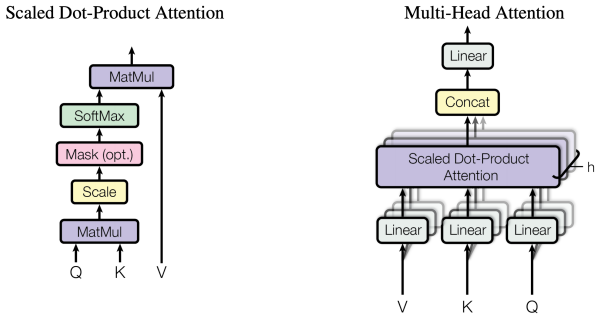

Figure 2.6: Scaled Dot-Product Attention (on the left) and Multi-Head Attention (on the right) [1]

an attention vector that contains weights representing the relevance given to a particular word within a sentence. Since each word weights its own value much higher compared with rest of the words in the sentence, multiple attention vectors need to be generated per word. This is why it is called Multi-Head Attention. Then, a weighted average is performed to compute the final vector of every word, capturing the contextual relationship between words in that sentence.

The attention calculation is expressed as

$$Attention(Q, K, V) = \text{softmax}(\frac{Q \cdot K^T}{\sqrt{d_k}}) \cdot V$$

where Q is the vector representation of the query, K contains the keys of the words in the query, V has the values of the words, and d_k is the dimension of the query and values.

Since there are multiple attention heads calculating the attention scores, these score need to be concatenated as follows

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h) \cdot W^O$$

where W^O is a weight matrix, h is the number of attention vectors and

$$head_i = Attention(Q \cdot W_i^Q, k \cdot W_i^K, V \cdot W_i^V)$$

W^Q, W^K and W^V are also weight matrices. The computation of each attention vector can be performed in parallel, which allows for fast processing.

Once the attention vectors are calculated, they are sent to the next layer, the Feed-Forward Network layer. This layer will essentially transform the attention vectors into a form that is acceptable by the next Encoder/Decoder block. Since the attention vectors are independent of

each other, this can be done in parallel.

The second block is the Decoder, composed by a Masked Multi-head Attention layer, a Multi-Head Attention layer and a Feed-Forward Network layer. The last two layers are similar to the ones present in the Encoder. The Masked layer has the particularity of masking positions that are ahead of the prediction before the softmax step in the self-attention calculation. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i [1].

The resulting masked attention vector then goes into another Multi-Head Attention Layer, similar to the one in the Encoder, and is used as the vector of values, whereas the query and keys are provided by the Encoder.

2.3.4 Bidirectional Encoder Representations from Transformers

BERT [2] is a transformer-based model developed by Google that achieved state-of-the-art performance across various NLP tasks, including Question Answering, Natural Language Inference, and others. Unlike other language models, BERT employs bidirectional training, which differs from the conventional unidirectional modeling where models consider text sequences either from left to right or vice versa. This approach allows BERT to attain a more comprehensive understanding of language, mirroring the way humans need to examine the context surrounding a word for full comprehension.

BERT is able to address diverse downstream tasks using special input representations designed to distinctly represent both an individual sentence and a pair of sentences in one token sequence. The input undergoes an initial representation via token embeddings, utilizing WordPiece [8] embeddings that break down words into smaller, trainable subword units. Additionally, segment embeddings are added to distinguish between sentences in pairs, enabling BERT to understand relationships and contextual dependencies. Finally, positional embeddings are added to capture the sequential order of tokens within the sequence, compensating for the lack of inherent word order information in transformer models, since they process input data in parallel. The complete input representation process is illustrated in figure 2.7.

The key feature of the BERT model is its utilization of bidirectional training. BERT considers both directions simultaneously, allowing it to capture a richer understanding of the context and relationships between words in a given text, enabling it to generate more accurate and contextually aware representations of language. However the bidirectional training has one downside. In standard unidirectional training, it is the existence of direction that grants an unbiased prediction of each word. But when this is removed the prediction of words becomes

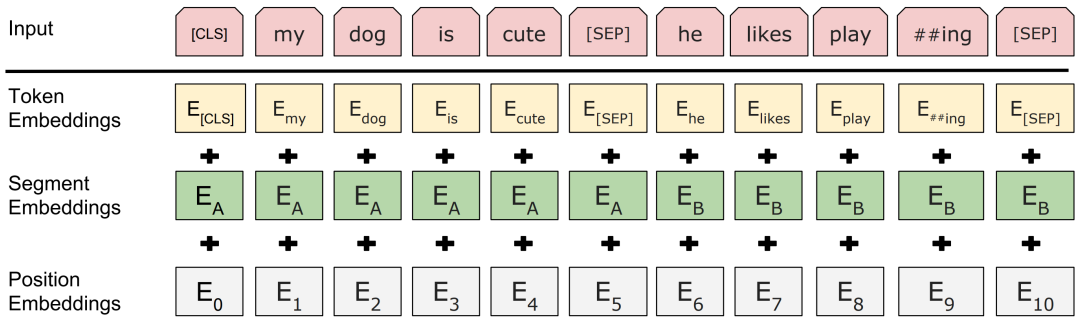

Figure 2.7: The three components of word embeddings - Token, Segment and Position [2]. The [CLS] token is a special token placed in the start of every sequence. The [SEP] token is another special token used to separate sentences.

biased, since it is based on both its left and right, which allows words to predict themselves.

To tackle this problem and reduce the bias, BERT uses a masking technique, named Masked Language Model (MLM). In short, they mask a small percentage of the input at random (replacing it with a [MASK] token), and then predict those masked tokens (words) based on the surrounding unmasked tokens. The masking is applied to 15% of the input, since this percentage achieved the better performance.

Although this approach solves the previous problem, it leads to a mismatch between the pre-training and the fine-tuning, because during the pre-training the predictions are only being made on the masked tokens, but in the fine-tuning phase no masking takes place. To solve this issue the masked tokens are not always replaced by the actual [MASK] token. Only 80% of the masked tokens are replaced with the [MASK] token. 10% are replaced with a random token and the remaining 10% keep the original token.

Some NLP tasks, such as Question Answering, require the model to understand relations between sentences. In order to learn these relations, during the pre-training BERT receives multiple sentences as input, and then attempts to evaluate if the next sentence is a subsequent sentence of the previous one. This training task is called Next Sentence Prediction (NSP). More specifically, when choosing the sentences A and B from each training example, 50% of the time B is the next sentence that follows A, and in the other 50% B is a random sentence.

2.4 Hyper-parameter Tuning Techniques

Hyper-parameter tuning is a very important step in the machine learning process, because it significantly affects the performance of machine learning models. Hyper-parameters are

parameters whose values are set before the learning process begins, as opposed to during the learning process. Examples include learning rate, batch size, and number of epochs.

The three main reasons that make the optimization of these parameters so important are:

- **Model Performance:** The right hyper-parameters can drastically improve the performance of the model, while the wrong ones can make the model perform poorly.
- **Preventing Overfitting or Underfitting:** Proper tuning can help balance variance and bias in a model, thus helping to prevent overfitting and underfitting.
- **Efficiency:** Well-tuned hyper-parameters can make the training process much more efficient, potentially saving a lot of computational resources and time.

The challenge is that there's no one-size-fits-all set of hyper-parameters for all machine learning problems, and the optimal configuration often depends on the specific dataset and model being used. Therefore, hyper-parameter optimization typically involves running many training jobs with different hyper-parameters to find the combination that leads to the best model performance.

Typically, several strategies can be employed for hyper-parameter optimization. The three methods that are frequently utilized include Grid Search, Random Search and Bayesian Optimization. Additionally, a more advanced and recent method that is gaining popularity is Population Based Training (PBT).

In the case of **Random Search**, a variety of neural networks are trained simultaneously yet independently, and the model with the best performance is chosen after training concludes. This typically implies that only a small portion of the population will be trained using effective hyperparameters. Conversely, a larger segment will be trained using less optimal hyperparameters, resulting in a significant expenditure of computational resources that ultimately doesn't contribute to the final, optimal model. (Figure 2.8)

Grid Search is one of the most basic and straightforward methods of hyper-parameter tuning. As the name suggests, grid search involves defining a grid of hyper-parameters and systematically working through every combination. For instance, if we have two hyper-parameters, and three possible values for each, grid search will train and evaluate a model for all nine combinations. While grid search can be computationally expensive and time-consuming, it is exhaustive and guaranteed to find the best combination within the specified grid.

Bayesian Optimization is a more sophisticated method of hyper-parameter tuning that constructs a probability model of the objective function and uses it to select the most promising hyper-parameters to evaluate in the actual objective function. The key idea is to spend more time

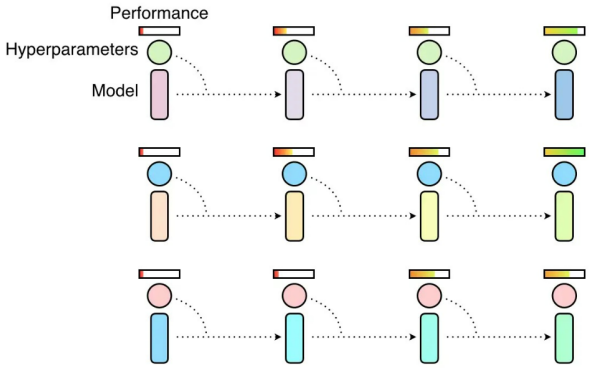

Figure 2.8: Random Search of hyperparameters, where numerous hyperparameter configurations are tested simultaneously, but independently. While certain hyperparameter settings may result in models exhibiting strong performance, others may not yield favorable outcomes.[4]

in picking the next set of hyper-parameters to evaluate, in the hopes of finding better solutions with fewer evaluations.

Population Based Training (PBT) [9] is a method for training neural networks that efficiently identifies the optimal set of hyper-parameters and model for a task. Like Random Search, PBT trains a population of networks simultaneously, but with a very significant difference, it uses information from the entire population to refine hyper-parameters and allocate resources to promising models. (Figure 2.9)

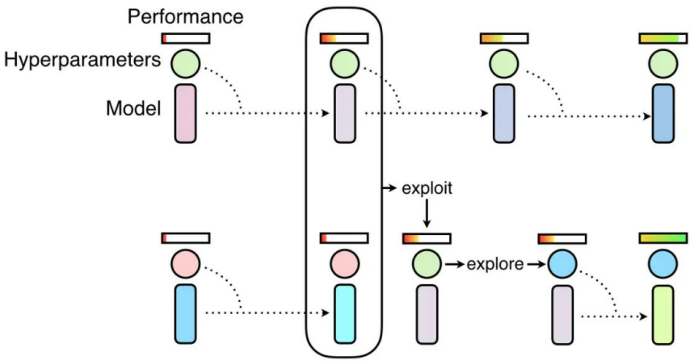

Figure 2.9: Population Based Training of neural networks starts like random search, but allows models to exploit the partial results of other models and explore new hyperparameters as training progresses.[4]

This technique is a hybrid of random search and hand-tuning methods, drawing inspiration from genetic algorithms. It starts by training multiple neural networks in parallel with random hyper-parameters but adjusts them based on the performance of the other models in

[4]From https://www.deepmind.com/blog/population-based-training-of-neural-networks

the population. A model might adopt parameters from better-performing ones or explore new hyper-parameters by randomly altering current values.

This technique allows PBT to quickly find good hyper-parameters, dedicate more training time to promising models, and adapt hyper-parameters during training, leading to automatic discovery of the best configurations.

Considering these benefits, we decided to utilize the PBT algorithm available in the Ray Tune library for the optimization of hyper-parameters.

3

Related Work

Contents

In this chapter, we provide a description of important concepts that we applied in our system, specifically the Best Matching 25 (BM25) algorithm and the Sentence-BERT model. We will also explore two related systems that present innovative strategies that we adapted into our own model.

3.1 Lexical approaches applied to Information Retrieval

3.1.1 Term Frequency-Inverse Document Frequency Algorithm

Term Frequency-Inverse Document Frequency Algorithm (TF-IDF) is a statistical measure often used in information retrieval. It evaluates how important a word is to a document in a collection of documents. The importance increases proportionally to the number of times a word appears in the document and is offset by the number of documents that contain the word. This way, stop-words such as 'a', 'the', 'that', 'do'... rank low despite appearing multiple times, since they are not relevant to any document in particular.

It is composed by two parts:

- TF (Term Frequency) – represents how frequently a term appears in a document.

- IDF (Inverse Document Frequency) – evaluates how common or rare a word is in the document collection. Values near 0 show that terms are very common and values near 1 show that terms are rare. This is the part of the algorithm that weighs down stop-words and scales up the rare ones.

In mathematical terms, the relevance of a term t to a document d from the document set D is calculated as follows

$$TF\text{-}IDF(t, d, D) = TF(t, d) \cdot IDF(t, D)$$

where

$$TF(t, d) = \log(1 + freq(t, d))$$
$$IDF(t, D) = \log \frac{\#D}{count(d \in D : t \in d)}$$

3.1.2 Best Matching Algorithm

The Opaki BM25 [10] is widely used as an IR ranking algorithm. It is considered a state-of-the-art approach in IR that relies on TF-IDF, and is used by search engines to estimate the relevance of documents to a given query. Given a query Q, containing keywords $q_1, q_2, ..., q_n$, the BM25 score of a document D is defined as

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})}$$

In this formula, $f(q_i, D)$ stands for the term frequency of q_i in the document D, avgdl is the average document length in the set of documents and $|D|$ is the length (amount of words) of document D. k_1 and b are the optimization parameters. The IDF term is usually computed as follows

$$IDF(q_i) = ln(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1)$$

This IDF formula is a variation of the one presented in the previous section. The N stands for the amount of documents in the collection and $n(q_i)$ is the number of documents containing q_i.

3.2 Sentence-BERT

Many of the most widely used transformers for NLP tasks, such as question answering and language modeling are word-level transformers. However, for a task like semantic search, which requires a strong sentence-level understanding, using just word-level transformers becomes computationally unfeasible. The first attempts to use BERT to create sentence embeddings were made by averaging all the word-level embeddings. However, this method resulted in worse results than other embedding techniques.

In 2019, a group of researches from the Ubiquitous Knowledge Processing lab (UKP) developed a modification of the BERT model using siamese BERT networks named Sentence-Bidirectional Encoder Representations from Transformers (SBERT) [3]. This model not only extracts semantically relevant sentence embeddings but also achieves significantly faster processing speed compared to BERT. In the paper [3] the authors where able to reduce the effort for finding the most similar pair from 65 hours with BERT to about 5 seconds with SBERT, while maintaining the accuracy from BERT.

SBERT uses a siamese architecture that is composed by two BERT networks, that are identical and share the same parameters. During the training, the model receives multiple pairs

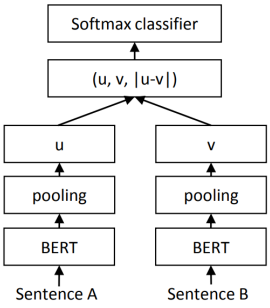

Figure 3.1: SBERT with classification function [3]

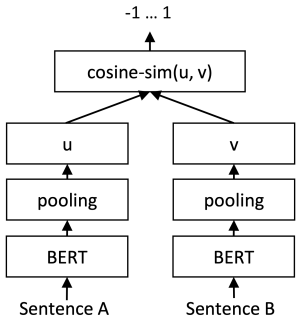

Figure 3.2: SBERT with regression function [3]

of sentences, where each BERT network is fed a sentence to encode. These pairs include a premise and a hypothesis. Each pair is assigned one of three labels: 0 if the premise suggests the hypothesis (entailment), 1 if they are not related, and 2 if they contradict each other. So, sentence A (premise) is fed to the BERT network A and sentence B (hypothesis) goes to network B. The encodings will then go through a pooling operation to joint all the token embeddings outputted by the BERT networks into one vector. In the original paper [3] they found that mean-pooling had better results than other methods like max and [CLS] vector strategies. With the resulting vectors, the authors propose three approaches for optimising different training objectives: using a classification objective function (figure 3.1), using a regression objective function (figure 3.2) and using a triplet objective function.

The model was trained using a softmax-loss approach on the Stanford Natural Language Inference (SNLI) [11] and Multi-Genre NLI (MNLI) corpora [12]. SNLI contains 570K sentence pairs, and MNLI contains 430K.

To evaluate how the Sentence-BERT model performs on common STS tasks, the authors compared different metrics to assess the similarity between the embeddings. They chose to use the cosine similarity, but also tried to use negative Manhattan and Euclidean distances. The results were roughly the same, so they picked the cosine similarity, since it is the most used metric. The evaluation was performed with both unsupervised and supervised STS tasks.

Regarding the unsupervised approach, the authors used the STS tasks 2012-2016[1], the STS benchmark [13] and the SICK-Relatedness [14] datasets. Each of the datasets contain labels for sentence pairs from 0 to 5 that represent how similar a pair is. SBERT was only outperformed by the Universal Sentence Encoder (USE) in the SICK-R dataset. This is justified by the fact that USE was trained on various datasets, including news, question-answer pages and discussion forums, while SBERT only had the knowledge gained with the pre-training from BERT and NLI data. The results are presented in figure 3.3.

[1]http://alt.qcri.org/semeval2020/

Model	STS12	STS13	STS14	STS15	STS16	STSb	SICK-R	Avg.
Avg. GloVe embeddings	55.14	70.66	59.73	68.25	63.66	58.02	53.76	61.32
Avg. BERT embeddings	38.78	57.98	57.98	63.15	61.06	46.35	58.40	54.81
BERT CLS-vector	20.16	30.01	20.09	36.88	38.08	16.50	42.63	29.19
InferSent - Glove	52.86	66.75	62.15	72.77	66.87	68.03	65.65	65.01
Universal Sentence Encoder	64.49	67.80	64.61	76.83	73.18	74.92	**76.69**	71.22
SBERT-NLI-base	70.97	76.53	73.19	79.09	74.30	77.03	72.91	74.89
SBERT-NLI-large	72.27	**78.46**	**74.90**	80.99	76.25	**79.23**	73.75	76.55
SRoBERTa-NLI-base	71.54	72.49	70.80	78.74	73.69	77.77	74.46	74.21
SRoBERTa-NLI-large	**74.53**	77.00	73.18	**81.85**	**76.82**	79.10	74.29	**76.68**

Figure 3.3: Spearman rank correlation ρ between the cosine similarity of sentence representations and the gold labels for various Textual Similarity (STS) tasks. Performance is reported by convention as $\rho \cdot 100$. STS12-STS16: SemEval 2012-2016, STSb: STSbenchmark, SICK-R: SICK relatedness dataset. [3]

For the supervised approach, two strategies were tested. The first one was to fine-tune the SBERT using only the STS benchmark dataset (STSb), and the second was to first fine-tuned SBERT with the NLI dataset and only then use STS benchmark dataset (refered to as SBERT-NLI in figure 3.4). which proved to be a better option. At prediction time, they used the cosine similarity to measure the sentence embeddings similarity. The results are presented in figure 3.4.

Model	Spearman
Not trained for STS	
Avg. GloVe embeddings	58.02
Avg. BERT embeddings	46.35
InferSent - GloVe	68.03
Universal Sentence Encoder	74.92
SBERT-NLI-base	77.03
SBERT-NLI-large	79.23
Trained on STS benchmark dataset	
BERT-STSb-base	84.30 $\pm$ 0.76
SBERT-STSb-base	84.67 $\pm$ 0.19
SRoBERTa-STSb-base	**84.92** $\pm$ 0.34
BERT-STSb-large	**85.64** $\pm$ 0.81
SBERT-STSb-large	84.45 $\pm$ 0.43
SRoBERTa-STSb-large	85.02 $\pm$ 0.76
Trained on NLI data + STS benchmark data	
BERT-NLI-STSb-base	**88.33** $\pm$ 0.19
SBERT-NLI-STSb-base	85.35 $\pm$ 0.17
SRoBERTa-NLI-STSb-base	84.79 $\pm$ 0.38
BERT-NLI-STSb-large	**88.77** $\pm$ 0.46
SBERT-NLI-STSb-large	86.10 $\pm$ 0.13
SRoBERTa-NLI-STSb-large	86.15 $\pm$ 0.35

Figure 3.4: Evaluation on the STS benchmark test set. The metric employed was the Spearman correlation along with the standard deviation. SBERT was fine-tuned on the STSb dataset, SBERT-NLI was pretrained on the NLI datasets, then fine-tuned on the STSb dataset. [3]

3.3 COLIEE 2021

COLIEE is a competition that focus on establishing the state-of-the-art approaches for information retrieval and entailment for the legal domain. Despite the 2022 edition of this competition having already taken place, the systems developed by the participating teams have not yet been published, and for this reason the competition of the previous year will be taken into account.

The COLIEE 2021 competition consists of five tasks in total, that are divided into two components: case law and statute law components. The case law component integrates an information retrieval task (Task 1), and the verification of an entailment relation between an existing case and an unseen one (Task 2). The statute law component in composed by an information retrieval task (Task 3), an entailment/question answering task based on retrieved civil code statutes (Task 4) and another entailment/question answering task but this time without retrieved civil code statutes (Task 5).

In this thesis we will focus on the information retrieval task from the statue law component (Task 3). The goal of the task is to return relevant legal articles from the collection in response to a query in natural language. In the COLIEE 2021 competition a total of six teams submitted results for this task. The best one was the OvGU team. The evaluation measures used in this task were the F2 measure, precision and recall. The F2-score is defined as:

$$F2 = \frac{(1+\beta^2) \cdot Precision \cdot Recall}{\beta^2 \cdot Precision \cdot Recall}$$

where the precision is the number of true positive predictions divided by the total number of predicted positives (true positives + false positives), the recall is the number of true positive predictions divided by the total number of actual positives (true positives + false negatives), and β is a parameter that determines the relative importance of precision and recall. The F2-score ranges from 0 to 1, with 1 being the best possible score. A higher F2-score indicates a better balance between precision and recall, with a stronger emphasis on recall. Next we have a summary of the OvGU team's submission.

3.3.1 Team OvGU

This team proposes a combination of a two-stage TF-IDF vectorization with Sentence-BERT embeddings [15]. They started by enriching the training data with multiple adjustments as proposed in 3.5. In the first adjustment they added structural information to the articles using the section titles in the civil code. This helped create hierarchical relations between articles within the same section. In the second adjustment, they crawled Japanese open source commentary

	Description
Articles with metadata	training data + details regarding *Part*, *Chapter*, *Section* and *Subsection*.
Articles with crawled data	training data + translated crawled data from the website https://ja.wikibooks.org/
Articles with relevant queries	training data + queries from training data if the entailment label is *Y* for the respective article.

Figure 3.5: Data Enrichment

on the articles. In doing so, they potentially enriched the articles with general remarks, corner cases, previous versions, related articles and a reasoning for the relation. In the third and final adjustment, they enriched the training data with queries that have a positive entailment relationship. To obtain these queries they parsed the training data labels of task 4.

After applying these data enrichment techniques, they computed the TF-IDF vectors for the queries and the articles in two stages.
The first stage used articles enriched with relevant queries and metadata. The computation of the vectors relied on sub-linear term frequency scaling and L2 normalization. The vectors after the first stage yielded significant precision-recall trade-offs, which lead to lower F2 scores. To counter-balance this trade-off, the team provided a different and unique representation of the vectors, which resulted in a second stage of TF-IDF vectorization where the query and articles vectors had been created considering only the enrichment with metadata.
The sentence-BERT embeddings were created with all the enrichment techniques described in 3.5. The implementation[2] relied on Reimers et al. [3] and used the pre-trained paraphrase-distilroberta-base-v1 model to create the article and query embeddings.
In the end, they calculated the cosine similarity for each query-article pair, obtaining three different scores, one for the first stage of TF-IDF, another for the second stage and a third one for the sentence-BERT embeddings. Then proceeded to sum the three scores and normalize. Only the articles with scores above a certain threshold (a maximum of four articles) were presented as the final results. An overview of the approach can be seen in figure 3.6.

[2]https://github.com/UKPLab/sentence-transformers

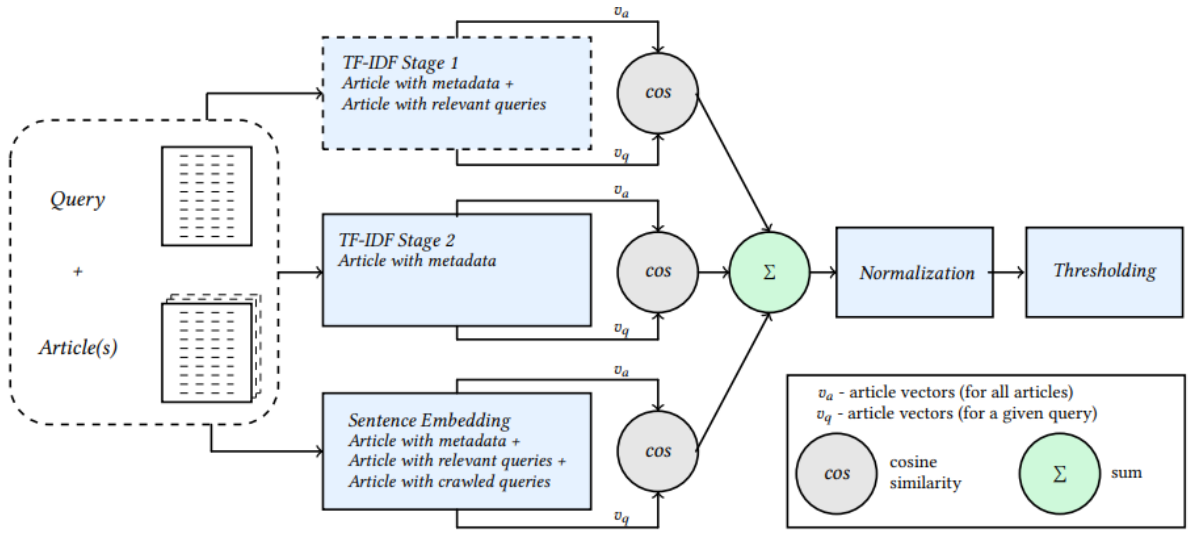

Figure 3.6: Sentence-BERT Embeddings with TF-IDF

3.4 NLP Applied To Portuguese Consumer Law

A similar problem was assigned to Nuno Cordeiro, as part of his master's thesis in 2022. In his thesis [16], he developed a new search system, entitled Legal Semantic Search Engine (LeSSE), that combines syntactic search with semantic search. The syntactic search allows users to search for literal terms (such as names and titles), and the semantic search helps in cases where the answers contain juridical jargon that wasn't used in the query. Several state-of-the-art techniques such BERT and BM25 algorithm were used, which are also relevant to our context. An overview of the approach can be seen in figure 3.7.

Before any search takes place, all the documents stored in the database go through a text segmentation phase, where the documents' text is divided by line breaks and each generated segment contains information about its location in the document. Since these documents are used in every search, doing this pre-processing step before the search actually takes place its beneficial, because it avoids repetitive computations saving time and resources. Once the documents' segments are created, both the segments and the query will go through further processing, divided in syntactic and semantic pre-processing. The segments pre-processing is done first when the system starts, and the query is done as soon as it is inserted into the search system.

The syntactic pre-processing consisted of five steps: word tokenization, removal of punctuation, word lowering, stop-word removal and unicode. This steps are required to construct the bag-of-words necessary for the BM25 algorithm. After that, the processed segments and the query will take part in the syntactic search, where the BM25 algorithm is used to compare the

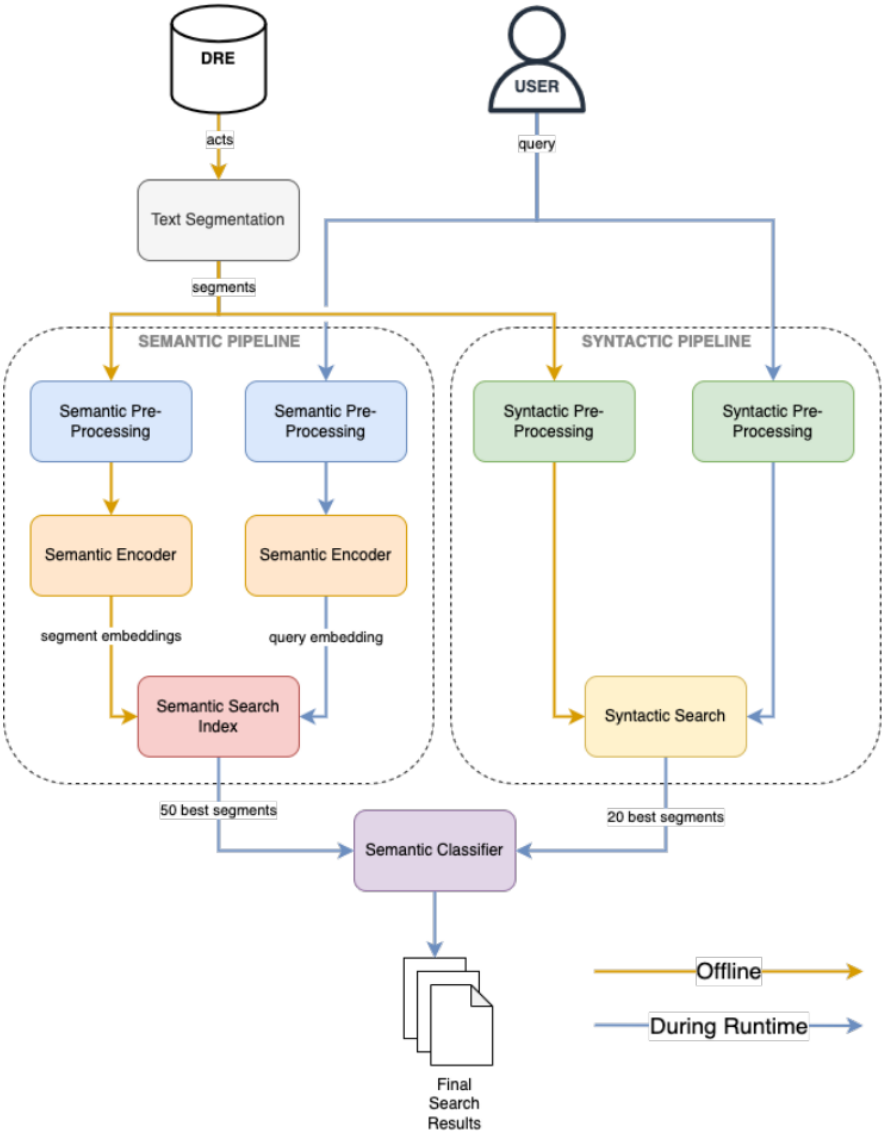

Figure 3.7: LeSSE

query with the segments, on a syntactic level. Only the top 20 segments with the highest scores are selected.

The semantic pre-processing consisted only on the BERT Tokenizer[3] from the Hugging Face library. Once this step is complete, the segments and the query are then sent to a semantic encoder, which will generate embeddings from them. The semantic encoder is BERTimbau [17], a BERT model trained on the BrWaC [18] corpus. Since this BERT model is only capable of generating token embeddings, in order to produce a segment embedding, the embeddings from all the tokens in it must be averaged into a single segment embedding.

[3]https://huggingface.co/docs/transformers/main_classes/tokenizer

Once this step is complete, a Faiss search index[4] is created with the segment embeddings and stored locally in the server that hosts the search engine. Every time a query is encoded, it is compared with the sentence embeddings to find the top 50 with the highest cosine similarity scores.

In the end, the 50 best segments from semantic search and the 20 best from syntactic search are combined and ordered through a reordering model (another BERTimbau model) to produce the final results.

[4]https://faiss.ai/

4

System Development

Contents

This thesis aims to develop a reliable search system with semantic capabilities for the legal domain. Several ideas from the systems discussed in Chapter 3 were combined into a single system. This system employs a combination of established document retrieval methods and the latest advancements in Machine Learning and Natural Language Processing. This hybrid approach enhances the search system with the ability to perform semantic searches.

This section starts with a general overview of the data corpus and the training dataset. Following that, we take a look at the system's workflow, where we will delve deeper into each individual system component, coupled with a detailed explanation of the selection, arrangement, and presentation of results to the user. In the end, we will present a detailed explanation of the training process for the language model.

4.1 Data Corpus and Training Dataset

As mentioned in the introduction, the COLIEE competition provides access to the statute law competition data corpus, which was used in the information retrieval task from the statue law component (Task 3) from COLIEE 2021.

This data corpus has in total 724 articles from the japanese civil code, structurally divided by the following hierarchical titles: Part, Chapter, Section and Subsection. Each article is composed by the ID that represents it and by the description. Example: Article 12 - A person subject to a decision for commencement of curatorship becomes a person under curatorship, and a curator is appointed for that person.

There are some articles that contain multiple paragraphs, which could potentially surpass the character limit in the language model (BERT) that is used in the developed system. Since BERT can only handle input segments with a maximum size of 512 tokens, any article that surpasses this limit will have to be divided into smaller segments. We've done this segmentation using line breaks. Each of the generated segments will contain information about the location of the original article so that they can be later referenced in the results. We will therefore have a new segmented data corpus composed by segments of the articles from the original data corpus. As we will see further on in this chapter, we will perform two training sessions, where we fine-tune the language model using the segmented data corpus in one session, and in the second session we use the original data corpus, in order to experiment if segmentation benefits the learning process of the language model.

In addition to the civil code corpus, COLIEE also provides the training dataset with queries and relevant articles from the corpus. The training data is composed in total by 806 training

samples. The format of each sample is as follows[1]:

```
<pair id="H18-9-1" label="Y">
<t1>
Article 175
No real right may be established other than those prescribed by laws including this Code.
Article 265
A superficiary has the right to use another person's land in order to own structures, or trees or bamboo, on that land.
Article 270
A farming right holder has the right to pay rent and engage in cultivation or livestock farming on another person's land.
Article 280
A servitude holder has the right to use another person's land for the convenience of their own lands in accordance with purposes prescribed in the act establishing the servitude;provided, however, that this right must not violate the provisions ( limited to those that relate to public policy) under Section 1 of Chapter 3 (Extent of Ownership)..
</t1>
<t2> Usufructuary rights are only established for real estate. </t2>
</pair>
```

```
<pair id="H19-1-3" label="N">
<t1>
Article 96
(1) A manifestation of intention based on fraud or duress is
```

[1]Examples available at https://sites.ualberta.ca/~rabelo/COLIEE2021/

```
    voidable.
(2) If a third party commits a fraud inducing a first party to
    make a manifestation of intention to a second party, that
    manifestation of intention is voidable only if the second party
     knew or could have known that fact.
(3) The rescission of a manifestation of intention induced by
    fraud under the provisions of the preceding two paragraphs may
    not be duly asserted against a third party in good faith acting
     without negligence..
</t1>
<t2>
A person who made a manifestation of intention which was induced
    by duress emanated from a third party may rescind such
    manifestation of intention on the basis of duress, only if the
    other party knew or was negligent of such fact.
</t2>
</pair>
```

Here we have two training samples, with the ids ”H18-9-1” and ”H19-1-3”. The tag t2 is a query sentence and the tag t1 shows corresponding civil code articles, in the sample ”H18-9-1” there are four relevant articles and in the sample ”H19-1-3” there is just one. In every sample there is a minimum of one and a maximum of four articles.

The training dataset provided is destined for two tasks from the COLIEE competition. The first task is the information retrieval task, where the objective is to find relevant articles to a given query sentence. In the examples above the relevant articles are 175, 265, 270 and 280 for the query of the first example and 96 for the second. The second task is an entailment task, where the goal is to answer Yes/No to the query sentence. If the label in the tag pair is ”Y”, it means the answer to the query sentence is ”Yes”. Otherwise, if the label is ”N”, it means the answer is ”No”. In the example of ”H19-1-3”, based on the entailment from the corresponding article 96, the answer to the query sentence is ”No”. In example ”H18-9-1”, since the label is ”Y”, this means the answer to the query sentence is ”Yes”. There are in total 409 samples labeled ”Y” and 397 labeled ”N”.

In this thesis, since our focus is to extract relevant articles, the labels become irrelevant. It is worth noting that we will use all 806 samples, which includes both samples labeled ”Y” and

”N”. This is based in the intuition that articles from samples labeled ”N” are just as relevant to the associated query as the samples labeled ”Y”, since in both scenarios those articles possess the necessary information to answer the query.

4.2 System Overview

Similarly to [16], the first step was to split the legal documents from the data corpus into smaller text segments. In figure 4.1 this is designated as text segmentation. As we previously explained, this step was taken as a precautionary measure, despite no articles exceeded the inherent 512-token limitation of BERT models. Moreover, this served to test whether segmenting the articles would enable a more detailed comparison with the query, and thereby increasing the likelihood of capturing the similarity between the segment and the query. This approach aimed to enhance the model's performance in accurately matching queries with relevant content. The performance of this approach will then be compared with the model trained using the original data corpus (without segmentation). Throughout this chapter we focus mostly on the segmented dataset, however the same procedures are applied to the articles of the original dataset, with exception of the segmentation phase. Therefore when we describe a procedure applied to a 'segment', it can also be rewritten to an 'article'.

During the text segmentation phase, the text of each article is divided into smaller segments separated by line breaks. These segments will contain information about the location of the article from which they came from, so that they can be later referenced in the results. Notice that the segmentation will only be done with the data corpus and not the training dataset. This is simply to avoid unnecessary work, since we will use the articles IDs from the training samples to extract them from the data corpus.

After that, the generated segments go through Lexical and Semantic pre-processing. Both segmentation phase and pre-processing of the documents are executed prior to the initiation of any search. This preemptive action conserves time and resources by avoiding redundant computations, as all results are stored in the database for future use. While the search query also undergoes pre-processing, this procedure is carried out during the search time due to the inevitable variance in queries with each search.

The semantic pre-processing is the first phase of the semantic pipeline, which generates tokens to be utilized by the semantic encoder, more specifically the Sentence-BERT model. These tokens are transformed into embeddings, forming the foundation for the semantic search process. Following this, the semantic search uses these embeddings to compute the relevancy scores of each segment, effectively ranking them based on their alignment with the search query.

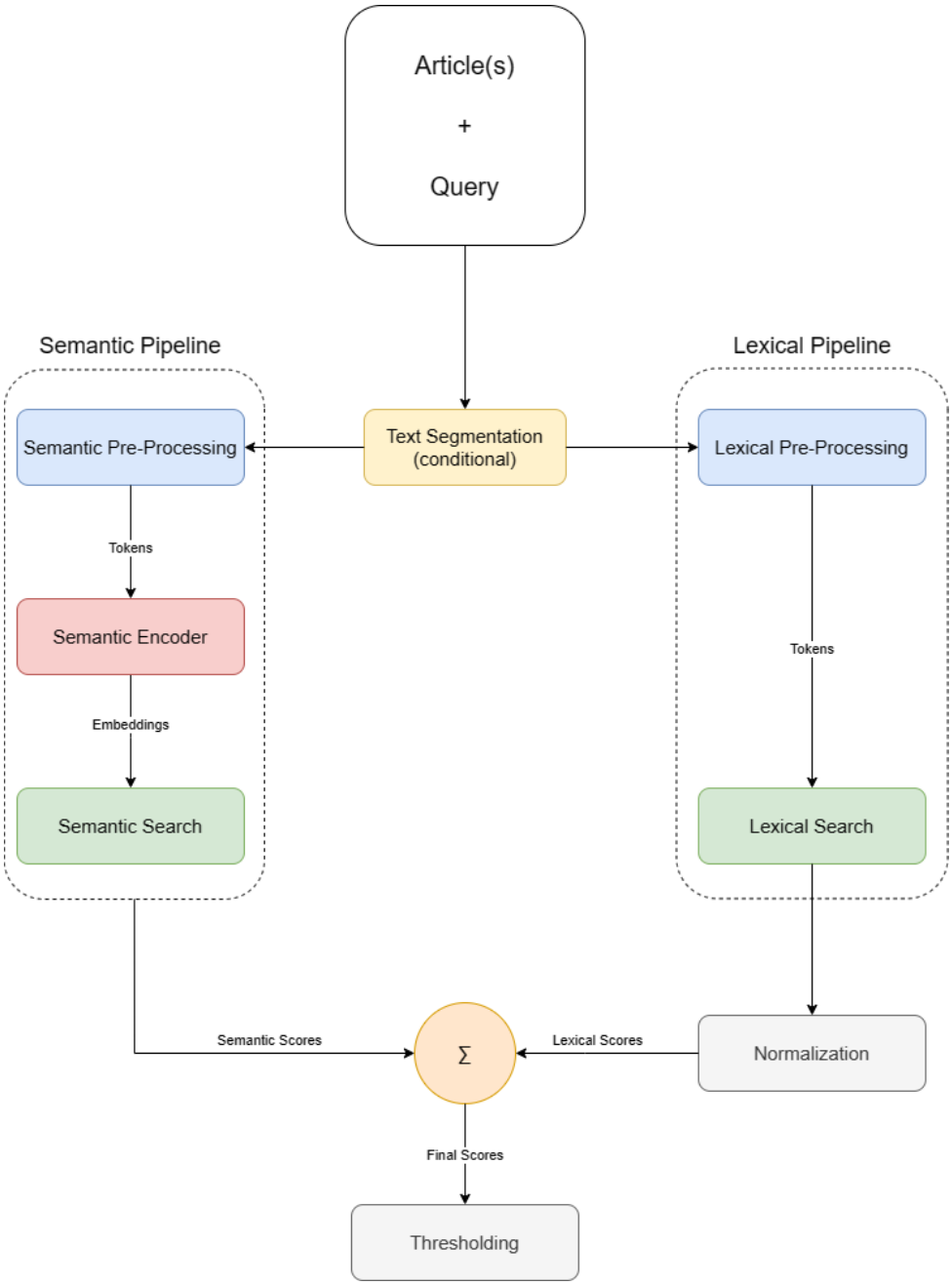

Figure 4.1: Proposal System Architecture

Similarly to the semantic pipeline, the lexical pre-processing is the first step in the lexical pipeline, which generates tokens for utilization in the lexical search. This search leverages the BM25 algorithm to compute the relevancy scores of each segment. This process ranks the segments based on their correspondence with the search query in a lexical context.

During the search phase, the query is processed through lexical pre-processing and subsequently used to compute a similarity score for each segment. In the semantic pipeline, the query is put through the semantic pre-processing stage and is transformed into an embedding, which is then utilized for the search. This process results in the generation of semantic similarity scores between each segment and the query, based on the likeness of their corresponding embeddings.

In the concluding phase, the scores from the semantic and lexical searches of each segment are summed. However, as these scores operate on different scales and therefore cannot be directly summed, they first require normalization. Following this, the results are sorted based on the sum of these normalized scores and subsequently presented to the user.

4.2.1 Lexical Pipeline

Before initiating the lexical similarity search, it's necessary to process both the segments and query. This ensures that the system can recognize them even if they're written differently than their dictionary representation. The lexical similarity search in this system employs the BM25 algorithm, which follows a bag-of-words approach. Essentially, this means breaking a phrase into individual words, and then comparing the words from the query with the words in the corpus. For two words to be identified as identical, they must consist of the same sequence of Unicode3 characters.

Due to this requirement, the lexical search needs to undergo a series of lexical pre-processing steps, consisting of five main actions:

1. Word Tokenization with the Natural Language Toolkit (NLTK): Here, each segment and query is divided into tokens (essentially words) using NLTK's word tokenization tool. This helps us form the bag-of-words needed for the BM25 algorithm.

2. Removal of Punctuation: As the previous step doesn't eliminate punctuation, all punctuation characters are removed to prevent them from being counted as words.

3. Word Lowercasing: Words that have uppercase letters are converted to lowercase. This ensures that a word in the user's search query, even if it mistakenly starts with an uppercase letter, can be matched with words in the corpus.

4. Stop-Word Removal: To conduct a meaningful keyword search on the corpus, both the corpus segments and the query undergo a stop-word removal process. Stop-words, by definition, don't add extra meaning to sentences. We use NLTK's stop-word dictionary to eliminate all irrelevant words. This involves retaining only the words that aren't found in the stop-word corpus. This step is crucial in text processing because these words frequently appear in the English language, and the similarity between two phrases (segment and query) shouldn't be assessed based on their count of stop-words, which don't provide any contextual information about the phrase.

5. Unidecode: This final step ensures that none of the words in the segments or the query contain any special characters. Each word is processed through the unidecode function from the Unidecode library.

At the end of the lexical pipeline we have the lexical search. Here, a lexical comparison is made between the query and the segments by utilizing the BM25Okapi[2] class from the rank bm25 Python library. Before any search occurs, the BM25Okapi object is initialized with the corpus, comprising all pre-processed segments, during the system's initialization stage. Once the search is complete, a lexical score is attributed to each segment based on the comparison made with the query.

4.2.2 Semantic Pipeline

In the semantic search pipeline, the pre-processing phase includes the use of a DataLoader, which is an essential component for model training. The DataLoader is primarily used for two critical tasks: batching and shuffling of the data. Batching allows us to group multiple data points together, enabling the model to process several data instances simultaneously during training, thereby improving computational efficiency. Shuffling, on the other hand, randomizes the sequence of data points, preventing the model from learning any unintended sequential patterns that could bias the training process. Additionally, the DataLoader transformed the data into a format that is compatible with the model's fit() function from the SentenceTransformers library [3], which is responsible for model training.

Following the DataLoader's role in batch formation and data shuffling, another step occurs in the form of tokenization. The model's fit() function uses the BERT Tokenizer[3], which takes the batched data and converts it into a format that is suitable for the BERT model. Basically,

[2]https://pypi.org/project/rank-bm25
[3]https://huggingface.co

the Tokenizer breaks down our text data into individual tokens, and then converts them into embeddings that are fed to the semantic encoder.

The semantic encoder is the "multi-qa-mpnet-base-cos-v1"[4] SBERT model. This model is responsible for generating sentence embeddings able to capture the semantic information of the segments and queries. Similarly to the pre-processing phase, the segments' embeddings are generated before any search takes place and they are stored in the database for future use. On the other hand, the queries' embeddings are generated dynamically whenever new queries are received as input. This allows for efficient retrieval of relevant information when a query is made.

The "multi-qa-mpnet-base-cos-v1" model ensures that the generated embeddings effectively represent the semantic similarity between segments and queries, enabling accurate matching and retrieval of relevant information. It was trained on 215M question-answer pairs from various sources and domains, including StackExchange, Yahoo Answers, Google and Bing search queries. The training was made with Multiple Negatives Ranking Loss using Mean-pooling and cosine-similarity as similarity function.

To adapt the SBERT model to the specific vocabulary used in the law documents and queries, fine-tuning was necessary. The objective was to train the model to recognize common semantic associations between queries and segments. For this purpose, the fit() function was employed, which handles the entire fine-tuning and evaluation process. The chosen loss function was also the Multiple Negatives Ranking Loss, since this is the most suited loss function for our context. This loss function works great to train embeddings for retrieval setups where we only have positive pairs (such as query and relevant document) as it will sample n-1 negative pairs in each batch randomly (notice that positive and negative pairs refer to the existence or not of a semantic relation between the pair, for example, if a sentence answers a certain query than it is a positive pair, even if the answer to the query is "No"). The performance using this loss function usually increases with increasing batch sizes. Through this fine-tuning process, the model was able to learn and capture the nuanced semantic relationships between queries and law segments, enhancing the accuracy and effectiveness of the semantic search system.

Once the query embeddings are generated, the system proceeds to compute the semantic scores for all segments of the corpus for each query based on cosine similarity.

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

4.2.3 Results Selection

In the end, we sum the scores from both pipelines for each segment. Each has two different similarity scores from the semantic and lexical searches. These scores, however, don't share the same scale, so they need to be normalized. They were normalized to be within 0 and 1, corresponding to the scale of the scores calculated with the cosine similarity between embeddings. The scores from the BM25 algorithm were normalized with the min-max normalization. Finally, once the scores are in the same scale, they are summed together.

Since in COLLIE 2021 test data none of the queries has more than 4 relevant articles, the threshold value to filter out the top n relevant articles for each query was the top 4 articles.

In order to determine a suitable threshold for the article scores, [15] discovered that an index-based approach yielded better results compared to using a single threshold value for the entire set. We followed the same strategy, where we consider the article at the 1st index as the best article, and establish a threshold of 0.91 or higher of the score obtained by the best article in order to return a second article. This means that we will always show the most relevant article found for the given query, and we will only show the second best article if its score is 91% or higher of the best article's score. We also established the threshold of 0.85 or higher for articles found at subsequent indices. As mentioned the limit for the retrieved articles is 4.

4.2.4 Model Training

As part of the SBERT model fine-tuning process, we conducted a hyper-parameter tuning process to enhance the learning capabilities of the model and maximize its performance on previously unseen data. This was done using the Ray Tune python library[5], a library widely used for distributed hyper-parameter tuning, that helps finding the best combination of parameters to improve the performance of machine learning models. It includes the latest hyper-parameter search algorithms such as Population Based Training (PBT).

4.2.4.A Model Optimization

In order to determine the most effective data type for training this model, we explored two training datasets: one featuring the original articles from the data corpus and another with segmented articles. The decision for the most optimal training dataset was made by applying a uniform evaluation process to both. Initially, we split each dataset into three subsets: a training set for model training; a validation set for assessing model performance throughout the training

[5]https://docs.ray.io/en/latest/tune/index.html

process; and a test set for evaluating the model's performance on new data after the training is complete. The datasets were randomly split following the percentage-split approach: 70% for training, 20% for evaluation and the other 10% for testing.

Training Set (70%):

We gave most of the data, 70%, to the training set. This is where the model learns from the examples we have. The more data it has to learn from, the better it can understand different patterns and do a good job on new sentences.

Evaluation Set (20%):

We set aside 20% of the data for evaluation. This part is used to decide if the model is getting better during training. We made it bigger than the test set because our model saves its progress after each epoch only if it achieves a higher evaluation score than before. Having more evaluation examples lets us be more sure about the model's improvement.

Test Set (10%):

Finally, we kept 10% of the data as a test set. The model never sees this data while it's learning. It's like a final exam for the model. We use it to see how well the model can work on new, unseen sentences.

Regarding the method of evaluation, the SBERT library offers multiple evaluation methods that can be used to evaluate the model during training. We chose the Information Retrieval Evaluator, which seems to be the most suited method, since our objective is to retrieve relevant articles. Nevertheless, configuring this evaluator involved some data manipulation because the input structures it needed were three different dictionaries that required the following format:

Queries: Dict[str, str], (query id – query text)

Corpus: Dict[str, str], (article id – article text)

Relevant Articles: Dict[str, Set[str]], (query id – Set[article id])

The dataset with the original articles was not an issue, but the segmented dataset required more preparation regarding the corpus dictionary, since instead of just one article text we had multiple segments for one single key, which is impossible. So new keys were created, one for each segment.

Once both datasets were ready, we applied the PBT tuning function from the Ray Tune library on each dataset individually, to find the best hyper-parameters for the SBERT model. The chosen parameters along with their corresponding model performance are exhibited in tables 4.1 and 4.2. The best hyper-parameters for each dataset were chosen according to the scores returned by the evaluators. These scores were determined after every epoch, relying on the F2-scores computed using the evaluation dataset. These results were achieved using the Adam optimizer with a batch size of 16 and a total of 50 epochs. The optimization was done with a machine with

Warmup Steps	Learning Rate	Weight Decay	Total Time (s)	Score
20	7.278e-04	3.54e-03	376.72	0.46
25	1.41425e-06	0.00141618	357.043	0.505795
15	0.00423863	1.01155e-05	353.383	0.0104166
25	3.53185e-05	0.00178533	408.662	0.599241
20	0.000527638	0.018277	560.259	0.510428
20	6.98678e-06	0.0690479	352.694	0.558441
15	2.37852e-06	0.0357856	352.611	0.511946
24	8.38413e-06	0.0552383	381.552	0.567375
10	0.0128065	0.00231253	338.841	0.00624998
20	2.03408e-06	4.46288e-05	338.798	0.506869
25	7.43204e-05	1.18305e-06	336.051	0.624067
5	1.13627e-05	8.35368e-05	400.264	0.586016
20	0.000117189	0.0341678	345.045	0.636006
15	0.00242922	0.0204541	347.343	0.00624998

Table 4.1: Optimization results with original dataset

1 NVIDIA GeForce RTX 3080 Ti Graphics Processing Unit (GPU), with 12 GB of memory.

After the SBERT hyper-parameters optimization we also performed a manual tuning of the BM25 parameters for the k1 and b variables. However no change was needed, since the default values k1=1.75 and b=0.75 provided the best results among all the values tested.

Warmup Steps	Learning Rate	Weight Decay	Total Time (s)	Score
15	5.49557e-06	1.65412e-05	1140.12	0.302465
15	7.25663e-06	2.00469e-05	1137.88	0.307929
10	3.2815e-05	2.34815e-05	1144.43	0.387205
25	1.00527e-05	0.000283041	1153.38	0.340459
15	2.6252e-05	1.87852e-05	1154.91	0.441065
20	5.06496e-06	2.34047e-05	1132.94	0.307303
10	1.35188e-05	0.000968559	1142.3	0.363419
8	1.0815e-05	0.000774847	1140.49	0.322712
25	1.38242e-05	0.000482393	1133.27	0.361111
25	2.26339e-05	2.44292e-06	1133.92	0.379225
15	2.14e-05	2.35203e-06	621.163	0.343096
25	4.77961e-05	5.06075e-05	658.695	0.354582
20	3.82369e-05	4.0486e-05	658.497	0.348743
20	1.7842e-05	3.58032e-05	668.483	0.362261
15	3.0323e-05	1.77788e-05	659.009	0.359656
30	5.03091e-05	1.78093e-06	650.749	0.390855
25	6.52722e-05	1.6197e-05	650.235	0.33195
20	1.00619e-05	3.18337e-06	652.348	0.379393

Table 4.2: Optimization results with segmented dataset

4.2.4.B Training Conclusion

In this section, we draw conclusions from our training efforts and present our findings based on the evaluation scores and mean loss values obtained during the fine-tuning of the SBERT model. As mentioned before, we conducted two distinct training sessions, one using the original dataset and another using a segmented version of the dataset. By comparing the results, we gained insights into the model's behavior and performance under different training conditions, while also taking into account the specific hyper-parameters used for each training session.

Our first training approach involved utilizing the original dataset to train the SBERT model. This training session was conducted with the following hyper-parameters, derived from the Ray Tune optimization process:

Warmup Steps: 20

Learning Rate: 0.00011718867818415094

Weight Decay: 0.03416775724773422

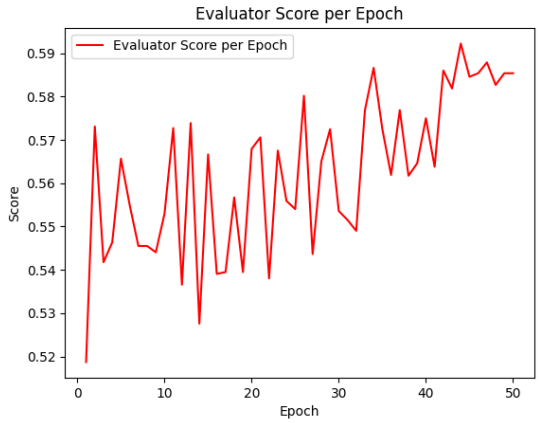

Figure 4.2: Evaluator Score per epoch

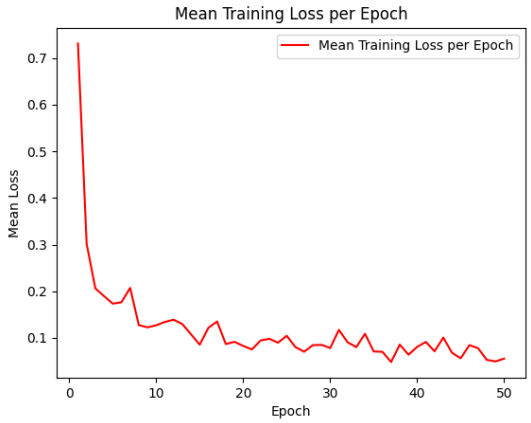

Figure 4.3: Average Batch Loss per Epoch

Figure 4.4: Fine-tuning with original dataset

Through a training process spanning 50 epochs, we observed several noteworthy trends. The evaluator score, which reflects the model's performance in terms of the F2-score metric, exhibited an overall improvement with constant fluctuations throughout the training process (Figure 4.2). It was not until approximately the 45th epoch that the model achieved its highest evaluator score. These observations suggest that, with these specific hyper-parameters and the original dataset, the model could potentially achieve even higher scores with training extending beyond the 50 epochs. The mean loss, indicative of the model's convergence and learning progress, demonstrated a remarkable decrease during the initial epochs (Figure 4.3). The loss

reduction was particularly rapid in the early stages, indicating that the model quickly adapted to the data. However, after the 20th epoch, the reduction in mean loss became much less significant, signifying a stabilization in the model's learning curve. This stabilization suggested that the model had reached a certain level of optimization.

In contrast, our second training approach involved using a segmented version of the dataset. This approach aimed to investigate the impact of dataset segmentation on the SBERT model's training dynamics. The training was conducted with the following hyper-parameters:

Warmup Steps: 15

Learning Rate: 2.6251961553012977e-05

Weight Decay: 1.8785212995084844e-05

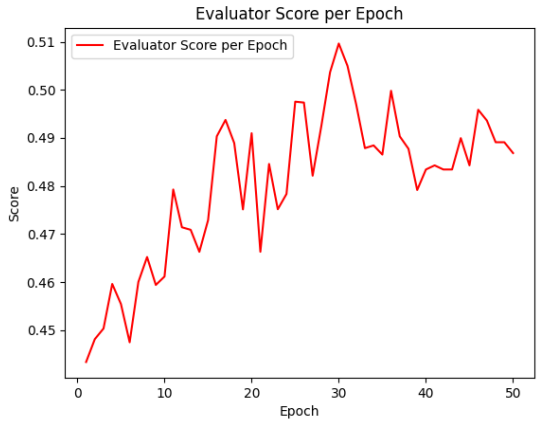

Figure 4.5: Evaluator Score per epoch

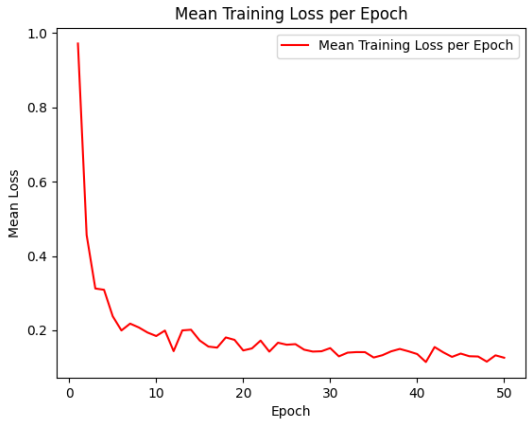

Figure 4.6: Average Batch Loss per Epoch

Figure 4.7: Fine-tuning with segmented dataset

The results from this training session, led to the following conclusions: this dataset allowed the model to achieve its highest evaluation score much earlier in the training process, around the 30th epoch (Figure 4.5). However, even though it converged faster, the highest score and the overall scores observed throughout the 50 training epochs was considerately less when compared with the previous training session. Similar to the original dataset training, the mean loss for the segmented dataset training also exhibited rapid reduction in the initial epochs (Figure 4.6). However, the decrease in mean loss was less pronounced after approximately 6 epochs. This trend mirrored the evaluation scores, signifying that the model with the segmented dataset reaches a stable learning state sooner than the previous session at the cost of overall lower performance.

The comparative analysis of these two training sessions illustrates a trade-off between training efficiency and model performance. Training with the original dataset allowed the model to

eventually achieve higher evaluation scores and a lower loss, but it required a more extended training duration. On the other hand, training with the segmented dataset resulted in faster convergence and earlier stabilization but yielded lower overall performance scores. These findings underscore the importance of considering dataset choice in the context of model training. The decision should align with the specific goals of the task at hand, balancing the need for rapid convergence with the pursuit of optimal model performance.

The entire code for the developed system, spanning from data preparation to the fine-tuning process, is accessible in the GitHub repository ”https://github.com/GonmasPT/tese.git”.

5

Evaluation

This chapter presents an assessment of the search system we developed and fine-tuned when using each of the two training datasets, the original and the segmented datasets. The evaluation methodology was focused on measuring the system's precision, recall, and F2-score on the test data.

For the original training dataset, the search system achieved a precision of 0.54, a recall of 0.66, and a F2-score of 0.64. These results indicate a moderately high degree of relevance and completeness in the retrieved results. However, when the model was fine-tuned on the segmented dataset, it exhibited a reduced performance, yielding a precision of 0.26, a recall of 0.50, and an F2-score of 0.42.

These outcomes align with the consistent pattern observed during the model's training process. When the model underwent fine-tuning with the segmented dataset, there was a notable reduction in both the Mean Loss per Epoch and the Evaluator Score per Epoch. Consequently, it was expected that the final evaluation scores on the test data would also reflect lower values when using the segmented dataset.

While interpreting these results, it is important to note that the performance was likely hindered due to hardware limitations. Specifically, the graphics card memory available for the study only allowed training the model with a batch size up to 16. Any attempt to increase the batch size led to an error indicating that CUDA was out of memory. Greater batch sizes could potentially improve the model's learning process and thus, its performance might have been enhanced if larger batch sizes could have been used.

To contextualize our search system's performance, it is important to compare it with the top performing team from the COLIEE 2021 competition, since our model was trained on the same dataset. The best run from the top-performing team achieved a precision of 0.67, a recall of 0.78, and an F2-score of 0.73. This performance surpasses that of our system on both datasets.

It's worth noting that the top-performing team at COLIEE leveraged multiple techniques to enrich their training data, which likely contributed to their superior results. Thus, it might be beneficial for future research to explore similar data enrichment techniques or alternative methods of fine-tuning, especially considering the hardware constraints faced during the current study. This could potentially yield a semantic search system that performs at the same levels as the top-performing systems in competitions like COLIEE. In addition, the system created by this team differed slightly from ours, featuring two stages of lexical search rather than just one, which may also explain their improved results.

Furthering our evaluation, we conducted a comparative analysis regarding the model trained with the original dataset between the performance of the semantic pipeline, the lexical pipeline, and the complete model integrating both pipelines. The results of this analysis are illustrated in

	Precision	Recall	F2-Score
Lexical Scores	0.45	0.47	0.46
Semantic Scores	0.50	0.75	0.67
Combined Scores	0.54	0.66	0.64

Table 5.1: Comparison between scores

table 5.1.

Interestingly, while the complete model outperformed the lexical pipeline that merely relied on the BM25 algorithm, the semantic pipeline alone achieved slightly superior overall results compared to the complete system. This unexpected result may be attributed to the methodology employed for combining the scores from both pipelines.

In our current system, we aggregate scores from both pipelines with equal weighting. We then select the best four segments using the index-based approach mentioned in the previous chapter. This strategy may not optimally leverage the strengths of each pipelines and could explain why the complete model did not outperform the semantic pipeline. The divergence in scores from both pipelines suggests that assigning higher weights to the semantic pipeline, rather than maintaining a 50/50 balance with the lexical pipeline, could lead to better final scores.

In the next chapter Conclusions we will use the insights gained with this chapter and we'll further discuss the implications of our findings and outline potential directions for future research.

6

Conclusion

This thesis aimed to develop a superior search system for the legal domain, one that surpasses traditional literal search methods, such as the BM25 algorithm. The motivation behind this study came from the aspiration to provide an enhanced user experience in searching legal databases, particularly governmental digital gazettes, which are repositories that contain all laws and norms of the nation.

Our efforts culminated in a search system that combines semantic and lexical search. As we saw in the previous chapter, in table 5.1, the combined system demonstrated improved performance compared to relying solely on lexical search, thereby meeting our main objective. It indeed indicates that incorporating semantic search systems can enhance the efficiency of legal data retrieval in digital gazettes, thus, making legal research more accessible and straightforward for users.

Nonetheless, the combined system slightly under performed compared to the standalone semantic pipeline. This unexpected outcome can be traced back to our method of integrating the scores from both pipelines. Within our existing system, we combine scores from both pipelines evenly, and subsequently, we employ the index-based approach outlined in the previous chapter to identify the top four segments. However, it's possible that this strategy doesn't fully harness the individual strengths of each pipeline, potentially accounting for the complete model falling short of outperforming the semantic pipeline. The significant difference in scores between the two pipelines hints that allocating greater weight to the semantic pipeline, instead of maintaining an equal 50/50 distribution with the lexical pipeline, might result in improved final scores. This highlights the importance of fine-tuning the integration strategy to achieve optimal results, which can be an exciting area for further research.

Our model consisted of a simplified approach inspired by the top-performing team's methodology in COLIEE. This led us to develop a hybrid search system that incorporated both a semantic and a lexical pipeline, incorporating a SBERT model and the BM25 algorithm. The rationale behind this choice was rooted in the understanding that our potential users, particularly legal professionals, might employ highly specific and formal legal terminology in their searches. In such instances the BM25 algorithm was expected to be more advantageous than a purely semantic system.

The queries used for testing were sourced directly from COLIEE, reflecting queries that legal professionals commonly employ in their research. However, our findings and performance evaluations suggest that even within this context, where domain-specific language is prevalent, a semantic search system, if optimized appropriately, might offer superior results. This realization underscores the potential for further refinement and enhancement in our search system's architecture, for example, as previously mentioned, possibly favoring an increased focus on the

semantic pipeline to better cater to the specific needs of legal professionals.

In the current world of natural language processing, it's also worth noting that advanced large language models (Large Language Model (LLM)s) like GPT-4, Llama-2, and Claude 2 have reached remarkable levels of sophistication. These models exhibit an exceptional ability to grasp and comprehend semantic relationships between words, demonstrating their proficiency in understanding context and meaning. Consequently, as time progresses and increasingly powerful LLMs continue to emerge, traditional lexical approaches are progressively diminishing in relevance. The robust semantic capabilities of these cutting-edge models highlight the evolving landscape of information retrieval and underscore the potential for semantic search systems to increase the performance gap with their lexical counterparts.

Another noteworthy observation derived from our model's evaluation and fine-tuning was the significant performance gap between the model trained on the segmented dataset and the one trained on the original dataset. Initially, we segmented the data for two primary reasons. First, we were concerned about the 512-token limit inherent to BERT models. However, we later confirmed that this constraint didn't pose an issue, as the original dataset worked effectively. The second motivation for segmentation was the belief that it would enhance the model's ability to extract semantic connections between relevant articles and queries by eliminating irrelevant text.

The expectation was that by segmenting the articles, the model would focus on the article segments most pertinent to the query. Unfortunately, the results led us to a different conclusion. It appears that, despite successfully associating the most relevant segment with the query, the model also associated the less relevant segments of the same article, potentially reducing the overall relevance. A possible solution could involve employing a separate model to determine the single most relevant segment for a given query within the segmented articles, ensuring a more precise match between queries and article segments.

Additionally, we recommend future studies to explore methods for enriching the training data. The comparison with the top-performing team at the COLIEE competition, which utilized several data enrichment techniques, suggests this could be a promising strategy to enhance the performance of the semantic search system. It would also be worth exploring different SBERT models, primarily because these models often exhibit unique strengths that can be particularly well-suited to specific contexts or use cases.

It is important to notice, that despite promising results, our research encountered hardware limitations, particularly due to the 12 GB memory constraint on the available graphics card. This restriction impacted the batch size during training, potentially limiting the system's learning capacity and overall performance. However, future research efforts with access to more robust

hardware resources can address this constraint, potentially elevating the system's performance.

In conclusion, our study reveals the potential of semantic search systems to enhance legal data retrieval efficiency. These insights lay a good groundwork for future research in the field of legal data retrieval. While our work has already demonstrated marked improvements over traditional literal search methods, it has also revealed untapped opportunities for further optimization and enhancement. This knowledge serves as a sturdy foundation upon which future investigations can build, propelling the development of increasingly efficient and effective legal search systems.

Bibliography

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 4171–4186. [Online]. Available: https://doi.org/10.18653/v1/n19-1423

[3] N. Reimers and I. 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, 11 2019. [Online]. Available: https://arxiv.org/abs/1908.10084

[4] A. Goker and J. Davies, *Information Retrieval : Searching in the 21st Century*. Chichester U.K: Wiley, 2009.

[5] C. Sansone and G. Sperlí, "Legal information retrieval systems: State-of-the-art and open issues," *Information Systems*, vol. 106, p. 101967, 2022.

[6] K. T. Maxwell and B. Schafer, "Concept and context in legal information retrieval," in *Legal Knowledge and Information Systems*. IOS Press, 2008, p. 3.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[8] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Lukasz Kaiser, S. Gouws,

Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016.

[9] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu, "Population based training of neural networks," 2017.

[10] S. E. Robertson, S. Walker, K. S. Jones, and M. M. Hancock-Beaulieu, "Okapi at TREC-3," *Proceedings of the Third Text REtrieval Conference*, 1994.

[11] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning, "A large annotated corpus for learning natural language inference," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 632–642. [Online]. Available: https://aclanthology.org/D15-1075

[12] A. Williams, N. Nangia, and S. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 1112–1122. [Online]. Available: https://aclanthology.org/N18-1101

[13] D. Cer, M. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia, "SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, pp. 1–14. [Online]. Available: https://aclanthology.org/S17-2001

[14] M. Marelli, S. Menini, M. Baroni, L. Bentivogli, R. Bernardi, and R. Zamparelli, "A sick cure for the evaluation of compositional distributional semantic models," in *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, 2014, pp. 216–223.

[15] S. Wehnert, V. Sudhi, S. Dureja, L. Kutty, S. Shahania, and E. W. De Luca, "Legal norm retrieval with variations of the bert model combined with tf-idf vectorization," in

Proceedings of the eighteenth international conference on artificial intelligence and law, 2021, pp. 285–294.

[16] N. Cordeiro, "NLP applied to portuguese consumer law," *Master's thesis, Instituto Superior Técnico*, 2022.

[17] F. Souza, R. Nogueira, and R. Lotufo, "Bertimbau: pretrained bert models for brazilian portuguese," in *Brazilian conference on intelligent systems*. Springer, 2020, pp. 403–417.

[18] J. A. Wagner Filho, R. Wilkens, M. Idiart, and A. Villavicencio, "The brwac corpus: a new open resource for brazilian portuguese," in *Proceedings of the eleventh international conference on language resources and evaluation (LREC 2018)*, 2018.