Named Entity Recognition model based in Convolutional Neural Networks for automatic processing of Chilean environmental legal text documents

Aníbal Ayala Raso

July 29, 2022

Profesor Guía

Harvey Rosas Quinteros. Ph.D.

Instituto de Estadística, Universidad de Valparaíso

Tesis para optar al:

grado académico de: *Magíster en Estadística*

Dedication

to my family:

my beloved children and
Lida for all the patience and love.

Acknowledgements

My most sincere thanks to the company PREVSIS for the cooperation provided to carry out this thesis.

In turn, thanks to Professor PhD Harvey Rosas Quinteros for the trust in me to carry out a challenge like this.

Contents

Resumen

Esta tesis se basa en un proyecto de técnicas aplicadas de Procesamiento del Lenguaje Natural (NLP) para procesar texto legal ambiental chileno. El proyecto se realizó en cooperación con PREVSIS, una empresa chilena de prevención de riesgos que ofrece a terceros software de gestión de riesgos. De sus actividades surge una situación común con los documentos legales, específicamente con el texto legal ambiental, que desafía a las empresas a mantenerse actualizadas con los nuevos textos legales que se publican. El trabajo ha buscado resolver el problema entrenando un modelo NER personalizado utilizando el *framework* spaCy y Gensim. El modelo resultante puede anotar 17 etiquetas relacionadas con: la estructura del texto legal, acciones legales y acciones legales ambientales, entre otros. El corpus utilizado para entrenar el modelo personalizado se basó en las leyes ambientales chilenas, decretos y resoluciones publicadas a la fecha. Luego, el modelo NER fue empaquetado para ser utilizado en una aplicación web desarrollada con Streamlit como producto final de este trabajo.

Palabras claves

NLP, NER, Legal text, Environmental, spaCy, Streamlit, Gensim, CNN, Deep Learning.

Abstract

This thesis lies on an applied Natural Language Processing (NLP) techniques project to process Chilean environmental legal text. The project was done in cooperation with PREVSIS, a Chilean risk prevention company that offers to third parties risk management software. From their activities arises a common situation with legal documents, specifically with environmental legal text, that challenges companies to keep updated with new legal text that is published. The work has looked to solve the problem by training a custom NER model using the spaCy framework and Gensim. The resultant model can annotate 17 labels related to legal text structure, legal actions, and environmental legal actions, among others. The corpus used to train the custom model was based on environmental Chilean laws, decrees and resolutions published to date. Then, the NER model was packaged to be used in a web application developed with Streamlit as a final product of this work.

Keywords

NLP, NER, Legal text, Environmental, spaCy, Streamlit, Gensim, CNN, Deep Learning.

Chapter 1

Introduction

This thesis lies on an applied Natural Language Processing (NLP) techniques project to process Chilean environmental legal text. The project was done in cooperation with PREVSIS, a Chilean risk prevention company that offers to third parties risk management software. From their activities arises a common situation with legal documents, specifically with environmental legal text, that challenges companies to keep updated with new legal text that is published.

Usually, in NLP, this type of project involves using diverse technologies and tools related to Artificial Intelligence. However, it is a fact that laws, decrees, resolutions, and legal text, in general, are always an exciting field of Natural Language Processing to achieve because at least three situations arise:

1. Commonly, legislations are a wide piece of semi-structured text quite difficult to follow and understand.
2. Usually, authorities (government and legislators) publish new norms or resolutions under the frame of current legislation that are difficult to achieve due to the vast background needed to get the implications of those new norms.
3. Those texts are often unclear about subjects and other grammatical structures and always interpretable, except for legal actions.

However, legislations are unique to each country, and there is a vast difference in structure, used verbs, and semantic and syntactic dependencies, among others. That situation constrains the use of available NLP tools because most of them were trained (models) in other languages and legal structures and with different objectives than the pursuit in this work. To date, there is no work on environmental legal Chilean documents. Furthermore, when it talks about training a model, it involves different steps, each with difficulties depending on the framework used. Therefore, it will vary the technology and how to solve the problems.

The latter motivates to next main activities:

1. Afford a problem related to Natural Language Processing (NLP) of analysing the Chilean environmental legal text, which results can be seen in Section 4.1.
2. Train a custom Named Entity Recognition (NER) model to recover legal actions from the text, which results can be seen in Section 4.2 and 4.3.

3. Package and prototype the trained model; this way, we put in value all the previous processes. In Section 4.4 are presented the results.

Since the motif was stated and main activities fixed, the work started in an exploratory fashion where different tools were looked at. From that search, some tools were defined to be used. The `spaCy` framework and other related NLP tools, like `Gensim` and `Streamlit`, were finally selected. The first is the framework that allows training a custom model to annotate named entities (NER), and the second prepares the word embeddings necessary to improve the training process. The latter allows visualising the prototype capabilities in a friendly way by developing a simple web application.

Chapter 2

Background framework

2.1 Machine learning overview

2.1.1 Historical review

Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and by the use of data. A more formal definition of machine learning was also given by Mitchell (1997), as: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E".

Machine learning is based on fields such as information theory, logic gates, non-numerical computer programming and data structures; also it is closely related to Artificial Intelligence (for some researchers as a sub-field (Breiman, 2001; Langley, 2011) and others just some "intelligent parts" (Bishop, 2006; Mohri, Rostamizadeh, & Talwalkar, 2012)), Optimization (S., Nowozin, & Wright, 2012), Data Mining (Dean, 2014; Perner, 2011), Generalization (Barratt & Sharma, 2018; Kar, K, Jain, & Karnick, 2013; Qi, Silvestrov, & Nazir, 2016) and Statistics (Pavlyshenko, 2016; Suvrit, 2016; von Luxburg & Schoelkopf, 2008).

In 1959 Arthur Samuel, an IBM engineer, published a paper with "machine learning" in the title, being the very first time that phrase appeared in print (Brooks, 2022; Samuel, 1959). Samuel investigated two machine learning procedures using the game of checkers. His work is part of what the site "A history for machine learning" state as the pass from "theory to reality", where the modern machine learning starts in 2006 with the "reboot" of neural net research as "deep learning" (Google, 2017). Since then, ML and Artificial Intelligence (AI) application-related fields (Figure 2.1), image recognition, consumer prediction, natural language processing, and in parallel the increasing access to computational power and growing data production, researchers from around the globe have engrossed the knowledge and applications of this technology.

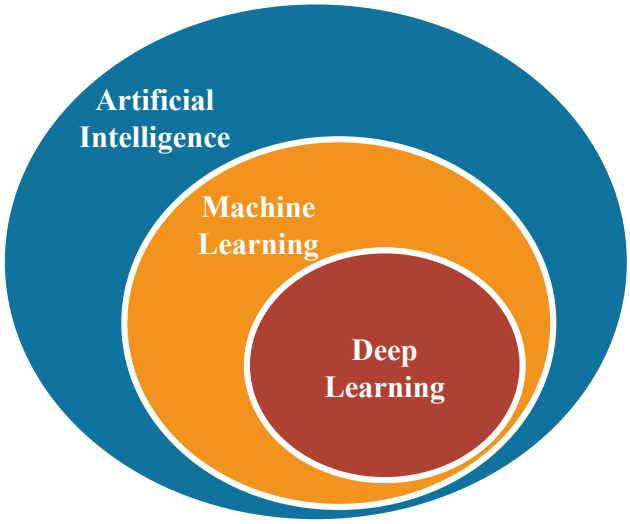

Figure 2.1: Relation between Artificial Intelligence, Machine Learning and Deep Learning.

ML grew out of the quest of AI to get machines that can learn from data (see Figure 2.2). Researchers used symbolic methods and what was then termed "neural networks"; these were mostly perceptrons and models that are variations of statistical generalized linear models (Sarle, 1994), as well as using probabilistic reasoning. Nevertheless, a machine learning system is trained rather than explicitly programmed. It's presented with many examples relevant to a task, and it finds statistical structure in these examples that eventually allows the system to come up with rules for automating the task. For instance, if you wished to automate the task of tagging your vacation pictures, you could present a machine learning system with many examples of pictures already tagged by humans, and the system would learn statistical rules for associating specific pictures to specific tags (Chollet, 2021).

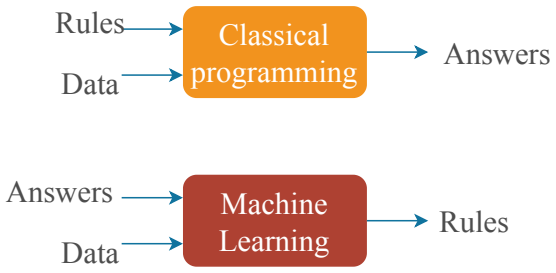

Figure 2.2: Schemes show how Machine Learning changes how a computer does helpful work. Source: adapted from Chollet (2021).

2.1.2 Importance and relevance today

As with any human technological advancement, it relates to the quest of problem-solving, as machine learning appears on logic and game theory problems and fast moves to machine automation tasks. Thereby, it's technology has proved to be widely applicable in fields like physics, chemistry, biology, biochemistry, medicine, astronomy, sociology (human behaviour), electronic commerce, robotics, among others. This fits a growing population's everyday activities in conjunction with advanced telecommunications and more than capable devices.

Its application to problem-solving it's the core behind ML, that is, the learner could be capable of generalizing from its experience (Bishop, 2006; Mohri et al., 2012). The training examples come from some generally unknown probability distribution (considered representative of the space occurrences). The learner has to build a general model about this space to produce sufficiently accurate predictions in new cases. However, the hypothesis's complexity (candidate model that approximates a target function for mapping examples of inputs to outputs (Russell & Norvig, 2021)) gives the best performance when computing (in polynomial time) those predictions in the context of generalization, which should match the complexity of function underlying the data. If hypothesis is less complex than the function, model has under-fitting the data. If complexity of the model is increased in response, then the training error decreases. Nevertheless, if the hypothesis is too complex, then the model is subject to over-fitting, and its generalization will be poorer (Alpaydin, 2010).

Applied ML techniques cover tasks related to searching and detecting patterns in any data (space of occurrences), like images, text and sounds. With access to large amounts of related data provided by technology users (common users, researchers, companies, among other), and through the application of this generalization and optimization techniques, the importance and relevance today seem to be related to the capability of these algorithms to "understand" unseeing patterns in data that a few decades ago there were no means to achieve.

Modern-day machine learning has two objectives: one is to classify data based on models which have been developed, the other purpose is to make predictions for future outcomes based on these models. For example, a hypothetical algorithm specific to classifying data may use computer vision of moles coupled with supervised

learning (see Section 2.1.3) to train it to classify the cancerous moles. Likewise, a machine learning algorithm for stock trading might recommend the trader of future potential predictions.

Chollet (2021) establishes that machine learning discovers rules for executing a data processing task, giving examples of what is expected. So, to do machine learning, it is needed three things:

- **Input data points**. For instance, if the task involves speech recognition, these data points could be sound files of people speaking. On the other hand, if the task is image tagging, it could be pictures.
- **Examples of the expected output**. In a speech recognition task, these could be human-generated transcripts of sound files. In an image task, expected outputs could be tags such as “dog”, “cat”, or another.
- **A way to measure whether the algorithm is doing a good job**. It is necessary to determine the distance between the algorithm’s current output and the expected output. In addition, the measurement is used as a feedback signal to adjust how the algorithm works. This adjustment step is what is called learning.

This capability opens questions about technology reliability due to its technical nature, compared with a ”black box” that takes input data, transforms it, and then delivers results. However, the outstanding quality of these results made a great case to adopt this technology. Nevertheless, the growing adoption generates externalities, not necessarily good ones.

In 2018, Kate Crawford and Vladan Joler’s visual map and essay titled “Anatomy of an AI system” (Crawford & Joler, 2018), demonstrated the impact of an Artificial Intelligence (AI) device on a global scale in terms of human labour, data and resources that are required during its lifespan; from manufacture to disposal, using Amazon’s Echo as an example. It shows where contemporary technology is deeply rooted in the exploitation of human bodies and proposes a visual picture of AI’s impact on the environment and human rights. That essay reveals an ethical dilemma to anyone who works in this field because, on the one hand, ML and AI can play a role in helping to mitigate Climate Change, and on the other hand, AI is itself a significant emitter of carbon (Dhar, 2020). The imposed dichotomy requires a reasonable proxy to containt

any problem.

2.1.3 Featured approaches and algorithms

Machine Learning and Artificial Intelligence are more comprehensive concepts that englobe many techniques. However, these techniques have to be coupled with strategies that allow machines to learn. The main approaches used to learn from data are primarily supervised, unsupervised, and reinforcement learning.

- **Supervised learning**: refers to algorithms that build a mathematical model of a set of data that contains both the inputs and the desired outputs (Russell, Russell, Norvig, & Davis, 2010). Most used algorithms in machine learning are focused on this approach like artificial neural networks, back-propagation, boosting (meta-algorithm), naive Bayes and maximum entropy classifiers, nearest neighbours, support vector machines, among others.
- **Unsupervised learning**: refers to algorithms that learn patterns from untagged data, like grouping or clustering of data points. Exhibit self-organization that captures patterns as probability densities or combinations of neural feature preferences.
- **Semi-supervised learning**: semi-supervised learning fell between unsupervised learning (without any labelled training data) and supervised learning (with completely labelled training data). Some of the training examples are missing training labels, yet many machine learning researchers have found that unlabeled data can produce a considerable improvement in learning accuracy when used in conjunction with a small amount of labelled data. Most of the methods used are related to generative models, low-density separation, and Laplacian regularization.
- **Reinforcement learning**: is an area concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Alternatively, is the problem faced by an agent that learns behaviour through trial-and-error interactions with a dynamic environment. That environment is typically stated in the form of a Markov decision process (MDP) because many reinforcement learning algorithms for this context use dynamic

programming techniques (Kaelbling, Littman, Moore, et al., 1996). Some of the algorithms used are based on associative, deep, inverse, safe, partially supervised reinforcement learning.

2.2 Overview neural networks

Artificial neural networks (ANNs), usually simply called neural networks (NNs), are computing systems inspired by the biological neural networks that constitute animal brains. McCulloch and Pitts (1943) introduced the subject by creating a computational model for neural networks.

2.2.1 A brain inspired analogy

As the name suggests, neural networks are inspired by the brain's computation mechanism, which consists of calculation units called neurons. While the connections between artificial neural networks and the brain are, in fact, somewhat slim, it repeats the analogy here for completeness. In the analogy, a neuron is a computational unit with scalar inputs and outputs (Goldberg, 2017).

Each neuron has a weight associated and multiplies each input by this weight. Then sums[1] and apply a non-linear function to the result and pass it to its output. Figure 2.3 shows such a neuron.

2.2.2 How does a neural network work?

Usually, the neurons are connected, forming a network: the output of a neuron may feed into the inputs of one or more neurons. Indeed, a typical feed-forward neural network may be drawn as in Figure 2.4. Each circle is a neuron, with incoming arrows being the neuron's inputs and outgoing arrows being the neuron's outputs. Each arrow carries a weight, reflecting its importance (see next). Neurons are arranged in layers, reflecting the flow of information. The sigmoid shape inside the neurons in the middle layers represents a non-linear function (e.g., the logistic function $\frac{1}{(1+e^{-x})}$; about this topic please see Section 2.2.4) that is applied to the neuron's value before passing

[1] While summing is the most common operation, other functions, such as max, are also possible.

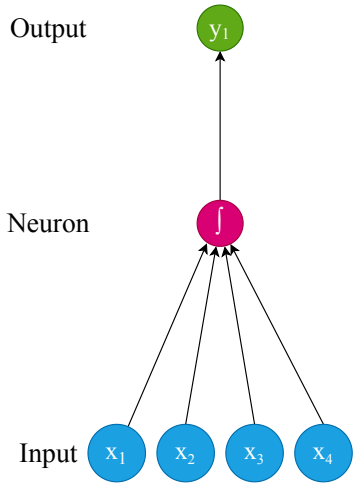

Figure 2.3: A single neuron with four inputs. Source: adapted from Goldberg (2017)

it to the output. In the figure, each neuron is connected to all the neurons in the next layer, being a fully connected layer or an affine layer. It is in the input layer, where the neurons are fed with training observations. Next, the middle or hidden layers perform most of the computations required by the network. Lastly, the output layer predicts the final output extracted from the previous two layers.

In a general form for a feed-forward network, deep feed-forward network or multilayer perceptron, that specific goal is to approximate some function f, are the quintessential deep learning models (Goodfellow, Bengio, & Courville, 2016). It can say that a k-layer neural network is a mathematical function f, which is a composition of multivariate functions: $f_1, f_2, \ldots, f_k$, and g, defined as:

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^p, \tag{2.1}$$

$$f = g \circ f_k \circ \ldots f_2 \circ f_1, \tag{2.2}$$

where, n is the dimension of the input x; p is the dimension of the output y; g is the output function (it can take various forms depending on the output variable); each function f_i is itself a composed multivariate function as:

$$f_i : \mathbb{R}^n \rightarrow \mathbb{R}^p, \tag{2.3}$$

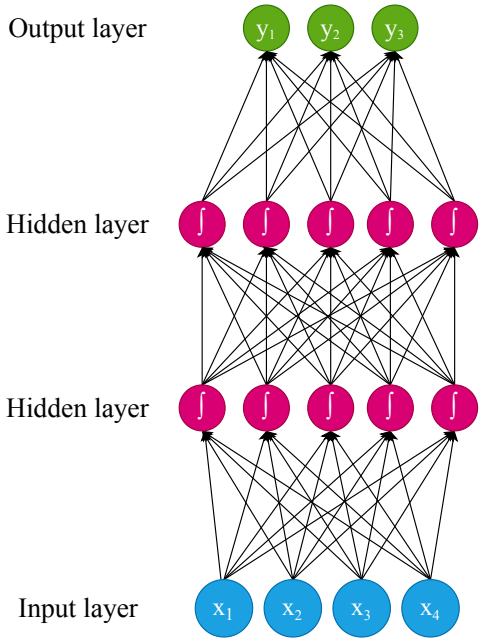

Figure 2.4: Feed-forward neural network with two hidden layers. Source: adapted from Goldberg (2017)

$$f_i(x) = a(\mathbf{w}_i\mathbf{x} + b_i). \tag{2.4}$$

Functions f_i are an intermediary function, where $wx + b$, that is a linear combination of the input x with its coefficients w, plus a bias b, and, a is called activation function.

Developing the global function by introducing the weights w and bias b in Equation 2.2, its gets:

$$\begin{aligned} f_i(x) &= g \circ f_k \circ \dots f_2 \circ f_1(x), \\ &= g(a(\dots a(\mathbf{w}_2 a(\mathbf{w}_1 x + b_1) + b_2) \cdots + b_K)). \end{aligned} \tag{2.5}$$

For the dimension of the domains and the codomains, each function f_i maps its input in $\mathbb{R}^{n_{i-1}}$ (of dimension n_{i-1}) into the codomain $\mathbb{R}^{n_i}$ (of dimension n_i), and the dimension of $\mathbb{R}^{n_i}$ can be chosen by the user (it is one of the hyperparameters of neural networks):

$$f_i : \mathbb{R}^{n_{i-1}} \rightarrow \mathbb{R}^{n_i}. \tag{2.6}$$

There is a particular case: for $i = 1$, the input of the function f_1 is the global input x. So the dimension is n. Considering its matrix form, all the complexity linked to the dimensions occurs in the linear equation $\mathbf{W}_i\mathbf{x} + b_i$, were:

- The dimension of $\mathbf{x}$ is n_{i-1}.
- $\mathbf{W}$ is a matrix with dimension $n_i \times n_{i-1}$.
- The dimension of b_i is n_i.

All the intermediary functions f_i are in a nested composition function where the function g can take several forms depending on the network objective. Some of this forms can be:

$$\begin{aligned}
\text{Regression: } g(x) &= \mathbf{W}_{k+1}\mathbf{x} + \mathbf{b}_{k+1}, \\
\text{Binary classification: } g(x) &= \frac{1}{1 + \exp(-(\mathbf{W}_{k+1}\mathbf{x} + \mathbf{b}_{k+1}))}, \\
\text{Multiclass classification: } g(x) &= \frac{e^{\mathbf{W}_{k+1}\mathbf{x}+\mathbf{b}_{k+1}}}{\sum_{j=1}^{C} e^{\mathbf{W}_j\mathbf{x}+\mathbf{b}_j}}
\end{aligned} \tag{2.7}$$

A generalised schema for how a neural network works is in Figure 2.5.

2.2.3 Neural network as a complex function

Following the idea proposed by McCulloch and Pitts (1943), each neuron, node or unit (all are synonymous in this context) within a network has its role. Data is generally passed as input to the first neuron layer, for instance, a $N \times N$ pixels image, where each pixel is fed as input to each neuron of the first layer. The same will work for text strings where a span of text (a token) is passed to the first neuron layer. Neurons of one layer are connected to the following layers and are assigned a weight. Then, the inputs $(x_1, x_2, \ldots, x_i)$ are multiplied by their corresponding weights, and the sum is sent to the neurons in a posterior hidden layer. Each of these neurons is associated with a numerical value called the "bias", further added to the input sum.

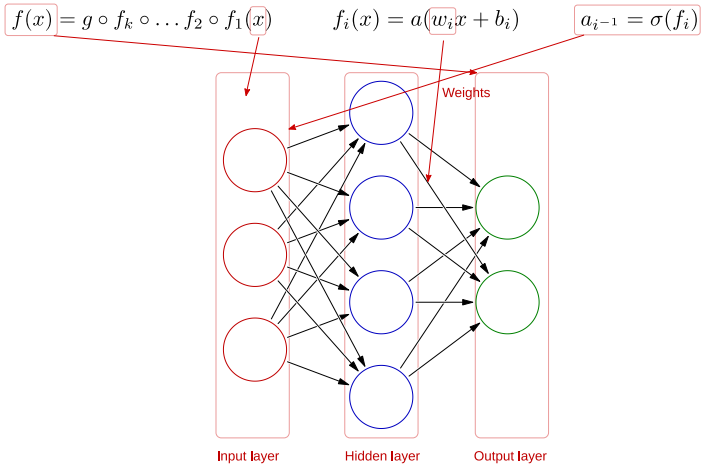

Figure 2.5: The input layer of neurons or nodes represents the input x and the number of neurons is the dimension of the input x. The output layer represents the output y and the number of the neurons depends on the nature of the target variable. For regression and binary classification, it is only 1. For multi-class classification, the number of neurons is the number of classes. A hidden layer represents the result of intermediary functions f_i. The links or synapses represent the values of the weights w_i. So all the links together between two layers represent a matrix. The number of links is $n_i \times n_{i-1}$, as the dimension of the matrix $\mathbf{W}_i$.

This value is then passed through a threshold function called the "activation function", which determines whether the particular neuron will get activated or not. Finally, the activated neuron transmits data to neurons of the next layer. This data is propagated through the network, and the neuron with the highest value determines the output. The most straightforward neural network is called a perceptron, which is simply a linear model. Goldberg (2017) define a perceptron as follow:

$$\begin{aligned} NN_{\text{perceptron}}(\mathbf{x}) &= \mathbf{xW} + \mathbf{b} \\ \mathbf{x} \in \mathbb{R}^{d_{in}}, \mathbf{W} \in \mathbb{R}^{d_{in} \times d_{out}}, \mathbf{b} &\in \mathbb{R}^{d_{out}}, \end{aligned} \tag{2.8}$$

where $\mathbf{W}$ is the wight matrix and $\mathbf{b}$ is the bias term[2]. As layers can be added at will depending on the use objective, the network could have multilayers, as in Figure 2.4

[2]A bias term can be added to a layer by adding an additional neuron with no incoming connections and whose value is always 1.

that have two-hidden layers, taking the perceptron the following form:

$$\begin{aligned} NN(\mathbf{x}) &= \left(g^2\left(g^1\left(\mathbf{x}\mathbf{W^1}+\mathbf{b^1}\right)\mathbf{W^2}+\mathbf{b^2}\right)\right)\mathbf{W^3} \\ \mathbf{x} &\in \mathbb{R}^{d_{in}}, \mathbf{W^1} \in \mathbb{R}^{d_{in}\times d_1}, \mathbf{W^2} \in \mathbb{R}^{d_1\times d_2}, \\ \mathbf{b^1} &\in \mathbb{R}^{d_1}, \mathbf{b^2} \in \mathbb{R}^{d_2}, \mathbf{W^3} \in \mathbb{R}^{d_2\times d_{out}}, \end{aligned} \tag{2.9}$$

and Russell and Norvig (2021) proposed that a_j denote the output of a neuron j and let $w_{i,j}$, be the weight attached to the link from neuron i to neuron j:

$$a_j = g_j\left(\sum_i (w_{i,j} \times x_i) + Bias\right), \tag{2.10}$$

where g_j is a nonlinear activation function associated with unit j. The vectorized form of the above equation looks like this:

$$a_j = g_j(\mathbf{w}^\top \mathbf{x}), \tag{2.11}$$

where $\mathbf{w}$ is the vector of weights leading into unit j and $\mathbf{x}$ is the vector of inputs to unit j.

2.2.4 Activation functions

The activation function can take various forms such as sigmoid, tanh, Relu, among others.

$$\sigma(z) = \frac{1}{1+e^{-z}}, \tag{2.12}$$

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \tag{2.13}$$

$$\text{ReLU}(z) = \max\{0, z\}, \tag{2.14}$$

$$\text{softplus}(z) = log(1+e^z). \tag{2.15}$$

2.3 Natural Language Processing (NLP)

2.3.1 Common frameworks used in NLP

Several Deep Learning frameworks have been developed in the last few years. In addition, various frameworks and libraries have also been used to expedite the work

with good results. Through their use, the training process has become more accessible. Table 2.1 lists the most utilized frameworks and libraries. However, the development scope for those frameworks can differ from each other. For example, Keras, TensorFlow and Torch are developed with research and production scope. On the other hand, spaCy is developed only with production in mind. Many of them can be used to process image and text data (TensorFlow, Torch, MXNet, among others) being TensorFlow the most used due to its versatility (Alzubaidi et al., 2021).

Table 2.1: List of the most common frameworks and libraries used in this context. Source: adapted from Alzubaidi et al. (2021).

Framework	License	Core language	Year of release	Homepages
TensorFlow	Apache 2.0	C++ & Python	2015	https://www.tensorflow.org/
Keras	MIT	Python	2015	https://keras.io/
Caffe	BSD	C++	2015	http://caffe.berkeleyvision.org/
MatConvNet	Oxford	MATLAB	2014	http://www.vlfeat.org/matconvnet/
MXNet	Apache 2.0	C++	2015	https://github.com/dmlc/mxnet
CNTK	MIT	C++	2016	https://github.com/Microsoft/CNTK
Theano	BSD	Python	2008	http://deeplearning.net/software/theano/
Torch	BSD	C & Lua	2002	http://torch.ch/
DL4j	Apache 2.0	Java	2014	https://deeplearning4j.org/
Gluon	AWS Microsoft	C++	2017	https://github.com/gluon-api/gluon-api/
OpenDeep	MIT	Python	2017	http://www.opendeep.org/
spaCy	MIT	Python & Cython	2016	http://spacy.io/

2.3.2 Bag-of-Words

Goldberg (2017) states that a general feature extraction procedure for sentences and documents is the bag-of-words approach (BoW). In this approach, we look at words histogram within the text, i.e., considering each word count as a feature.

The BoW is a representation of text that describes the occurrence of words within a document, and it involves two things:

- A vocabulary of known words, and

- a measure of the presence of known words.

It is called a "bag" of words because information about the document's order or structure does not get it. The only concern to the model is whether known words occur in the document, not where. Nevertheless, it has some shortcomings, such as:

- **Vocabulary**: The vocabulary requires careful design to manage the size, which impacts the sparsity of the document representations.
- **Sparsity**: Sparse representations are harder to model for computational reasons (space and time complexity) and information reasons. The challenge is that the models harness little information in an ample representative space.
- **Meaning**: Discarding word order ignores the context and, in turn, the meaning of words in the document (semantics). Context and meaning can offer a lot to the model that, if modelled, could tell the difference between the identical words differently arranged ("this is interesting" vs "is this interesting"), synonyms ("old bike" vs "used bike"), and much more.

An approach that can be considered as a BoW extraction feature is the one-hot and dense vector representations, where the input vector can be decomposed into an average of $|D|$ vectors, each corresponding to a particular document position i:

$$x = \frac{1}{|D|} \sum_{i=1}^{|D|} x^{D_{[i]}}, \tag{2.16}$$

here, $D_{[i]}$ is the bigram at document position i, and each vector $x^{D_{[i]}} \in \mathbb{R}$ is one-hot vector, in which all entries are zero except the single entry corresponding to the letter bigram $D_{[i]}$ which is 1. This resulting vector x is commonly referred to as an averaged bag of bigrams or, more generally, a bag of words (Goldberg, 2017).

2.3.3 Word2Vec

Developed by Tomaš Mikolov and colleagues (Mikolov, Chen, Corrado, Dean, et al., 2013; Mikolov, Sutskever, et al., 2013), the `Word2Vec` algorithm is a widely popular algorithm that generates word embeddings. In contrast to the so-called count-based methods (Word-context Matrices, Similarity Measures, Word-context Weighting and

PMI and Dimensionality Reduction through Matrix Factorization), the neural network community advocates the use of distributed representations of word meanings. In a distributed representation, each word is associated with a vector in $\mathbb{R}^d$, where the meaning of the word is captured in the different dimensions of the vector. While each dimension corresponds to a specific context the word occurs in, the dimensions in the distributed representation are not interpretable. Specific dimensions do not necessarily correspond to specific concepts. The distributed nature of the representation means that a given aspect of meaning may be captured by (distributed over) a combination of many dimensions and that a given dimension may contribute to capturing several aspects of meaning (Goldberg, 2017).

`Word2Vec` also starts with a neural language model and modifies it to produce faster results, being not a single algorithm but rather a software package implementing two different context representations (CBOW and Skip-Gram) and two different optimization objectives (Negative-Sampling and Hierarchical Softmax). The Negative-Sampling objective (NS) variant of `Word2Vec` works by training the network to distinguish "good" word-context pairs from "bad" ones. However, `Word2Vec` replaces the margin-based ranking objective with a probabilistic one.

Consider a set D of correct word-context pairs and a set $\bar{D}$ of incorrect word-context pairs. The algorithm aims to estimate the probability $P(D = 1|w, c)$ that the word-context pair (w, c) came from the correct set D. This probability should be high (close to 1) for pairs from D and low (close to 0) for pairs from $\bar{D}$. The probability constraint dictates that $P(D = 1|w, c) = 1 - P(D = 0|w, c)$. The probability function is modelled as a sigmoid over the score $s(w, c)$:

$$P(D = 1|w, c) = \frac{1}{1 + e^{-s(w,c)}}. \tag{2.17}$$

The corpus-wide objective of the algorithm is to maximize the log-likelihood of the data $D \cup \bar{D}$:

$$\mathcal{L}(\Theta; D, \bar{D}) = \sum_{(w,c)\in D} \log P(D = 1|w, c) + \sum_{(w,c)\in \bar{D}} \log P(D = 0|w, c). \tag{2.18}$$

The positive examples D are generated from a corpus. The negative examples

$\bar{D}$ can be generated in many ways. In `Word2Vec`, they are generated by the following process: for each good pair $(w, c) \in D$, sample k words $w_{1:k}$ according to a specified distribution and add each pair (w_i, c) as a negative example to $\bar{D}$. This results in the negative samples data $\bar{D}$ being k times larger than D. The number of negative samples k is a hyperparameter of the algorithm that controls the trade-off between computational efficiency and training quality (Goldberg, 2017).

The negative words w can be sampled according to their corpus-based frequency: $\frac{\#(w)}{\sum_{w'} \#(w')}$, or, as implemented in the original `Word2Vec`, according to a smoothed version in which the counts are raised to the power of $\frac{3}{4}$ before normalizing: $\frac{\#(w)^{0.75}}{\sum_{w'} \#(w')^{0.75}}$. Here, $\#(w)$ denotes the frequency count of word w in the corpus. This smoothed version gives more relative weight to less frequent words, reducing the dominance of highly frequent words and resulting in better word similarities in practice. The exponent of 0.75 was chosen empirically and represents a balance between uniform sampling and frequency-based sampling (Goldberg, 2017).

2.3.4 Transformers

The Transformer architecture makes it highly efficient to parallelize Machine Learning (ML) training. Massive parallelization thus makes it feasible to train models on large amounts of data in a relatively short period. It was proposed by Vaswani et al. (2017) as an effort to solve the constraints that sequential computation has given by the sequence nature of the algorithm that precludes the parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples. Transformer architecture mechanism (Figure 2.6) avoids recurrence and instead relies entirely on an attention mechanism[3] to draw global dependencies between input and output.

Transduction models have an encoder-decoder structure, where encoder maps an input sequence of symbol representations $(x_1, \ldots, x_n)$ to a sequence of continuous representations $z = (z_1, \ldots, z_n)$. Given z, the decoder then generates an output se-

[3] An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values. The weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

quence $(y_1, \ldots, y_m)$ of symbols one element at a time. At each step, the model is auto-regressive (Graves, 2013), consuming the previously generated symbols as additional input when generating the next.

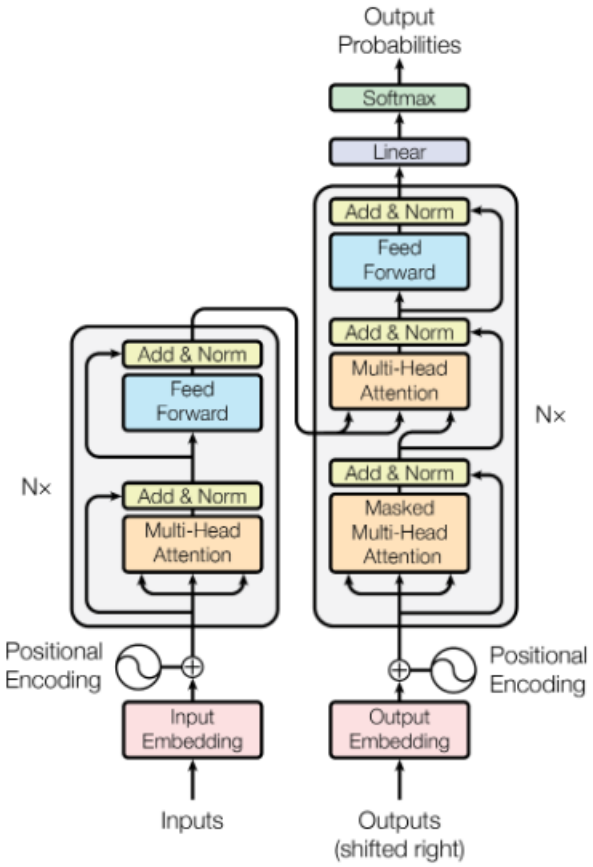

Figure 2.6: The Transformer model architecture. Source: Vaswani et al. (2017)

Transformers use dot product attention (see left-hand in Figure 2.7), which follows a general attention procedure that can be defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.19}$$

where $Q \in \mathbb{R}^{n \times d_k}$ is the query matrix, $K \in \mathbb{R}^{m \times d_k}$ is the key matrix, $V \in \mathbb{R}^{m \times d_v}$ is the value matrix, n is the query sequence length, m is the source sequence length, d_k is the dimension of keys and queries, and d_v is the dimension of values. The scaling

factor $\sqrt{d_k}$ prevents the dot products from becoming too large, which could push the softmax function into regions with extremely small gradients.

In practice, the attention function is computed simultaneously over a set of queries packed together in a matrix Q. In this scenario, the keys and values are also represented as matrices K and V, respectively, and the output is also a matrix. The Figure 2.7 shows a schematic of the attention function. Also, a comprehensive analysis is made by Vasilev (2019).

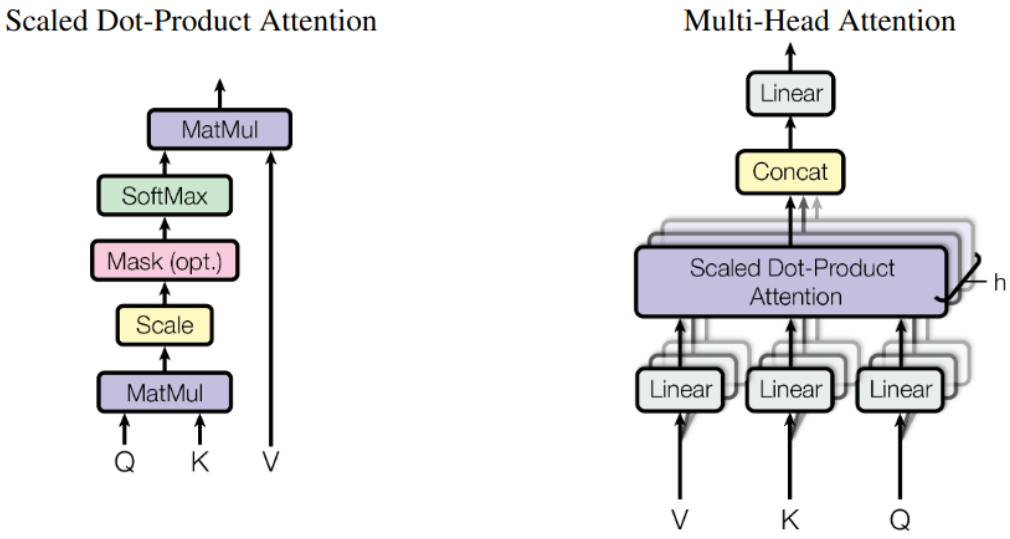

Figure 2.7: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel. The inputs V, K, and Q represent the value, key, and query matrices, respectively. Source: Vaswani et al. (2017)

On right-hand in Figure 2.7, Vaswani et al. (2017) proposed the multi-head attention. Instead of a single attention function with a d_{model}-dimensional keys, queries and values are linearly projected h times to produce h different d_q-,d_k-, and d_v-dimensional projections of these values. Then, it applies separate parallel attention functions (or heads) over the newly created vectors, which yield a single d_v-dimensional output for each head. Finally, it concatenates the head outputs to produce the final attention result. Multihead attention allows each head to attend to different elements of the sequence. At the same time, the model combines the outputs of the heads in a single cohesive representation. More precisely, it can be defined as:

$$\begin{aligned} \text{MultiHead}(Q,K,V) &= \text{Concat}(head_1, head_2, \ldots, head_h)W^O, \\ \text{where} \quad head_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V), \end{aligned} \tag{2.20}$$

where $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, and $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ are learnable projection matrices used to transform the input queries, keys, and values for the i-th head, respectively. The parameter $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ is a learnable output projection matrix that linearly transforms the concatenated head outputs to produce the final attention result. Each head operates independently with its own set of projection matrices, allowing the model to jointly attend to information from different representation subspaces at different positions (Vasilev, 2019).

As is the model architecture presented in Figure 2.6 has two parts: the encoder (the left-hand component in the diagram) and the decoder (the right-hand component in the diagram). The encoder begins with an input sequence of one-hot encoded words, which are transformed into d_{model}-dimensional embedding vectors. The embedding vectors are further multiplied by $\sqrt{d_{model}}$. Later, the positional information is encoded, augmenting each embedding vector. Summarising, the positional vector of length, d_{model}, is added to the embedding vector, and the result is propagated further in the encoder. The rest of the encoder comprises a stack of $N = 6$ identical blocks. Each block has two sublayers: a multi-headed self-attention mechanism and a simple, fully connected feed-forward network which is defined as:

$$\text{FFN}(x) = \text{ReLU}(W_1 x + b_1)W_2 + b_2, \tag{2.21}$$

where $x \in \mathbb{R}^{d_{model}}$ is the input vector, $W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$ and $W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$ are weight matrices, $b_1 \in \mathbb{R}^{d_{ff}}$ and $b_2 \in \mathbb{R}^{d_{model}}$ are bias vectors, d_{model} is the model dimension, and d_{ff} is the inner dimension of the feed-forward network (typically $d_{ff} = 4 \times d_{model}$). The ReLU activation function is applied element-wise to the intermediate representation.

The network is applied to each sequence element x separately. It uses the same set of parameters (W_1, W_2, b_1, and b_2) across different positions but different parameters across the different encoder blocks. Each sublayer (multi-headed attention and

feed-forward network) has a residual connection around itself and ends with normalization over the sum of that connection, its output, and the residual connection.

On the other hand, the decoder is similar to the encoder. The input at step t is the decoder's predicted output word at step $t-1$. The input word uses the same embedding vectors and positional encoding as the encoder. The decoder continues with a stack of $N = 6$ identical blocks, somewhat similar to the encoder blocks. Each block consists of three sublayers, and each sublayer employs residual connections and normalization. These sublayers are masked multi-headed attention, a common multi-headed attention mechanism and a feed-forward network. The decoder ends with a fully connected layer, followed by a softmax, which produces the most probable next word of the sentence.

2.3.5 Transformer language models

Bidirectional Encoder Representations from Transformers (BERT)

It is an ML model for natural language processing. It was developed in 2018 by researchers at Google AI Language (Devlin, Chang, Lee, Toutanova, et al., 2018) and serves as a swiss army knife solution to the most common language tasks, such as sentiment analysis and named entity recognition (Muller, 2022). It has two elements and is based on the encoder part of the Transformer architecture:

- Encoder representations. This model uses only the output of the multilayer encoder part of the transformer architecture.
- Bidirectional. The encoder has an inherent bidirectional nature.

Transformer-XL

This model language is an improvement over the vanilla transformer, where XL stands for extra long. It was introduced by Dai et al. (2019) and looked to improve the standard transformer that depends on the segment length. However, it is based only on the decoder part of the Transformer architecture. Therefore, the transformer-XL decoder is not the same as the decoder in the complete encoder-decoder transformer because it does not have access to the encoder state as the standard encoder does. This gives a unidirectional nature to the language model, being an autoregressive model.

This model introduces a recurrence relationship in the transformer model. For that, during the training, the model caches its state for the current segment, and when it processes the next segment, it has access to that cached (but fixed) value. Also, it uses a relative positional encoding that works similarly to the vanilla positional encoder. In addition, it dynamically passes the relative distance, making it possible for the query vector to distinguish between positions of the x vector.

2.4 Deep Learning in NLP

There are many deep learning networks, among the most commonly used are recursive and recurrent neural networks (RvNN & RNN) and convolutional neural networks (CNN) (Alzubaidi et al., 2021). RvNN and RNN are briefly explained in this section, with a more detailed view of a variation of these neural networks, like long-short term memory networks (LSTM) and Stack-LSTM or S-LSTM.

2.4.1 Recursive and recurrent Neural Networks (RvNN & RNN)

The RvNN architecture is generated for processing objects, which have randomly shaped structures like graphs or trees (Alzubaidi et al., 2021). The approach of RvNNs in a discriminative parsing architecture aims to learn a function $f : X \rightarrow Y$, where Y is the set of all possible binary parse trees. An input x consists of two parts: (i) A set of activation vectors $a_1, \ldots, a_{Nsegs}$, which represent input elements such as image segments or words of a sentence. (ii) A symmetric adjacency matrix A, where $A(i, j) =$ 1, if segment i neighbors j. This matrix defines which elements can be merged. For sentences, this matrix has a special form with 1$\square$$s$ only on the first diagonal below and above the main diagonal. RvNN has been proven highly effective in the NLP context (Socher, Lin, Ng, Manning, et al., 2011).

On the other hand, RNN is mainly applied in speech processing and NLP contexts. Unlike conventional networks, RNN uses sequential data in the network. Since the embedded structure in the data sequence delivers valuable information, this feature is fundamental to a range of applications. For instance, it is important to understand the context of the sentence to determine the meaning of a specific word. Thus, it is possible to consider the RNN as a unit of short-term memory, where x repre-

sents the input layer, y is the output layer, and s represents the state (hidden) layer. Pascanu, Gulcehre, Cho, and Bengio (2014) introduced three different types of deep RNN techniques, namely "Hidden-to-Hidden", "Hidden-to-Output", and "Input-to-Hidden".

Pascanu et al. (2014) stablished that RNN simulates a discrete-time dynamical system that has an input x_t, an output y_t and a hidden state h_t, where subscript t represents time. The dynamical system is defined by:

$$h_t = f_h(x_t, h_{t-1}), \tag{2.22}$$

$$y_t = f_o(h_t), \tag{2.23}$$

where f_h and f_o are a state transition function and an output function, respectively. Each function is parameterized by a set of parameters; θ_h and θ_o.

Given a set of N training sequences $D = \left\{\left((x_1^{(n)}, y_1^{(n)}), \ldots, (x_{T_n}^{(n)}, y_{T_n}^{(n)})\right)\right\}_{n=1}^{N}$, the parameters of an RNN can be estimated by minimising the following cost function:

$$J(\theta) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} d\big(y_t^{(n)}, f_o(h_t^{(n)})\big), \tag{2.24}$$

where $h_t^{(n)} = f_h\big(x_t^{(n)}, h_{t-1}^{(n)}\big)$, $h_0^{(n)} = 0$ and $d(a,b)$ is a predefined divergence measure between a and b, such as Euclidean distance or cross-entropy.

A conventional RNN is constructed by defining the transition function and the output function as:

$$h_t = f_h(x_t, h_{t-1}) = \phi_h\big(W^\top h_{t-1} + U^\top x_t\big), \tag{2.25}$$

$$y_t = f_o(h_t) = \phi_o\big(V^\top h_t\big), \tag{2.26}$$

where W, U and V are respectively the transition, input and output matrices, and ϕ_h and ϕ_o are element-wise nonlinear functions. It is usual to use a saturating nonlinear function such as a logistic sigmoid function or a hyperbolic tangent function for

ϕ_h. Although RNNs can, in principle, model long-range dependencies, training them is difficult in practice since the repeated application of a squashing nonlinearity at each step results in an exponential decay in the error signal through time. Figure 2.8 shows a scheme with a conventional RNN.

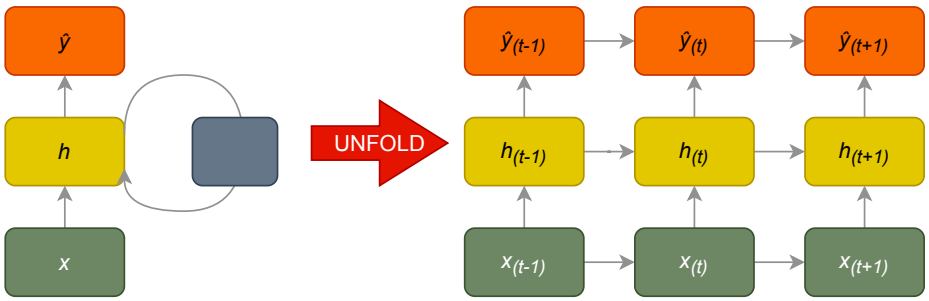

Figure 2.8: Scheme of an RNN network where is shown the unfolded three layers, input (x_t), hidden (h_t) and output (y_t). RNNs process sequences sequentially, maintaining information in hidden states $h_{(t)}$ while sharing weight matrices across time steps. Training suffers from vanishing and exploding gradient problems, limiting long-term dependency learning. Source: adapted from Becker et al. (2020)

The main problem with conventional backpropagation through time (BPTT) or real-time recurrent learning (RTRL) is that error signals flowing backwards in time tend to (1) blow up or (2) vanish; the temporal evolution of the backpropagated error exponentially depends on the size of the weights. The first case may lead to oscillating weights; in the second, learning to bridge long time lags takes a prohibitive amount of time or does not work at all (Hochreiter & Schmidhuber, 1997).

With that in mind, Long Short-Term Memory models (LSTMs (Hochreiter & Schmidhuber, 1997)) and Gated Recurrent Units (GRUs (Cho et al., 2014)) were developed; evolving to encoder-decoder architectures (bi-LSTM) (Schuster & Paliwal, 1997)

2.4.2 Long-Short Term Memories Neural Networks (LSTM & Stack-LSTM)

Dyer et al. (2015) state that LSTMs are a variant or extension of recurrent neural networks (RNNs) designed to cope with the vanishing gradient problem inherent in RNNs, which was described by Hochreiter and Schmidhuber (1997) and Graves

(2013). As it was shown in Figure 2.8, RNNs read a vector x_t at each time step and compute a new (hidden) state h_t by applying a linear map to the concatenation of the previous time step's state h_{t-1}, taking the input and passing it through a logistic sigmoid non-linearity function.

With LSTMs, there is no need to keep a finite number of states from beforehand as required in, for example, the hidden Markov model (HMM) (Kaelbling et al., 1996). LSTMs provide an extensive range of parameters such as learning rates and input and output biases. Hence, no need for fine adjustments. The complexity of updating each weight is reduced to $O(W)$ with LSTMs, where W is the total number of weights in the network (Hochreiter & Schmidhuber, 1997), similar to BPTT, which is an advantage.

Hochreiter and Schmidhuber (1997) establishes that constructing an architecture that allows for constant error flow through special and self-connected units, extending the embodied constant error carrousel (CEC), which is the central feature of LSTMs, by introducing additional features.

Dyer et al. (2015) establishes that LSTMs have an extra memory "cell" (c_t) constructed as a linear combination of the previous state and signal from the input. LSTM cells process inputs with three multiplicative gates, which control what proportion of the current input to pass (i_t) into the memory cell and what proportion of the last memory cell to "forget" (f_t). The updated value of the memory cell after an input x_t is computed as follows (see Figure 2.9 for a scheme representation of the follow equations):

$$i_t = \sigma\big(W_{ix}x_t + W_{ih}h_{t-1} + W_{ic}c_{t-1} + b_i\big), \tag{2.27}$$

$$f_t = \sigma\big(W_{fx}x_t + W_{fh}h_{t-1} + W_{fc}c_{t-1} + b_f\big), \tag{2.28}$$

$$c_t = f_t \odot c_{t-1} + \tanh\big(W_{cx}x_t + W_{ch}h_{t-1} + b_c\big), \tag{2.29}$$

where σ is the component-wise logistic sigmoid function, and $\odot$ is the component-wise (Hadamard) product. The value h_t of the LSTM at each time step

is controlled by a third gate (o_t) that is applied to the result of the application of non-linearity to the memory cell contents:

$$o_t = \sigma\big(W_{ox}x_t + W_{oh}h_{t-1} + W_{oc}c_t + b_o\big), \tag{2.30}$$

$$h_t = o_t \odot \tanh(c_t). \tag{2.31}$$

To improve the representational capacity of the LSTMs (and RNNs generally), LSTMs can be stacked in "layers" (Pascanu et al., 2014). In these architectures, the input LSTM at higher layers at time t is the value of h_t, computed by the lower layer (and x_t is the input at the lowest layer). Finally, an output is produced at each time step from the h_t value at the top layer:

$$y_t = g(h_t), \tag{2.32}$$

where g is an arbitrary differentiable function.

Conventional LSTMs model sequences in a left-to-right order. The innovation from Dyer et al. (2015), is to augment the LSTM with a "stack pointer". Like in a conventional LSTM, new inputs are always added in the right-most position, but in stack LSTMs, the current location of the stack pointer determines which cell in the LSTM provides c_{t-1} and h_{t-1} when computing the new memory cell contents.

Besides to adding elements to the end of the sequence, the stack LSTM provides a `pop` operation which moves the stack pointer to the previous element (i.e., the previous element that was extended, not necessarily the right-most element). Thus, the LSTM can be understood as a stack implemented so that contents are never overwritten, that is, push always adds a new entry at the end of the list that contains a back-pointer to the previous `top`, and `pop` only updates the stack pointer. This control structure is schematised in Figure 2.10.

By querying the output vector to which the stack pointer points (i.e., the h_{TOP}), a continuous-space "summary" of the contents of the current stack configuration is available. This value can be called the "stack summary", where elements near the `top`

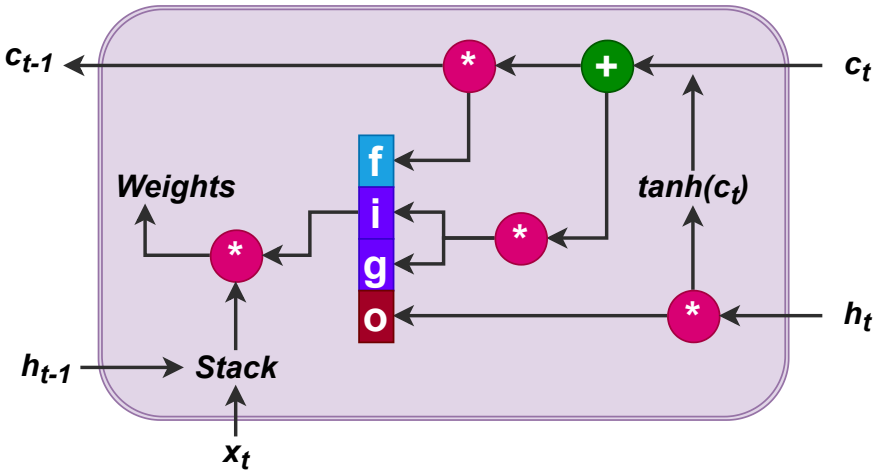

Figure 2.9: Scheme that shows the gradient calculation through backward propagation through time t using the chain rule in a "cell", where, x_t is the input at time t, c_t and c_{t-1} are cell state at time t and $t-1$, respectively. In the same way, h_t and h_{t-1} are the outputs at same times. Asterix (*) represent point-wise multiplication; plus sign (+) represent point-wise addition; f: forget gate (sigmoid); i and g are input gate (i: *sigmoid* & g: *tanh*); o represents output gate (*sigmoid*).

of the stack will influence the representation of the stack. However, the LSTM has the flexibility to learn to extract information from arbitrary points in the stack (Dyer et al., 2015).

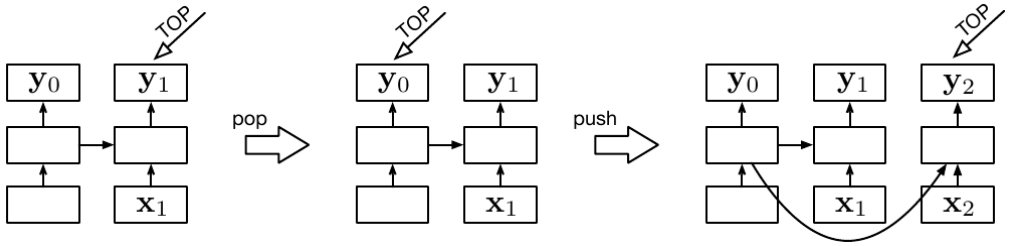

Figure 2.10: A stack LSTM extends a conventional left-to-right LSTM with the addition of a stack pointer (notated as `TOP` in the figure). This figure shows three configurations: a stack with a single element (left), the result of a pop operation to this (middle), and then the result of applying a push operation (right). The boxes in the bottom rows represent stack contents, which are the inputs to the LSTM, the upper rows are the outputs of the LSTM (in this paper, only the output pointed to by `TOP` is ever accessed), and the middle rows are the memory cells (the c_t's and h_t's) and gates. Arrows represent function applications (usually affine transformations followed by a nonlinearity). Source: Dyer et al. (2015)

2.4.3 Common Convolutional Neural Networks (CNNs)

It is a class of deep neural networks that extracts features from images and text as input to perform specific tasks such as image classification, face recognition, semantic image system and natural language processing. A CNN has one or more convolution layers for simple feature extraction, which execute convolution operation (i.e. multiplication of a set of weights with input) while retaining the critical features (spatial and temporal information) without human supervision.

The three primary layers that define the structure of a convolutional neural network are (see Figure 2.11):

1. Convolution layer (see Figure 2.12): represents the first layer of the convolutional network that performs feature extraction by sliding the filter over the data. The output or the convolved feature is the element-wise product of filters in the data and their sum for every sliding action.

   The output layer, also known as the feature map, corresponds to original data like words, punctuation, and dependencies, among others.

   In the case of networks with more convolutional layers, the initial layers are meant for extracting the generic features. At the same time, the complex parts

are removed as the network gets deep.

2. Pooling layer: represents the operation to reduce the dimensions of the feature maps. It is added after the convolutional layer (specifically after a nonlinearity has been applied to the features map output), and operates upon each feature map separately to create a new set of the same number of pooled feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network. Those operation involves sliding a two-dimmensional filter over each channel of feature map and summarising the features lying within the region covered by the filter.

   The pooling layer summarises the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarised features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input image.

   There is at least, three types of pooling layers:

   (a) Maximum or Max Pooling. It is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

   (b) Average Pooling. Computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.

   (c) Global Pooling. Reduces each channel in the feature map to a single value. Thus, an $n_h \times n_w \times n_c$ feature map is reduced to $1 \times 1 \times n_c$ feature map. This is equivalent to using a filter of dimensions $n_h \times n_w$ i.e., the dimensions of the feature map. Further, it can be either global max pooling or global average pooling.

   Pooling helps to make the representation become approximately invariant to small translations of the input. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do

not change.

3. Fully connected layer: represent a simple neural network or a multilayer perceptron (MLP), that was defined in Section 2.2.1.

Figure 2.11: Scheme of a Convolutional Neural Network. Source: adapted from Becker et al. (2020)

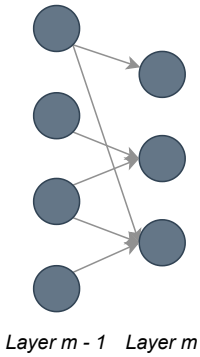

Figure 2.12: Convolutional Neural Network layer scheme. Source: adapted from Becker et al. (2020)

2.4.4 Convolutional Neural Networks (CNN) for sentence classification

Deep learning models have achieved remarkable results in computer vision and speech recognition. Within natural language processing, much of the work with deep learning methods have involved learning word vector representations through neural language models and performing composition over the learned word vectors for classification Kim (2014). Word vectors, wherein words are projected from a sparse, $1-of-V$ encoding (where V is the vocabulary size) onto a lower dimensional vector space via a hidden layer, are feature extractors that encode semantic features of words in their dimensions. For example, in such dense representations, semantically close words are likewise (in euclidean or cosine distance) in the lower dimensional vector space Kim (2014).

Convolutional neural networks (CNN) utilize layers with convolving filters that are applied to local features. Originally invented for computer vision, CNN models have subsequently been shown to be effective for NLP and have achieved excellent results in semantic parsing, search query retrieval, sentence modelling, and other traditional NLP tasks Kim (2014).

The model architecture is a slight variant of the CNN architecture Kim (2014). Let $x_i \in \mathbb{R}^k$ be a k-dimensional word vector corresponding to the i-th word in the sentence. A sentence of length n (padded where necessary) is represented as:

$$x_{1:n} = x_1 \oplus x_2 \oplus \cdots \oplus x_n, \tag{2.33}$$

where $\oplus$ is the concatenation operator. In general, let $x_{i:i+j}$ refer to the concatenation of words $x_i, x_{i+1}, \ldots, x_{i+j}$. A convolution operation involves a filter $w \in \mathbb{R}^{hk}$, which is applied to a window of h words to produce a new feature. For example, a feature c_i is generated from a window of words $x_{i:i+h-1}$ by:

$$c_i = f(w \cdot x_{i:i+h-1} + b), \tag{2.34}$$

where $b \in \mathbb{R}$ is a bias term and f is a non-linear function such as the hyperbolic tangent. This filter is applied to each possible window of words in the sentence $\{x_{1:h}, x_{2:h+1}, \ldots, x_{n-h+1:n}\}$ to produce a feature map:

$$c = [c_1, c_2, \ldots, c_{n-h+1}], \tag{2.35}$$

where $c \in \mathbb{R}^{n-h+1}$. Then a max-over-time pooling operation is applied over the feature map to take the maximum value $\hat{c} = \max(c)$ as the feature corresponding to this particular filter. The idea is to capture the most critical feature with the highest value for each feature map. This pooling scheme naturally deals with variable sentence lengths.

This describes how one feature is extracted from one filter. The model uses multiple filters (with varying window sizes) to obtain multiple features. These features form the penultimate layer and are passed to a fully connected softmax layer whose output

is the probability distribution over labels. In the multichannel architecture, illustrated in Figure 2.13, each filter is applied to both channels, and the results are added to calculate c_i in Equation (2.34). The model is otherwise equivalent to the single-channel architecture (Kim, 2014).

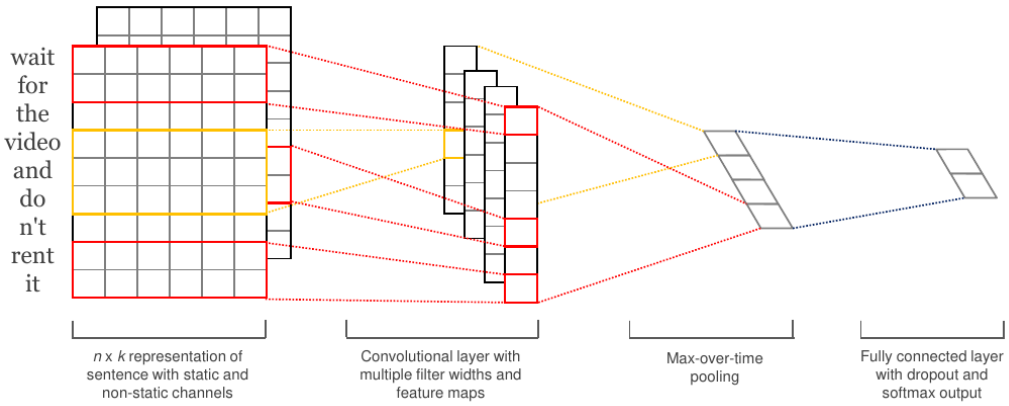

Figure 2.13: Model architecture with two channels for an example sentence. Source: Kim (2014)

2.4.5 Backpropagation in Neural Network Models

Backpropagation is a fundamental algorithm for efficiently computing the gradients needed to update network weights and minimize the loss function during training.

The algorithm operates in two phases: first, activations are computed in a forward pass through the network layers. Then, the gradients of the cost function with respect to each parameter are computed in a backward pass, propagating error signals from the output layer back to the input layer. This method enables the calculation of partial derivatives of the cost function with respect to each weight, which are essential for gradient-based optimization algorithms.

Backpropagation serves as the standard method for training neural networks iteratively. By systematically adjusting the network weights based on the computed gradients, backpropagation enables the network to learn complex patterns from data while minimizing prediction errors, thereby improving the model's accuracy and generalization capability.

2.4.6 Evaluation Metrics in Deep Learning

Evaluation metrics in Deep Learning (DL) tasks play a crucial role in developing optimized models. They are utilized throughout the standard data classification pipeline during both training and testing phases. During training, evaluation metrics serve as optimization criteria to guide algorithm development and hyperparameter selection. During testing, these metrics assess the performance of the trained classifier on unseen data to estimate its generalization capability.

The foundation of most classification metrics lies in the confusion matrix, where True Positives (TP) and True Negatives (TN) represent correctly classified positive and negative instances, respectively, while False Negatives (FN) and False Positives (FP) represent misclassified positive and negative instances, respectively. The most commonly used evaluation metrics are described below.

1. **Accuracy**: Measures the ratio of correct predictions to the total number of samples evaluated.
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.36}$$

2. **Sensitivity or Recall**: Measures the fraction of positive instances that are correctly identified.
$$\text{Sensitivity} = \frac{TP}{TP + FN} \tag{2.37}$$

3. **Specificity**: Measures the fraction of negative instances that are correctly identified.
$$\text{Specificity} = \frac{TN}{FP + TN} \tag{2.38}$$

4. **Precision**: Measures the fraction of predicted positive instances that are actually positive.
$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.39}$$

5. **F1-Score**: Calculates the harmonic mean of precision and recall, providing a balanced measure when both metrics are important.
$$F1_{\text{score}} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.40}$$

6. **Youden's J Statistic**: Also known as the J Score, this metric summarizes the performance of a binary classifier.

$$J_{\text{score}} = \text{Sensitivity} + \text{Specificity} - 1 \tag{2.41}$$

7. **False Positive Rate (FPR)**: Measures the probability of incorrectly classifying a negative instance as positive.

$$\text{FPR} = 1 - \text{Specificity} \tag{2.42}$$

8. **Area Under the ROC Curve (AUC)**: A ranking metric that evaluates classifier performance across all classification thresholds. Unlike threshold-dependent metrics, AUC provides a comprehensive assessment of classifier ranking performance. For binary classification problems, AUC is calculated as:

$$\text{AUC} = \frac{S_p - n_p(n_p+1)/2}{n_p n_n} \tag{2.43}$$

where S_p represents the sum of ranks of all positive samples, n_p is the number of positive samples, and n_n is the number of negative samples. AUC has been validated both empirically and theoretically as a robust metric for model selection and performance evaluation in classification tasks.

2.4.7 Loss Functions in Deep Learning

Section 2.4.4 has presented the convolutional neural network (CNN) architecture. In addition, the final classification process is achieved from the output layer, which represents the last layer of the CNN architecture. Some loss functions are utilized in the output layer to calculate the predicted error created across the training samples in the CNN model. This error reveals the difference between the actual output and the predicted one (Alzubaidi et al., 2021).

The loss function uses two parameters to calculate the error. The CNN estimated output (referred to as the prediction) is the first parameter. The actual output (referred to as the label) is the second parameter. Several types of loss functions are employed in various problem types. The following explains some of the loss function types (Alzubaidi et al., 2021).

a) **Cross-Entropy Loss Function**: This function is commonly employed for measuring the CNN model performance in multi-class classification problems. It is also referred to as the log loss function. Its output is a probability distribution where each $p_i \in (0,1)$.

It is usually employed as a substitution of the Euclidean loss function in multi-class classification problems. The output layer employs the softmax activations to generate the output within a probability distribution. The output class probability is given by:

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}, \tag{2.44}$$

where a_i represents the raw output (logit) from the preceding layer, and N represents the number of classes in the output layer. The mathematical representation of cross-entropy loss function is given by:

$$H(p,y) = -\sum_{i=1}^{N} y_i \log(p_i), \tag{2.45}$$

where y_i is the true label and $i \in [1,N]$.

b) **Euclidean Loss Function**: This function is widely used in regression problems, and its formula is given by:

$$H(p,y) = \frac{1}{N}\sum_{i=1}^{N}(p_i - y_i)^2. \tag{2.46}$$

c) **Hinge Loss Function**: This function is commonly employed in binary classification problems. This relates to maximum-margin-based classification, which is particularly important for Support Vector Machines (SVMs). The hinge loss function encourages the optimizer to maximize the margin between the two classes.

$$H(p,y) = \sum_{i=1}^{N} \max(0, 1 - y_i p_i), \tag{2.47}$$

where $y_i \in \{-1, +1\}$ represents the true class label and p_i is the predicted score (not probability). The margin is implicitly set to 1 in this formulation.

2.5 Neural architectures used in NLP by spaCy

The framework has its own deep learning library called Thinc used in background for different NLP models. For most tasks, spaCy uses a deep neural network based on CNN with a few tweaks in the processing. Specifically, spaCy's named entity recognition (NER) use a neural architecture that constructs and labels segments using a transition-based approach (Lample et al., 2016), that was inspired by shift-reduce parsers from Nivre (2004). Those, by which a Stack-LSTM model (Dyer et al., 2015) pass through .

Transition	Output	Stack	Buffer	Segment
	[]	[]	[Mark, Watney, visited, Mars]	
SHIFT	[]	[Mark]	[Watney, visited, Mars]	
SHIFT	[]	[Mark, Watney]	[visited, Mars]	
REDUCE(PER)	[(Mark Watney)-PER]	[]	[visited, Mars]	(Mark Watney)-PER
OUT	[(Mark Watney)-PER, visited]	[]	[Mars]	
SHIFT	[(Mark Watney)-PER, visited]	[Mars]	[]	
REDUCE(LOC)	[(Mark Watney)-PER, visited, (Mars)-LOC]	[]	[]	(Mars)-LOC

Figure 2.14: Example of a transition sequence for *Mark Watney visited Mars* with the Stack-LSTM model. It starts with all empty stack, all words on buffer and no estities. The transition inventory contains the following transitions: The SHIFT transition moves a word from the buffer to the stack, the OUT transition moves a word from the buffer directly into the output stack while the REDUCE(y) transition pops all items from the top of the stack creating a "chunk", labels this with label y, and pushes a representation of this chunk onto the output stack. The algorithm ends when the stack and buffer are both empty. Source: Lample et al. (2016)

spaCy's framework was called "Embed, Encode, Attend, Predict" by Honnibal (2018), where the solution to the problem of all words looks unique to a computer; i.e., "dog" and "puppy" are just strings of words that can quickly be learned with P(id|'dog'), where it needs to predict the P(id|'puppy'), that can be afforded by embeddings words. Furthermore, words are embedded using a Bloom filter (Bloom, 1970), meaning word hashes are kept as keys in the embedding dictionary instead of the word itself. This maintains a more compact embeddings dictionary, with words

potentially colliding and ending up with the same vector representations (see Figure 2.15).

However, with that approximation, it loses the context. So the solutions pass for learning to "encode" the context into a sentencing matrix by taking a list of word vectors. For this, `spaCy` uses CNNs for encoding. Nevertheless, that sentence matrix gives meanings to individual tokens and no representations of entire sentences.

The above can be solved by learning what to pay "attention", summarising sentences when which parts are more informative given a query and getting problem-specific representations. Nevertheless, the application needs a value for output, not a generic representation, and the framework needs to learn to "predict" target values turning a generic architecture into a specific solution. To do that, `spaCy` uses a multi-layer perceptron for inference.

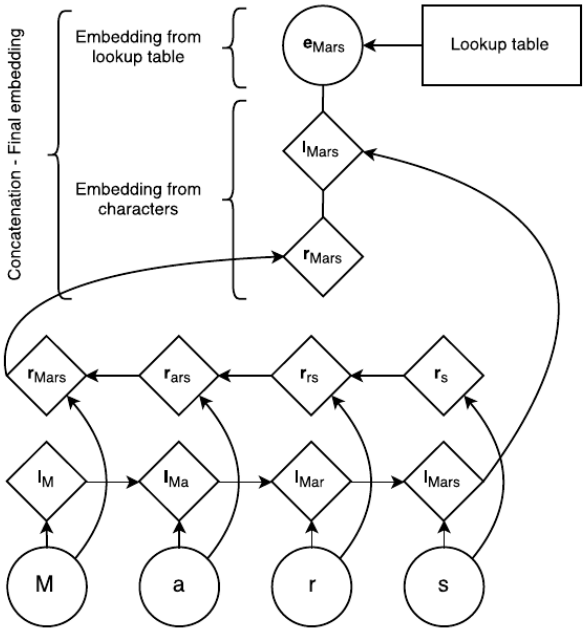

Figure 2.15: The character embeddings of the word "Mars" are given to a bidirectional LSTMs. Then it concatenate their last outputs to an embedding from a lookup table to obtain a representation for this word. Source: Lample et al. (2016)

2.6 Natural language processing on Chilean environmental legal text

2.6.1 NLP in legal domain

As stated in the preceding sections, processing natural language is one of the primary use of Deep Learning, and it has been used in different language domains. The legal text is one of them. There are efforts by many research groups to propose novels solutions in many different languages like Turkish, French, Spanish, Portuguese, and German, among others (Angelidis, Chalkidis, & Koubarakis, 2018; Barriere & Fouret, 2019; Glaser, Waltl, & Matthes, 2018; Mumcuoğlu, Öztürk, Ozaktas, & Koç, 2021; Sierra et al., 2018; Wang, Wu, Lei, & Peng, 2020). At the same time, other solutions to some legal issues like retain opinions from parties (Samarawickrama, de Almeida, Perera, de Silva, & Ratnayaka, 2021), bring NLP services (Moreno-Schneider et al., 2020), studied the benefits of using artificial intelligence in legal systems (Zhong et al., 2020), fine-tuning of named entity recognition models in the legal domain (Leitner, Rehm, & Moreno-Schneider, 2019) and solutions applied to legal resolutions (Dozier et al., 2010) were proposed.

As can see, there is a broad interest in the area. For example, the company LexPredict has been working in the area since 2015. This company presents the LexNLP Python library[4] that aims to make the task of processing legal documents a more straightforward task, whether for the analysis of statutes, regulations, court opinions, briefs or the migration of legacy contracts to intelligent contracts or distributed ledger systems (Bommarito II, Katz, & Detterman, 2021).

On the other hand, Gutiérrez-Fandiño, Armengol-Estapé, Gonzalez-Agirre, and Villegas (2021) propose a Spanish legal domain language, a `RoBERTa` model available at Huggingface[5]. That effort is one of the most advanced to date but is constrained only to Spain's legal domain. As the authors state, the legal field is a Spanish variation on its own. For that, every country should be afforded independently.

[4] https://github.com/LexPredict/lexpredict-lexnlp

[5] https://huggingface.co/PlanTL-GOB-ES/RoBERTalex

2.6.2 NLP in legal text of the environmental field

Most of the work done in NLP related to the environmental field is related to water issues. However, two works are interesting. The first proposes a tool to extract and map court decisions, linking the decision's court with its geographic location (Costumero et al., 2017). The second one proposes a multidimensional study based on data and text mining of prosecuted disputes on water rights in Chile (Herrera et al., 2019).

Another related work identifies incentives in forestry policy using Transformer architecture, specifically a variant of the BERT language model, to classify text sentences. It used a human-in-the-loop approach which curates the classified data Firebanks-Quevedo et al. (2022).

2.6.3 The originality of the work

To date, no approach to named entity recognition (NER) in Chilean environmental legal texts has been reported in the literature, which gives this thesis work inherent value and significance.

However, the rapid research and development in NLP has quickly advanced beyond the technology employed in this thesis. Nevertheless, this does not diminish the utility of the proposed approach. While state-of-the-art language models require substantial computational resources and energy for training, lightweight and energy-efficient models remain valuable for specialized applications and resource-constrained environments. Such models offer practical advantages for domain-specific tasks where the trade-off between performance and computational efficiency is favorable.

2.7 Research question and objective of the work

2.7.1 Research question

How can a framework like `spaCy` provide a consistent and affordable toolbox to train neural network models at a industry level for a specific knowledge area like Chilean environmental legislation?

2.7.2 General objective

Use and validate `spaCy` framework to train a Named Entity Recognition model based on Convolutional Neural Networks as a minimum viable prototype for automatic natural language processing for Chilean environmental legal text documents.

Specific objectives

1. Identify and determine the technical requirements for automatic natural language processing with NER in Chilean environmental legal documents.
2. Validate the functional and technical requirements for designing a scalable prototype for industrial production conditions that extract legal actions from Chilean environmental legal documents.
3. Design a prototype that can extract named entities from Chilean environmental legal documents.
4. Evaluate the performance of the designed prototype in the context of the analysis of Chilean environmental legal documents.

Chapter 3

Methodology

The work conducted in this thesis required implementing several methodologies to prepare data, train a Named Entity Recognition (NER) model, evaluate it against alternative approaches through benchmarking, and develop a viable prototype for potential production deployment. To accomplish these objectives, two primary computational platforms were utilized: a Linux system with an eight-core (16 threads) Ryzen Threadripper 1900X processor and 64 GB of RAM (detailed specifications are provided in Appendix A), and Amazon Web Services (AWS) Cloud Computing Services, where the trained model was deployed as a web application instance for testing and evaluation.

The general methodology for developing the in-house NER model followed an iterative experimental approach consisting of seven sequential experiments. Each iteration involved modifications to the training data, entity tags (including expansion of tag categories), and resulting model architectures. This iterative process continued until the trained model achieved a predetermined performance threshold, specifically an F1 score exceeding 90% (detailed in Section 3.3). The seventh and final experiment is reported in this work as it incorporates all improvements and insights gained from previous iterations. It should be noted that the experimental framework focused on model development and training, while deployment and visualization components were implemented separately.

Figure 3.1 presents a simplified workflow diagram divided into five main phases. The process begins with data collection of environmental legal texts from internet sources. Following data analysis, an ontology and vocabulary were constructed, and tags were assigned to specific word groups to prepare the data for training across different platforms. Model performance was benchmarked for comparative analysis rather than decision-making purposes. The resulting models were subsequently packaged as Python modules for streamlined installation and deployment. Finally, the trained models were integrated into a web application to demonstrate their functionality and deployed as an AWS instance for remote access by external applications and developers.

The following sections provide detailed descriptions of each component involved in developing NER models for environmental legal text analysis.

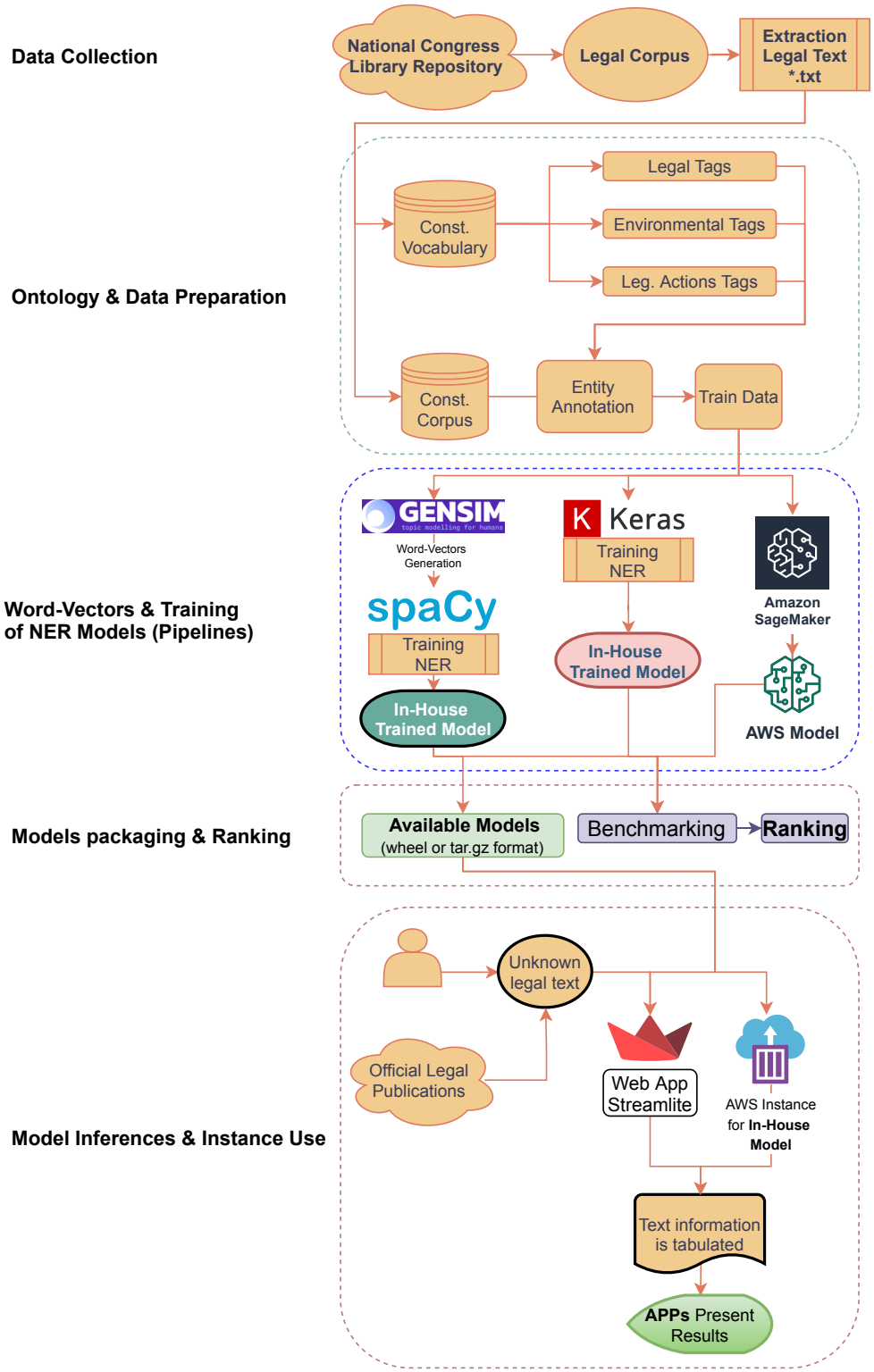

Figure 3.1: Simplified process flux from getting data to making inferences with trained models in a simple application and as AWS instance.

3.1 Collection and preparation of environmental legal data

The data for training a custom named entity recognition model (NER) was acquired from an especial website of the National Congress Library called *Ley Chile*[1]. This website has all the laws, norms and their modifications from the Chilean Congress.

3.1.1 Review of Chilean environmental laws, decrees and resolutions

In *Ley Chile*, a search was conducted with keywords like: *ambiental*, *normativa ambiental*, *ley ambiental*, among others. The main environmental legal text were identified from the search and collected it individually as raw text, where laws, decrees and resolutions were categories of the environmental legal text. There is also an extent resolution in the norms collected. Table 3.1 shows the type and number of environmental legal documents retrieved from *Ley Chile*.

Table 3.1: Type and number of environmental legal documents collected for training a custom NER model.

Documents	Number
Laws	6
Decrees	38
Resolutions	1
Ext. Resolutions	4

3.1.2 Description of the legal text obtained

Text as a part of Natural Language (NL) is characterised as a non-structured data type. By his side, the legal text is a semi-structured data type but varies widely between one another. These variations come in how text is organized, being different from a law to a resolution. For example, some texts use the named section “Title” (mainly laws),

[1] https://www.bcn.cl/leychile/

but resolutions do not use it. A widely used sub-section in the legal text is the word "Article", which always refer to a fundamental legal text structure, being used to cite a norm or a law in specific, e.g., Article 1° of law N°19,300 refer to "The right to live in a pollution-free environment …". Furthermore, the environmental legal text has the same structure as any other legal document. However, selecting the retrieved legal text used in this thesis is not biased toward any specific environmental topic being a random selection, especially in selecting decrees and resolutions.

The number of words shared by each type of document is shown in Table 3.2, with a total of 219,952 words from legal text.

Table 3.2: Number of words shared by type of document. Not considered symbols and numbers. Counts was retrieved from text processor.

Documents	**Word Numbers**
Laws	68,873
Decrees	136,990
Resolutions	14,089
Total	**219,952**

3.1.3 Preparation of environmental legal text for training and evaluation

Text manipulation was done on a Linux terminal and/or with R Statistical Software (R Core Team, 2021) using RStudio version 2021.09.1+372. The words and token statistics and remove stopwords was done with `tokenizer` and `stopwords` packages (Benoit, Muhr, & Watanabe, 2021; Mullen, Benoit, Keyes, Selivanov, & Arnold, 2018).

The legal text was concatenated in a single file and then passed by line to the software, given a total of 17,339 sentences with a mean length of 25.2 words or elements. Taking into account words, numbers, and symbols gives 436,632 elements. Without stopwords, total elements were reduced to 252,971 (still considering numbers and symbols). This number is greater than reported in Table 3.2 due to how the software counts words.

3.1.4 Definition of the name of the entities to train

Defining which elements or entities to name (group) for a custom NER model depends on the area's specific characteristics.

The entities to label (TAG) cover from legal actions to percentages. Also, it was interesting to label entities related to the legal text structure, like *Título*, *Artículo*, *Decreto*, among others. The TAG for such entities was defined as LEGEST.

A similar procedure was done with other types or groups of entities present in legal text. Firstly 12 named entities were defined in this work. Lastly, the TAGs used were 17 (see Table 3.3), intending to train the models to recognise a broad type of entities.

Table 3.3: Defined TAGs used to annotate entities.

TAG	Description
ACC	Refers to legal actions represented by verbs.
ADMB	Groups adjectives commonly used in environmental descriptions.
CAMB	Refers to environmental contaminants.
CUANT	Groups quantification entities.
ILAMB	Refers to specific environmental legal issues.
ILEG	Refers to common legal issues.
SA	Groups environmental situations typified in the norms.
SAMBL	Groups environmental subjects/nouns.
SUJ	Groups legal subjects/nouns.
VAMB	Groups verbs with environmental significance.
EST	Groups all State institutions.
LEGEST	Groups legal structures present in documents.
NUM	Groups numbers present in legal text. This TAG requires more examples.
FECHA	Groups dates and their forms present in legal text.
LUGAR	Groups typical areas present in legal text, mainly regions and cities.
ACCAMB	Groups entities related to environmental actions. Differs from VAMB by its grammatical tense.
PORC	Tag related to percentages.

The definition of the label (TAG) names has to be coherent with the aim of the work. Some approximations look for a narrow span of entities that work very well

for specialised industry text processing, resulting in small models. In this case, it is pursuing a domain-specific text like legal. However, it has to remain wider because of the amount of information in legal texts that can always be interesting to recognise. Later, this information can be used to extract and relate new or unknown legal texts like laws or norms actualisations with existing legal databases or, like in this work, recognise, annotate, and tabulate legal actions from official publications.

3.1.5 Specification of the environmental legal ontology

With `TAG`s already defined, 4,259 entities were selected from the legal text and/or arbitrary added (state institutions, dates, numbers, places and percentages) to be used for annotating it in the legal text or corpus. These entities construct the model's ontology, which can be increased or pulled over time. Table 3.4 shows the number of entities by each `TAG` type. The entities by `TAG` were saved in a `csv` file.

Table 3.4: Number of entities by `TAG` type.

TAG	Num. of entities
ACC	216
ADMB	29
CAMB	286
CUANT	91
ILAMB	141
ILEG	217
SA	49
SAMBL	139
SUJ	171
VAMB	67
EST	507
LEGEST	131
NUM	1904
FECHA	180
LUGAR	30
PORC	101
Total	**4,259**

The iterative procedure that increases the vocabulary allows for imposing the model, giving it more specificity, independent of possible associations automatically made for the model. It should be noted that these entities were used in three cases: lower, upper and upper/lower, to cover possible variations. Also, the entities were kept with accent marks as appropriate.

It is important to note that the number of entities by `TAG` can change arbitrarily. Still, annotating those changes should be done under the corpus's representativity criteria. In this case, the `TAG NUM` is by far more significant than the others where the standard followed was to cover the possible numbers present in a legal text. Nevertheless, the criteria used to annotate specific domain texts should always be done with the support of a professional in the area.

3.1.6 Annotating training text

Once the `TAGs` and ontology were defined, and before starting the training process, it was necessary to annotate selected entities in the corpus. This step is one of the most resource-intensive and time-consuming phases but is vital for two main reasons: the quantity and quality of the training data. At the beginning of this thesis, this step was performed manually using specialized software such as Doccano[2] or Prodigy[3]. However, due to the large number of entities required to improve the F1 score of the models (as observed during the initial experiments), it became necessary to transition quickly to unsupervised annotation methods. While texts from highly specialized domains can be annotated manually, this approach is extremely resource-intensive and time-consuming.

Therefore, there are several options for the annotation procedure depending on where the data will be input. In this work, two methods were used. The first was to annotate single words using `IOB` and `BILUO` tagging[4]. The second one was with a function from the `spaCy` library called `PhraseMatcher`, which allows searching and match multi-word sentences with a fixed pattern in a text and tagging with the previously defined `TAGs`. Both methods were encoded in scripts prescind manually annotated

[2]https://doccano.herokuapp.com/

[3]https://prodi.gy/

[4]https://en.wikipedia.org/wiki/Inside%E2%80%93outside%E2%80%93beginning_(tagging)

entities in a text, which is the most common way to do this in NLP.

Both approximations give exact information to the training software about the entity (`token`), the `TAG`, the entity position in the text (`span`) and if it is part of a multiword entity in a sentence (IOB and BILUO tagging format).

To achieve the above, Jupyter Notebooks (Kluyver et al., 2016) were used to annotate the corpus with several routines, mainly in `Python` (Van Rossum & Drake, 2009) and `R` scripts. This approach allows more reproducible prototyping but is not instead to replace a formal application or model training development. For more details in followed steps and code see Appendix B.

3.1.7 Converting sentences to suitable formats

In NLP, there is no specific format to process the data to train, and it will vary between software. For example, `Gensim` uses raw text, internally reads it by line, and preprocesses text to lower case. However, `spaCy` (Honnibal & Montani, 2017) in pretrain routines calls a file with the text separated by lines (`JSONL` format), but from version 3.0 it needs a binary file built with a `JSON` tagged text for training. On the other hand, training models with `Keras` framework need BILUO or IOB tagging mainly in a `JSON` file. `AWS` uses a specific way to input training data, asking for a file to point, a span with the entity and the associated `TAG` to that entity, all in a `csv` file.

Furthermore, the strategy that suits better is to take the annotated sentences obtained in the previous step saved on disk as a `JSON` file, to then pass it to the required format, depending on where it will be used. For more details in followed steps and code see Appendix B

3.2 Word vector generation with `Gensim` (`word2vec`)

Experimental work inevitably presents difficulties and challenges. One significant challenge was improving the performance scores of the trained model in `spaCy`[5]. This issue can be addressed by increasing the number of training examples and in-

[5]Large models in `spaCy` include pre-trained word embeddings that can be utilized for training purposes.

corporating word embeddings generated from domain-specific text, in this case, the same training corpus used for model development.

For this purpose, `Gensim` (Rehurek & Sojka, 2011) was employed as the framework to implement the `word2vec` algorithm (Mikolov, Chen, et al., 2013; Mikolov, Sutskever, et al., 2013) for generating word vectors corresponding to each word present in the text. The resulting vectors from `Gensim` subsequently improved model performance during training through the incorporation of domain-specific semantic representations.

To train the word embedding model in `Gensim`, two key parameters were modified: the vector dimensionality for each word was set to $N = 300$, and words with fewer than ten occurrences in the corpus were excluded from the vocabulary. All other parameters remained at their default values. Additional implementation details are provided in Appendix C.

3.3 Named Entity Recognition (NER) with `spaCy`

To train models that can recognise and annotate words automatically, it was used `spaCy`[6] (Honnibal & Montani, 2017) as the main framework. As `spaCy` is an industry-standard software written in `Cython` and designed for production use. It is a free, open-source library for advanced Natural Language Processing (NLP) in `Python`.

The framework has a central data structure upon three elements: `Language` class, the `Vocab` and the `Doc` object. The `Language` class is used to process a text and turn it into a `Doc` object. It is typically stored as a variable called `nlp`. The `Doc` object owns the sequence of `tokens` and all their annotations. By centralizing strings, word vectors and lexical attributes in the `Vocab`, it avoids storing multiple copies of this data having a single source of truth.

Text annotations are also designed to allow a single source of truth: the `Doc` object owns the data, and `Span` and `Token` are views that point to it. The `Doc` object is constructed by the `Tokenizer`, and then modified in place by the components of the **pipeline**. The `Language` object coordinates these components. It takes the raw text

[6]Fore more features you can visit this link: https://spacy.io/usage/spacy-101#features

and sends it through the **pipeline**, returning an annotated document. It also orchestrates training and serialization.

Statistical models power spaCy's tagger, parser, text categorizer and many other components. For example, every "decision" these components make – which part-of-speech tag to assign or whether a word is a named entity – is a prediction based on the model's current weight values. The weight values are estimated based on the model's examples during training. Therefore, it first needs training data – examples of text and the labels it wants the model to predict to train a model. This could be a part-of-speech tag, a named entity, or other information.

The processing pipeline consists of one or more pipeline components that are called on the Doc in order. The tokenizer (tok2vec) runs before the components. Pipeline components can be added using Language.add_pipe. They can contain a statistical model and trained weights, or only make rule-based modifications to the Doc. spaCy provides a range of built-in components for different language processing tasks and also allows adding custom components (see Figure 3.2).

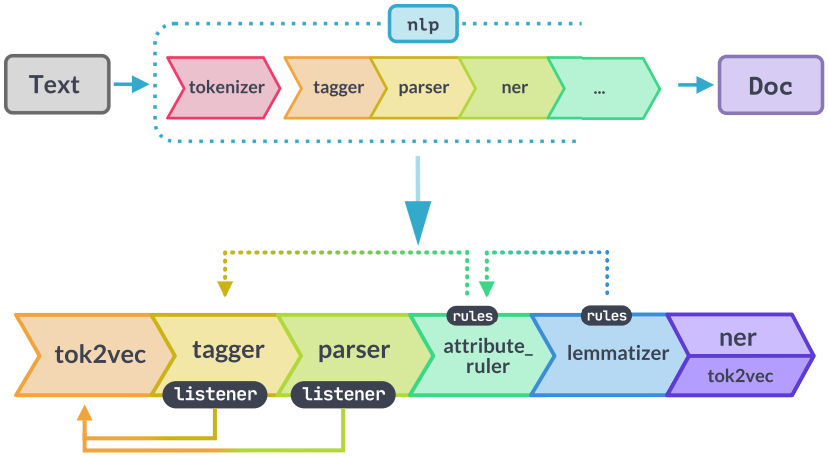

Figure 3.2: The spaCy design shows the pipeline components and how they are processed sequentially on the input text. It should be noted that the pipeline components can be modified, trained, and reordered, and that additional components such as listeners or rule-based components can be incorporated. Source: adapted from https://spacy.io/

3.3.1 Pipeline pre-training

There are numerous options available for selecting pre-trained models. `spaCy` provides its own models in different languages, but these models are primarily trained on news texts. This presents the disadvantage that the embedded vectors and associated word and sentence semantics are structured for contexts that are too broad or colloquial for specialized domains. The problem that emerged during experimentation was that the `tok2vec` component of the pipeline was too general for the specific legal environmental domain. To address this issue, `spaCy` offers the option to pre-train this component specifically for subsequent use in training the NER component.

The pre-training process initializes a `tok2vec` layer in the pipeline with information derived from raw text. To accomplish this, additional layers are incorporated to construct a network for a temporary task that compels the `tok2vec` layer to learn sentence structure and word co-occurrence statistics. `spaCy` provides two pre-training objectives, both variants of the close task introduced by Devlin et al. (2018) for BERT. The implementation followed the documentation provided by `spaCy`[7].

3.3.2 Pipeline training

As shown in Figure 3.3 training is an iterative process in which the model's predictions are compared against the reference annotations to estimate the loss's gradient. The loss gradient is then used to calculate the gradient of the weights through backpropagation. The gradients indicate how the weight values should be changed so that the model's predictions become more similar to the reference labels over time.

[7] https://spacy.io/usage/embeddings-transformers#static-vectors

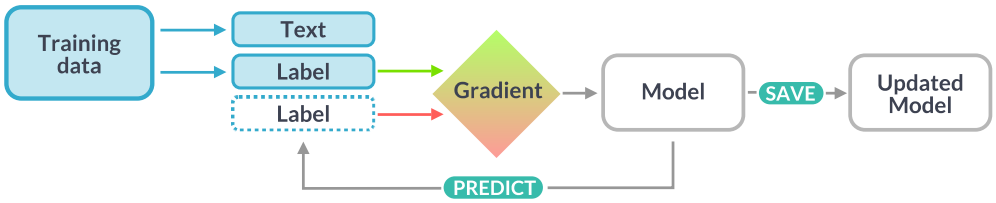

Figure 3.3: Training data: Examples and their annotations. Text: In the input text, the model should predict a label. Label: The label the model should predict. Gradient: The direction and rate of change for a numeric value. Minimising the gradient of the weights should result in predictions closer to the reference labels on the training data. Source: https://spacy.io/usage/training

To train the environmental legal NER model were ran several experiments. First, as a base framework, it was used a NER project template from `spaCy`'s repository[8], which was later adapted. The project template prepares for the user an end-to-end `spaCy` workflow that is based on a `proyect.yml` file that defines the assets a project depends on, like datasets and pre-trained weights, as well as a series of commands that can be run separately or as a workflow. For instance, to preprocess the data, convert it to `spaCy`'s format, train a pipeline, evaluate it and export metrics, package it and spin up a quick web demo. Below is an example directory tree from `spaCy`'s project website[9].

```
project.yml          # the project settings
project.lock         # lockfile that tracks inputs/outputs
assets/              # downloaded data assets
configs/             # pipeline config.cfg files used for training
corpus/              # output directory for training corpus
metas/               # pipeline meta.json templates used for packaging
metrics/             # output directory for evaluation metrics
notebooks/           # directory for Jupyter notebooks
packages/            # output directory for pipeline Python packages
scripts/             # directory for scripts, e.g. referenced in␣
↪commands
```

[8]This option is available since version 3.0.

[9]https://spacy.io/usage/projects.

```
training/           # output directory for trained pipelines
...                 # any other files, like a requirements.txt etc.
```

From the directory structure shown above, it was used mainly all of them (were left out `metas` and `notebooks` directories). However, it was added a directory for `Gensim` models with word vectors.

All the training routines were first to run in the console, but, in Appendix D it is shown a Jupyter notebook output with the training process of `tok2vec` and NER models, where it can see the commands used for:

1. Debug the data.
2. Train the models (`tok2vec` & NER).
3. Evaluate the models.
4. Package the models.

The neural network architectures used in training are the default settled in the configuration file, with minor modifications. As `spaCy` uses its own framework to construct their networks (`Thinc`), those are called with specific functions (see Appendix **??** for the details of configuration file) that call the `spaCy` architecture model and pass the parameters to it.

Before to load the neural network models. a `tokenizer` (`spacy.Tokenizer.v1`) that segment text and create a `Doc` objects with the discovered segment boundaries is called (loaded). Usually, the `tokenizer` is created automatically when the language subclass is initialised, and it reads its settings like punctuation and special case rules from the defaults provided by the language subclass. Follow in, a parser model is built(`spacy.TransitionBasedParser.v2`). Transition-based parsing is an approach to structured prediction where the task of predicting the structure is mapped to a series of state transitions. So, the neural network state prediction model consists of either two or three subnetworks: a `tok2vec` that map each token into a vector representation (ran once for each batch), a `lower` that constructs a feature-specific vector for each *token-feature* pair (ran once for each batch), the state representation is built summing the component features and applying the non-linearity function (that was

not used in this case), and an `upper` that construct a feed-forward neural network that predicts scores from the state representations. Those subnetworks use 64 hidden layers with two outputs.

Next, a listener layer is used (`spacy.Tok2VecListener.v1`) that act as a proxy(ies), passing the predictions from the `tok2vec` network to other components of the pipeline, and communicate gradients back upstream.

3.3.3 Pipeline metrics and evaluation

The metrics and evaluation used by `spaCy` to control the training are related to calculation of loss functions for `tok2vec` and `ner` models training. Also, the precision (P) and recall (R) were obtained and with those values the F1 score was calculated. In sections 2.3.1 and 2.3.2 the equation used for those calculations are presented.

To set the performance of the trained model it was done by a per-entity PRF scores (Precision, Recall, F1). No other scores available in `spaCy` were used.

3.3.4 Pipeline packaging

As previously noted, the training was an iterative process with several experiments. From those experiments, at least one trained model was obtained and named with version numbers. The last version trained was the 7.0.9 version (an arbitrary number). The first number refers to the associated experiment, and the last number corresponds to the last iteration.

3.3.5 Pipeline distribution/loading

The packaged pipeline stores inside the trained model components and configuration. As the pipeline can be installed in the host system using the `Python` library `pip`, those components should be called by `spaCy` from command line or from a application that need to use those components.

3.3.6 Evaluation and testing of the pipeline with environmental legal text not known by the model

The trained pipeline was evaluated using unknown data for the model. This data has the same structure that the train data. For that, `spaCy` has a particular function for evaluating a trained model that works directly in the project and is configured in the `project.yml` file (see Section 3.3.2). The metrics calculated are consistent with that shown in Section 3.3.3. The evaluation results are stored in a `JSON` format file, which was later used to present the results of this thesis.

3.3.7 Training validation analysis

The validation of the training process, hyper-parameters explicitly settings, was done with a k-fold cross-validation analysis with k = 10 using the entire dataset of legal text, computing Precision, Recall and F1 scores.

The general process was as follows:

- Shuffle the dataset randomly.
- Split the dataset into k groups.
- For each unique group:
  - Take the group as a holdout or test data set.
  - Take the remaining groups as a training data set.
  - Fitted a model on the training set and evaluated it on the test set.
  - Retain the evaluation score and discarded the model.
- Summarised the model's skill using the sample of model evaluation scores.

Its important to note that, each observation in the data sample is assigned to an individual group and stays there for the procedure. Each sample can be used in the holdout set once and to train the model $k-1$ times.

The selected fold gives a good trade-off between bias and variance (James, Witten, Hastie, & Tibshirani, 2021). Next, the sentences were shuffled and splited into training and evaluation (test) sets with a 90:10 - ratio. Later, the training set was split

similarly into training and developing sets. Finally, the training for the ten datasets was done using `spaCy`'s CLI, setting the patience parameter to 100, and using the pre-train model and vectors prepared in previous steps (see Sections 3.2 and 3.3). For more details, please see Appendixes E and F.

3.3.8 Demo application for extract legal actions

The trained model with `spaCy` was visualised using Streamlit framework[10], a Python-based software specifically prepared to design and build web applications and the `spacy-streamlit` library that allows translating `spaCy`'s objects to a web framework.

The application was designed for two main things, to use the custom-trained model in environmental legal NER and to extract legal and environmental actions from a given text. However, it can be used for any 17 TAGs recognised by the model. Therefore, the strategy to get the actions in the text was to use a hybrid model:

1. Load the custom-trained model in its last version and get the `tok2vec` and `ner` components.
2. Load from the large pre-trained model from `spaCy` the `parser` component that will analyse the dependency in text sentences using a variation of non-monotonic arc-eager transition-system described by Honnibal and Johnson (2015).
3. With those components loaded in a shared pipeline, the application will process texts passed by the user by a web interface.
4. In the process, the application will tokenise, annotate and get the dependency between words in the text.
5. Then, it will display the annotated text, a resume with the TAGs and a table with the actions (word recognised). Alternatively, the TAG searched can be changed to any other.

The web application for deployment was containerised with Docker[11] (Merkel,

[10] https://streamlit.io/
[11] https://www.docker.com/

2014). When the container is launched, the Streamlit server starts the application and can be accessed in the localhost direction in port 8501. However, the containerised application could be served at any internet domain.

Chapter 4

Results

4.1 Data preparation

Following the methodology proposed in section 3.1.6 and 3.1.7, the whole process of preparing the necessary training data is shown in more detail in Appendix B and follows the next sequence:

1. Import the required libraries.
2. Build upon the `spaCy` pre-trained large model.
3. Load text document and key entities/TAGs files to build patterns (keyword dictionary).
4. Check text length.
5. Import and apply `PhraseMatcher` function to keyword dictionary.
6. Check if the matcher is working.
7. Build the Corpus with sentences to annotate.
8. Check the Corpus length.
9. Build the patterns.
10. Creating an entity ruler with a blank *spaCy* model and the patterns.
11. Saving a `JSON` file with the training data and visualizing.
12. Transforming the data in `JSON` format to `DocBin` (`spaCy` format).

4.1.1 Annotated legal text

The processed legal text gives a corpus with a length of 1,416,885 tokens grouped in 6,593 sentences separated by a line. Then, the exact length of sentences was obtained for the train data in `JSON` format, which is the main format used. Then, the resultant train data was transformed into the following formats:

1. `JSONL` for `spaCy` pre-train and `DocBin` format.
2. `BILUO` format, for use in other frameworks like Keras/Tensorflow or AWS.
3. `DocBin` format for training.

Those transformations between formats give some flexibility to the whole training process.

4.2 Neural model with environmental legal text word vectors from Gensim

A neural network was trained with `Gensim` and the `word2vec` algorithm to obtain the word embeddings that will be used later as word vector weights in the NER training model. First, `Gensim` collected 9,298-word types from a corpus of 190,472 raw words and 11,927 sentences. Then, the model was trained by eight workers on 4,519 vocabulary words, 300 features and 3,809,440 raw words (2,524,663 effective words). The training time took 6.7 seconds with 374,721 effective words per second processing speed.

The similarity between pairs of words was checked. For that, six pairs of words were used all compared with the word "artículo" (see Table 4.1).

Table 4.1: Word pairs similarity checked.

Word 1	Word 2	Similarity
artículo	*ley*	0.71
artículo	*decreto*	0.85
artículo	*resolución*	0.35
artículo	*ambiental*	0.23
artículo	*contaminante*	-0.02

As is shown in the table above, the similarity between pairs *artículo* - *ley* and *artículo* - *decreto* have the highest similarity. Conversely, the pair *artículo* - *contaminante* has the lowest score. This result shows that `word2vec` algorithm can put together similar words even with not a high amount of words to process. Nevertheless, `Gensim`'s corpus was pre-processing to lowercase all words and removing punctuations and numbers; for that reason, the model does not recognize words that begin with the capital letter o upper case words.

Moreover, in Table 4.2 the positive relations for the keyword *contaminante* give an idea that the model also captures context between words.

Table 4.2: Positive similarity for the word *contaminante* where the higher the score more similar the word detected by the model. Note: *so* refers to contaminant Sulfur Monoxide.

Word	Positive Similarity
efluente	0.726
elemento	0.724
balance	0.696
parámetro	0.688
porcentaje	0.683
so	0.672
arsénico	0.672
medir	0.663
cálculo	0.662
magnitud	0.656

The obtained word vectors were stored in text format and as a binary file. Later, both files were passed to the `spaCy` training process. For more details and code review, see Appendix C.

4.3 NER model in Chilean environmental legal text

4.3.1 Pre-trained model

Following the documentation proposed by `spaCy`[1], it was built the `config_pretrain.cfg` file and modify `project.yml` files accordingly. However, initializing pre-training asks for parameters with not assigned values. To solve this, the default options that came from the documentation were removed, leaving a definitive version of the pre-training configuration.

Later, it was checked if the files passed to the pre-training were correct and to solve those problems, the pre-train data was again processed, the text was cleaned of "artefacts", and the binary file was created again in `spaCy` format. When pre-training, both files (`JSONL` and `DocBin` formats) were passed to `spaCy`, which did not give errors during the pre-train process. Indeed, it is unclear which of the two files was used

[1] https://spacy.io/usage/embeddings-transformers#static-vectors

for the pre-training or if they were both since in the pre-training configuration (see Appendix D), the path to the files were passed without identifying them.

The pre-training was carried out for 1,000 epochs (five hours app.), starting with a loss of 3.223 and ending with 0.137 in 999 epochs. The word frequency processed was in the order of magnitude of 8×10^3 words per second (w/s), processing more than 150 million words. This last value does not refer to the total number of words in the pre-training text but rather the total number of processed words.

The obtained pre-trained model was linked via the training file (`config.cfg`) to be used for training. It should be noted that the pre-training trains the layer of the `tok2vec` component of the model used by `spaCy` and that, given the training strategy, all other components, including NER, use it.[2]

Next box, shown the console output of `tok2vec` pre-training:

```
(.env) anibal@solidU:~/gitsources/tesis-estadistica/03-Experimentos/
 ↪Exp-7$ python -m spacy pretrain config_pretrain.cfg ./pretrain
 Output directory is not empty.
It is better to use an empty directory or refer to a new output path,␣
 ↪then the
new directory will be created for you.
 Using CPU
 To switch to GPU 0, use the option: --gpu-id 0
 Loading config from: config_pretrain.cfg
 Saved config file in the output directory
[W Module.cpp:482] Warning: Disabling benchmark mode for MIOpen is NOT␣
 ↪supported. Overriding value to True (function operator())

============== Pre-training tok2vec layer - starting at epoch 0␣
 ↪==============
 #      # Words   Total Loss     Loss    w/s
  0        11557       3.2227   3.2227   8491
  0        23307       4.9939   1.7712   8904
  0        35227       6.3427   1.3488   8749
```

[2] **Note**: Word vectors trained with `Gensim` were loaded for pre-training.

```
  0        47168       7.5661   1.2234   8706
.
. truncated output
.
  999    150590892    2587.0077   0.1471   8530
999    150602806    2587.1394   0.1317   8821
999    150614771    2587.2959   0.1565   8851
999    150626682    2587.4475   0.1516   8545
999    150638000    2587.5846   0.1371   8382
 Successfully finished pretrain
```

4.3.2 Trained `tok2vec` and NER models

From the iterative training, the last model version obtained was `7.0.9`. This last model was trained on 32,442 training and evaluation docs. This implies 1,157,725 total word(s) in the data (16,814 unique). As was noted before, it was used 4,519 vectors (4,519 unique keys with 300 dimensions). Of the total words in training data, 524,415 were without vectors (45%). It used 17 labels (`TAGs`) with a good amount of examples for all labels, and the data has examples without occurrences available for all labels.

Table 4.3 it shows the general results for the last trained model. It can be noted that the F1 has a mean value of 94.88%, and the `tok2vec` is 100.00%, a perfect score. This means that both scores give high confidence in the model training.

Table 4.3: General results for model evaluation. The table shows the overall mean for tokenization score, NER *Precision*, NER *Recall*, NER F1 and the *Speed* for the model version `7.0.9`.

Score	Value
tok2vec	100.00
NER P	95.44
NER R	94.33
NER F1	94.88
Speed	25,270

Table 4.4 it shows a detail on the metrics per-TAGs for the model version 7.0.9. The harmonic mean between precision and recall scores is higher with the NUM TAG (98.38%) that recognises numbers present in the corpus and is lower with the SA TAG (65.73%) that labels words related to environmental situations. The mean value for the scores was shown in the Table 4.3.

The significant difference between TAGs should be produced by the number of examples present in the corpus per TAG. As we can see, the precision is high in all the TAGs, being, at last, the recall score that influences; notice that the model had more difficulties in recognising some TAGs over others.

Table 4.4: Results per NER type. The table shows the NER *Precision*, NER *Recall*, NER F1 per-type of TAG for model version 7.0.9.

	Scores		
TAGs	**P**	**R**	**F1**
LEGEST	96.97	96.97	96.97
NUM	98.14	98.62	98.38
SAMBL	89.50	93.91	91.65
CAMB	95.47	92.99	94.21
ACCAMB	91.40	93.36	92.37
ILEG	96.88	95.22	96.04
ACC	97.43	95.62	96.51
ADMB	96.61	82.61	89.06
SUJ	94.39	91.42	92.88
CUANT	97.17	97.15	97.16
LUGAR	90.79	91.25	91.02
SA	95.92	50.00	65.73
EST	92.67	92.88	92.78
ILAMB	86.83	75.90	81.00
VAMB	94.41	70.31	80.60
PORC	100.00	89.29	94.34
FECHA	91.11	92.09	91.60

Compared with the latest spaCy's pre-trained large model for Spanish available[3], that have a *Precision*, *Recall* and F1 scores of 89.76, 89.79 and 89.77, respectively; had a mean scores of 5.53% lower than the trained model from scratch. This result surpasses

[3] es_core_news_lg-3.4.0

the accuracy evaluation of the spaCy's pre-trained model.

Figure 4.1 shows the training behaviour by epoch of five metrics of NER training of version 7.0.9 model: General score, F1, *Precision*, *Recall* and NER Loss function. The first four metrics are expressed as a percentage and have a similar behaviour obtaining the best result near the 1,000 epoch. As the value of the Loss function is meaningless, this value is always desirable to be as low as possible, but in this case, the best model does not match the minimum value of the Loss function.

As well, from the iterative process of training models and as a result of the improvements done, Figure 4.2 show the resultant metrics per model separated into two groups by versions: 4 - 5 and 6 - 7 versions. Mainly both groups differ from the TAGs used during the training because the first group has not grouped the legal structure in one only TAG, and the second has it. In the first group, version 4.0.3 has a poor result in all the metrics compared with the precedent and posterior versions (4.0.1 and 5.0.6). The reason is that in the precedent version, the training was done using the vectors embedding provided by the spaCy pre-training model (version 3.0.0). In the posterior version, the vector embeddings were trained with Gensim, and the pre-train procedure was used. That experiment shows the importance of having trained word vectors for the training process, especially if the model is trained for accuracy.

Similarly, as above, Figures 4.3 and 4.4 shows the F1 score per version model and by TAGs. It can be noted that the TAGs differ between figures, and the behaviour is not the same by each TAG in the different models. For example, the TAG NUM has a value near 1.00, and this is because of the high number of examples given during training. In contrast, the TAG ILAM or SA has few examples. This antecedent drives where to put the effort to improve the model performance.

The results presented above reveal the nature of Deep Learning models that need large amounts of examples to improve their performance, not only globally but also every time a new TAG is added (as was the case in this work), the performance varies. Those TAGs with few examples are falling behind. The strategy used for training, where words are chosen and associated with a TAG, gives some flexibility. However, if these chosen words are not frequently found in the corpus, performance will be low for that TAG, affecting the global performance. In that way, the trainer needs to add more annotated examples to the train data.

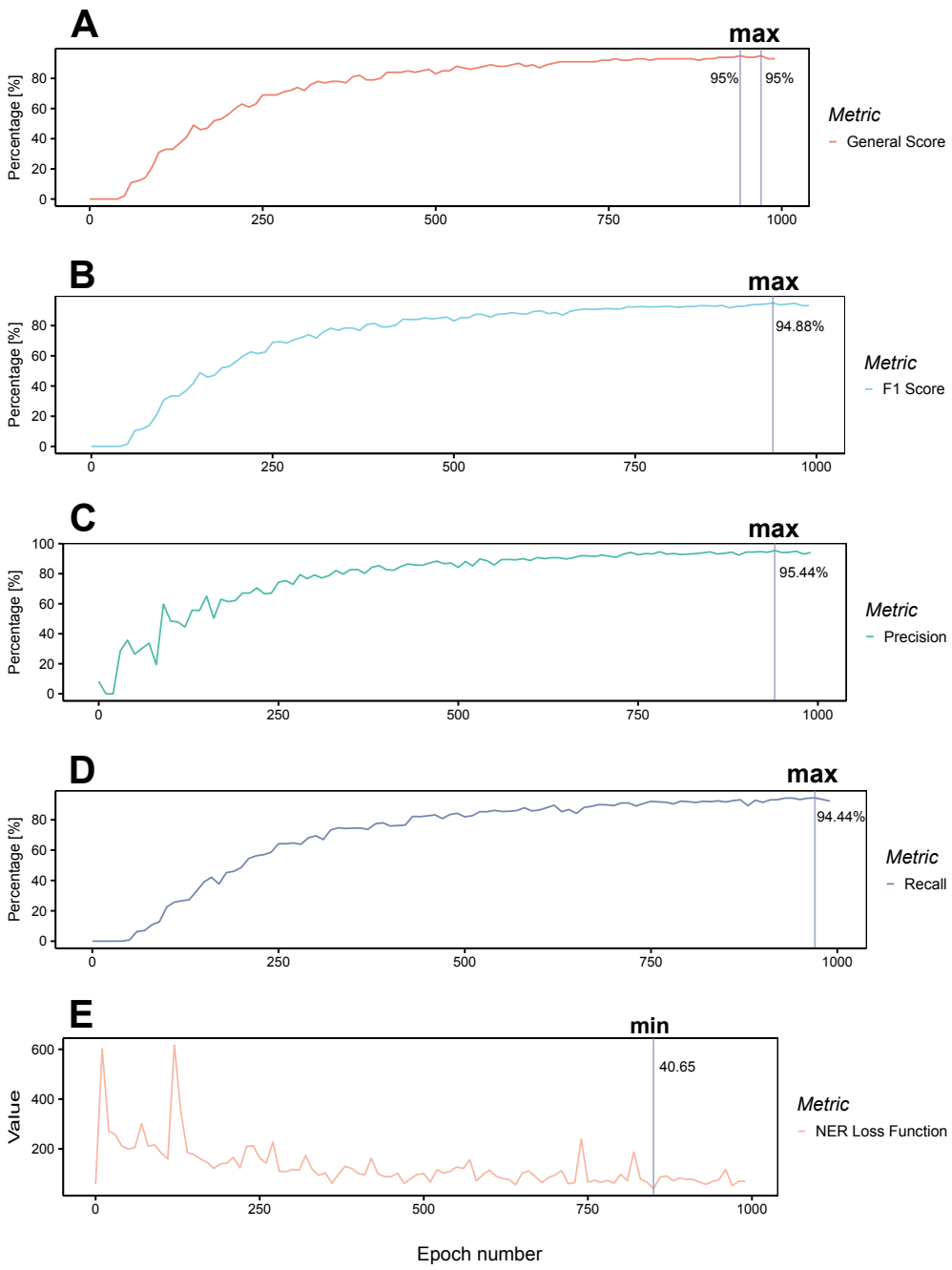

Figure 4.1: Five metrics results from training process by epochs number. **A**: General score expressed as percentage; **B**: F1 expressed as percentage; **C**: Precision score expressed as percentage; **D**: Recall score expressed as percentage; **E**: NER loss function score. Vertical lines point to maximum or minimum value.

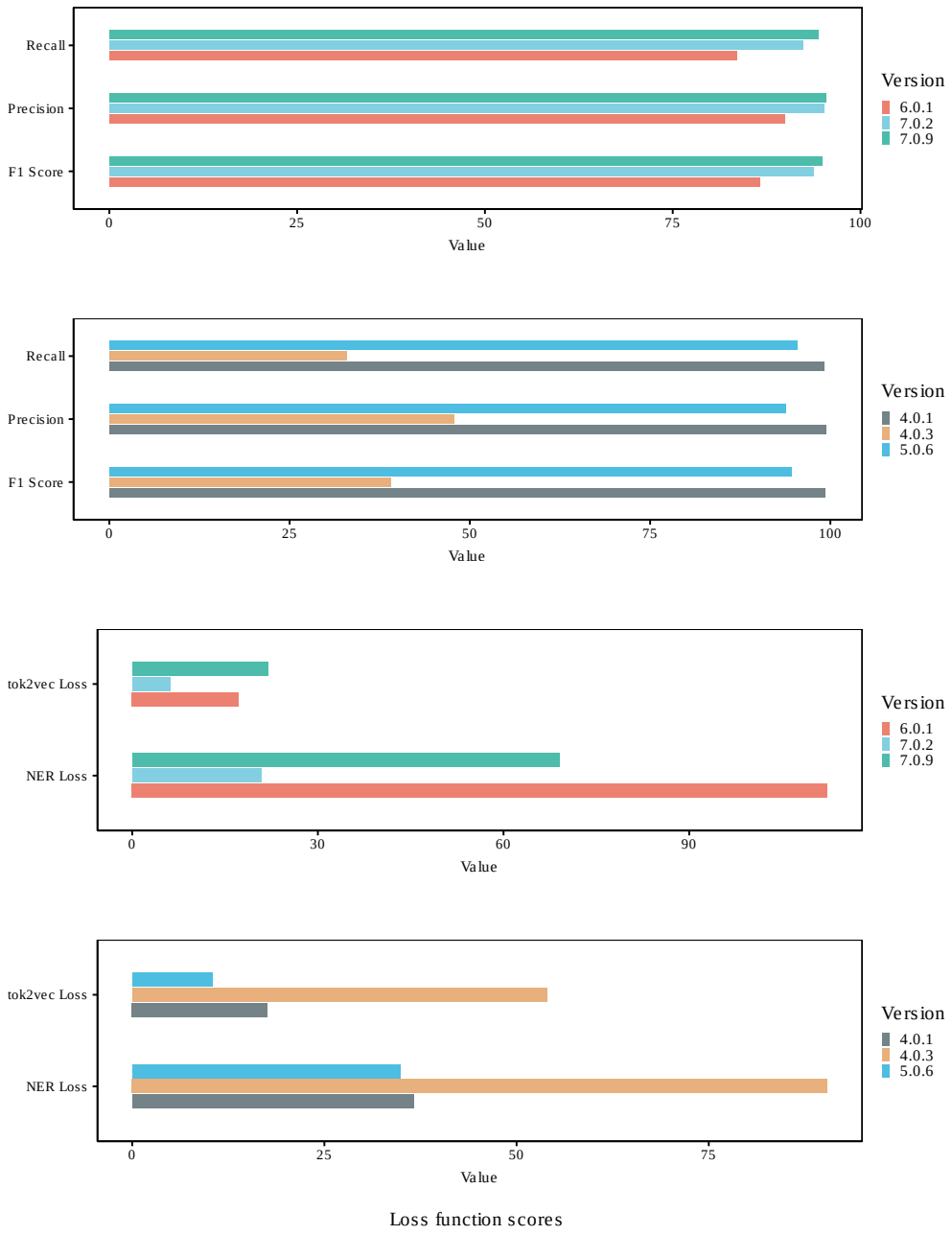

Figure 4.2: Metrics Recall, Precision, F1 score, `tok2vec` and NER Loss function per model version. The value is expressed as percentage.

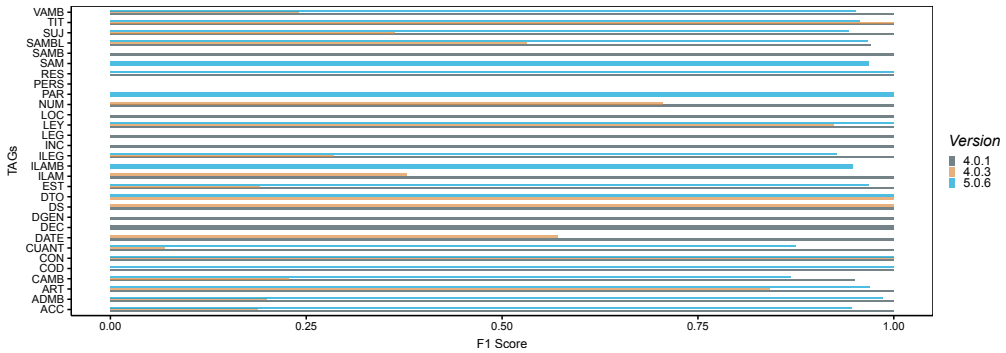

Figure 4.3: F1 score per model versions (4 - 5) and by `TAGs`. The value is expressed as unity.

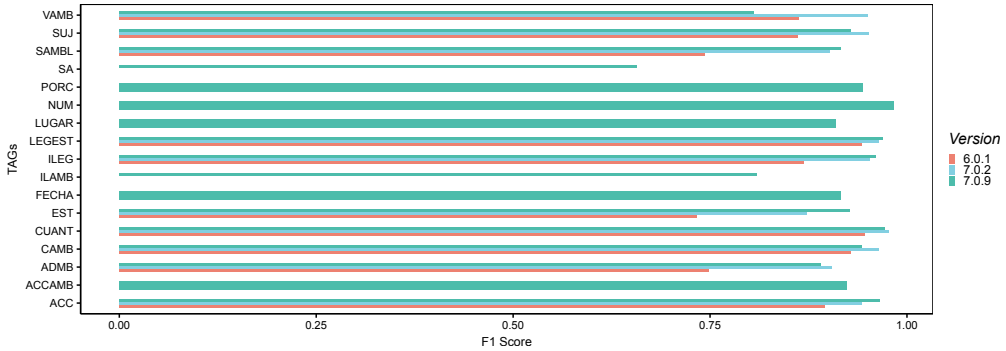

Figure 4.4: F1 score per model versions (6 - 7) and by `TAGs`. The value is expressed as unity.

4.3.3 K-Fold Crossvalidation of models trained with `spaCy`

To satisfy the validation requirement proposed in the general objective of training a NER model with the `spaCy` framework that can annotate Chilean environmental legal text, a k-Fold crossvalidation with a $k = 10$. As `spaCy` has not implemented the method, it was necessary to implement it as shown in Section 3.3.6. For more details in the implementation of the method and results please see Appendix E and F.

The results for the k-Fold crossvalidation are presented in Tables 4.5, 4.6 and 4.7.

Table 4.5 shows a summary with the global mean and its standard mean error (SME) for each score: Precision, Recall and F1. It can be noted that the F1 score has a global mean of 91.76, which is lower than the results shown for model version `7.0.9`.

That indicates that some folds have much lower results depending on the examples in the data.

Table 4.5: Global results for a k-10 crossvalidation with an N of 170 observations. The mean and SEM (Standard Error of the Mean) are expressed as a percentage.

Precision mean	Precision SEM	Recall mean	Recall SEM	F1 mean	F1 SEM
93.075	0.459	90.879	0.718	91.764	0.55

Table 4.6 shows the mean and its standard error (SME) for each score: Precision, Recall and F1, by fold. It can see that the scores fluctuate between folds relying upon the number of examples in the data for each `TAG`.

On the other hand, the k-Fold crossvalidation varies between `TAGs` that are actually with what the scores are built. For example, Table 4.7 shows a cross-table with four statistics, minimum and maximum values, the median with interquartile range (IQR), the mean with its standard deviation and the number of observations for each `TAG` by type of score: Precision, Recall and F1. It can be observed that `SA` and `ILAM` `TAGs` have a relative low F1 score (see min/max values) compared with the other `TAGs`. Furthermore, even though the training data was randomized, some folds may keep a few annotated examples, further affecting these labels.

Table 4.6: Main results for a k-10 crossvalidation with an N of 17 observations by fold. The mean and SEM (Standard Error of the Mean) are expressed as a percentage.

Fold	Precision mean	Precision SEM	Recall mean	Recall SEM	F1 mean	F1 SEM
K-1	94.370	0.275	92.793	0.526	93.512	0.385
K-2	92.846	0.415	86.420	0.920	89.135	0.640
K-3	92.860	0.380	91.365	0.667	92.005	0.507
K-4	94.685	0.360	92.223	0.614	93.328	0.452
K-5	92.040	0.411	88.512	0.678	90.140	0.526
K-6	95.619	0.379	94.589	0.426	95.030	0.357
K-7	91.852	0.522	90.864	0.680	91.190	0.556
K-8	91.992	0.614	93.415	0.453	92.546	0.480
K-9	92.795	0.586	90.488	0.838	91.287	0.622
K-10	91.694	0.544	88.124	1.062	89.463	0.803
Mean	93.075	0.449	90.879	0.686	91.764	0.533

Table 4.7: The group of crosstables shows scores with summary statistics of the k-10 Cross-validation analysis by tags (17 in total). The extreme values, median with its interquartile range, the mean with its standard deviation and N are presented as units. NA: refers to missing data.

Score	Statistics	ACC	ACCAMB	ADMB	CAMB	CUANT
Precision	Min / Max	0.93 / 0.98	0.89 / 0.97	0.72 / 0.93	0.89 / 0.96	0.96 / 0.99
	Med [IQR]	0.96 [0.95;0.97]	0.94 [0.92;0.95]	0.89 [0.84;0.90]	0.95 [0.93;0.95]	0.98 [0.97;0.99]
	Mean (std)	0.96 (0.02)	0.94 (0.02)	0.86 (0.07)	0.94 (0.02)	0.98 (0.01)
	N (NA)	10 (0)	10 (0)	10 (0)	10 (0)	10 (0)
Recall	Min / Max	0.92 / 0.99	0.88 / 0.95	0.82 / 0.88	0.85 / 0.95	0.95 / 1.00
	Med [IQR]	0.98 [0.96;0.98]	0.92 [0.91;0.94]	0.85 [0.82;0.88]	0.92 [0.89;0.93]	0.98 [0.97;0.99]
	Mean (std)	0.97 (0.02)	0.92 (0.02)	0.85 (0.03)	0.91 (0.03)	0.98 (0.02)
	N (NA)	10 (0)	10 (0)	10 (0)	10 (0)	10 (0)
F1	Min / Max	0.93 / 0.98	0.89 / 0.96	0.78 / 0.89	0.90 / 0.96	0.95 / 0.99
	Med [IQR]	0.96 [0.96;0.97]	0.93 [0.92;0.93]	0.86 [0.84;0.88]	0.93 [0.91;0.94]	0.98 [0.97;0.99]
	Mean (std)	0.96 (0.01)	0.93 (0.02)	0.85 (0.04)	0.93 (0.02)	0.98 (0.01)
	N (NA)	10 (0)	10 (0)	10 (0)	10 (0)	10 (0)

Table 4.7: *(Continued)*

Score	Statistics	EST	FECHA	ILAMB	ILEG
Precision	Min / Max	0.89 / 0.95	0.88 / 0.98	0.76 / 0.90	0.93 / 0.97
	Med [IQR]	0.93 [0.92;0.94]	0.94 [0.91;0.95]	0.83 [0.78;0.87]	0.96 [0.95;0.96]
	Mean (std)	0.93 (0.02)	0.93 (0.03)	0.83 (0.05)	0.95 (0.02)
	N (NA)	10 (0)	10 (0)	10 (0)	10 (0)
Recall	Min / Max	0.90 / 0.96	0.68 / 0.98	0.63 / 0.83	0.96 / 0.99
	Med [IQR]	0.92 [0.91;0.94]	0.95 [0.90;0.97]	0.71 [0.68;0.79]	0.98 [0.96;0.99]
	Mean (std)	0.93 (0.02)	0.92 (0.09)	0.73 (0.07)	0.98 (0.01)
	N (NA)	10 (0)	10 (0)	10 (0)	10 (0)
F1	Min / Max	0.90 / 0.95	0.81 / 0.96	0.69 / 0.86	0.94 / 0.98
	Med [IQR]	0.93 [0.92;0.94]	0.94 [0.90;0.95]	0.78 [0.73;0.80]	0.97 [0.96;0.97]
	Mean (std)	0.93 (0.02)	0.92 (0.05)	0.77 (0.06)	0.96 (0.01)
	N (NA)	10 (0)	10 (0)	10 (0)	10 (0)

Table 4.7: *(Continued)*

Score	Statistics	LEGEST	LUGAR	NUM	PORC
Precision	Min / Max	0.95 / 1.00	0.90 / 1.00	0.97 / 0.99	0.84 / 1.00
	Med [IQR]	0.98 [0.97;0.99]	0.95 [0.94;0.99]	0.98 [0.97;0.98]	1.00 [0.94;1.00]
	Mean (std)	0.98 (0.01)	0.96 (0.04)	0.98 (0.01)	0.96 (0.06)
	N (NA)	10 (0)	10 (0)	10 (0)	10 (0)
Recall	Min / Max	0.96 / 0.99	0.96 / 1.00	0.96 / 0.99	0.69 / 1.00
	Med [IQR]	0.98 [0.97;0.98]	0.98 [0.98;0.98]	0.98 [0.97;0.98]	1.00 [0.94;1.00]
	Mean (std)	0.98 (0.01)	0.98 (0.01)	0.98 (0.01)	0.95 (0.10)
	N (NA)	10 (0)	10 (0)	10 (0)	10 (0)
F1	Min / Max	0.96 / 0.99	0.94 / 0.99	0.96 / 0.99	0.76 / 1.00
	Med [IQR]	0.98 [0.97;0.99]	0.96 [0.96;0.99]	0.98 [0.97;0.98]	0.97 [0.95;1.00]
	Mean (std)	0.98 (0.01)	0.97 (0.02)	0.98 (0.01)	0.95 (0.07)
	N (NA)	10 (0)	10 (0)	10 (0)	10 (0)

Table 4.7: *(Continued)*

Score	Statistics	SA	SAMBL	SUJ	VAMB
Precision	Min / Max	0.67 / 0.89	0.86 / 0.96	0.89 / 0.97	0.84 / 1.00
	Med [IQR]	0.87 [0.80;0.88]	0.92 [0.92;0.94]	0.94 [0.93;0.96]	1.00 [0.93;1.00]
	Mean (std)	0.82 (0.08)	0.92 (0.03)	0.94 (0.02)	0.96 (0.06)
	N (NA)	10 (0)	10 (0)	10 (0)	10 (0)
Recall	Min / Max	0.44 / 0.89	0.91 / 0.96	0.86 / 0.94	0.56 / 0.89
	Med [IQR]	0.78 [0.78;0.89]	0.92 [0.92;0.93]	0.90 [0.88;0.90]	0.81 [0.78;0.83]
	Mean (std)	0.78 (0.14)	0.93 (0.02)	0.89 (0.02)	0.78 (0.09)
	N (NA)	10 (0)	10 (0)	10 (0)	10 (0)
F1	Min / Max	0.57 / 0.89	0.89 / 0.96	0.89 / 0.95	0.71 / 0.91
	Med [IQR]	0.82 [0.77;0.82]	0.93 [0.92;0.94]	0.91 [0.91;0.92]	0.87 [0.85;0.90]
	Mean (std)	0.79 (0.09)	0.93 (0.02)	0.92 (0.02)	0.86 (0.06)
	N (NA)	10 (0)	10 (0)	10 (0)	10 (0)

It is also interesting to show k-Fold crossvalidation summary statistics for each score by fold. For example, Table 4.8 shows the total number (n), minimum and maximum values (min, max), the median, first and third quartile and the interquartile range, the median absolute deviation (mad), the mean value with its standard deviation (sd), the standard error (se) and the confidence interval (ci), for each score obtained from the analysis. It can be noted that the second, fifth and tenth folds have the lowest mean value for the F1 score.

For clarity and to graphically show if it exists some difference between the means obtained from the k-Fold crossvalidation analysis, was prepared a boxplot with a Wilcoxon test. That analysis did not show any significant difference between means (see Appendix D). The results are shown in Figure 4.5, where the Precision, Recall and F1 scores are presented by fold.

Table 4.8: Summary statistics for F1, Precision and Recall scores are grouped by fold. The statistics shown are total number (n), minimum and maximum values (min, max), the median, first and third quartile and the interquartile range, the median absolute deviation (mad), the mean value with its standard deviation (sd), the standard error (se) and the confidence interval (ci). Values are expressed as units.

Fold	Scores	n	min	max	median	q1	q3	iqr	mad	mean	sd	se	ci
K-1	F1	17	0.836	1.000	0.951	0.915	0.974	0.059	0.040	0.935	0.050	0.012	0.026
K-1	Precision	17	0.865	1.000	0.946	0.933	0.974	0.041	0.041	0.944	0.036	0.009	0.018
K-1	Recall	17	0.778	1.000	0.958	0.897	0.978	0.081	0.040	0.928	0.069	0.017	0.035
K-2	F1	17	0.714	0.989	0.924	0.871	0.945	0.074	0.059	0.891	0.083	0.020	0.043
K-2	Precision	17	0.792	1.000	0.949	0.896	0.956	0.060	0.029	0.928	0.054	0.013	0.028
K-2	Recall	17	0.556	0.978	0.902	0.818	0.947	0.128	0.086	0.864	0.120	0.029	0.062
K-3	F1	17	0.764	1.000	0.926	0.903	0.970	0.067	0.066	0.920	0.066	0.016	0.034
K-3	Precision	17	0.824	1.000	0.939	0.889	0.959	0.070	0.057	0.929	0.050	0.012	0.026
K-3	Recall	17	0.683	1.000	0.925	0.886	0.979	0.093	0.079	0.914	0.087	0.021	0.045
K-4	F1	17	0.803	1.000	0.944	0.907	0.981	0.074	0.055	0.933	0.059	0.014	0.030
K-4	Precision	17	0.850	1.000	0.952	0.921	0.986	0.065	0.050	0.947	0.047	0.011	0.024
K-4	Recall	17	0.761	1.000	0.956	0.882	0.978	0.096	0.049	0.922	0.080	0.019	0.041
K-5	F1	17	0.716	0.972	0.918	0.877	0.959	0.081	0.061	0.901	0.069	0.017	0.035
K-5	Precision	17	0.762	1.000	0.927	0.900	0.965	0.065	0.056	0.920	0.054	0.013	0.028
K-5	Recall	17	0.676	0.973	0.916	0.871	0.956	0.085	0.062	0.885	0.088	0.021	0.045
K-6	F1	17	0.842	1.000	0.961	0.946	0.987	0.042	0.039	0.950	0.047	0.011	0.024
K-6	Precision	17	0.800	1.000	0.966	0.951	0.984	0.033	0.026	0.956	0.049	0.012	0.025
K-6	Recall	17	0.831	1.000	0.962	0.937	0.991	0.053	0.043	0.946	0.056	0.013	0.029
K-7	F1	17	0.750	0.982	0.939	0.895	0.962	0.067	0.053	0.912	0.073	0.018	0.037
K-7	Precision	17	0.718	1.000	0.936	0.893	0.965	0.071	0.063	0.919	0.068	0.017	0.035
K-7	Recall	17	0.667	0.980	0.934	0.897	0.970	0.073	0.056	0.909	0.089	0.021	0.046
K-8	F1	17	0.762	0.992	0.934	0.909	0.970	0.061	0.049	0.925	0.063	0.015	0.032
K-8	Precision	17	0.667	1.000	0.938	0.914	0.964	0.050	0.039	0.920	0.080	0.019	0.041
K-8	Recall	17	0.803	1.000	0.934	0.904	0.984	0.080	0.066	0.934	0.059	0.014	0.030
K-9	F1	17	0.693	0.993	0.935	0.892	0.978	0.086	0.064	0.913	0.081	0.020	0.042
K-9	Precision	17	0.727	1.000	0.952	0.918	0.981	0.063	0.046	0.928	0.076	0.019	0.039
K-9	Recall	17	0.627	1.000	0.944	0.889	0.980	0.091	0.059	0.905	0.109	0.026	0.056
K-10	F1	17	0.571	0.975	0.919	0.909	0.957	0.048	0.055	0.895	0.105	0.025	0.054
K-10	Precision	17	0.764	1.000	0.951	0.912	0.959	0.047	0.022	0.917	0.071	0.017	0.036
K-10	Recall	17	0.444	1.000	0.918	0.854	0.970	0.116	0.083	0.881	0.138	0.034	0.071

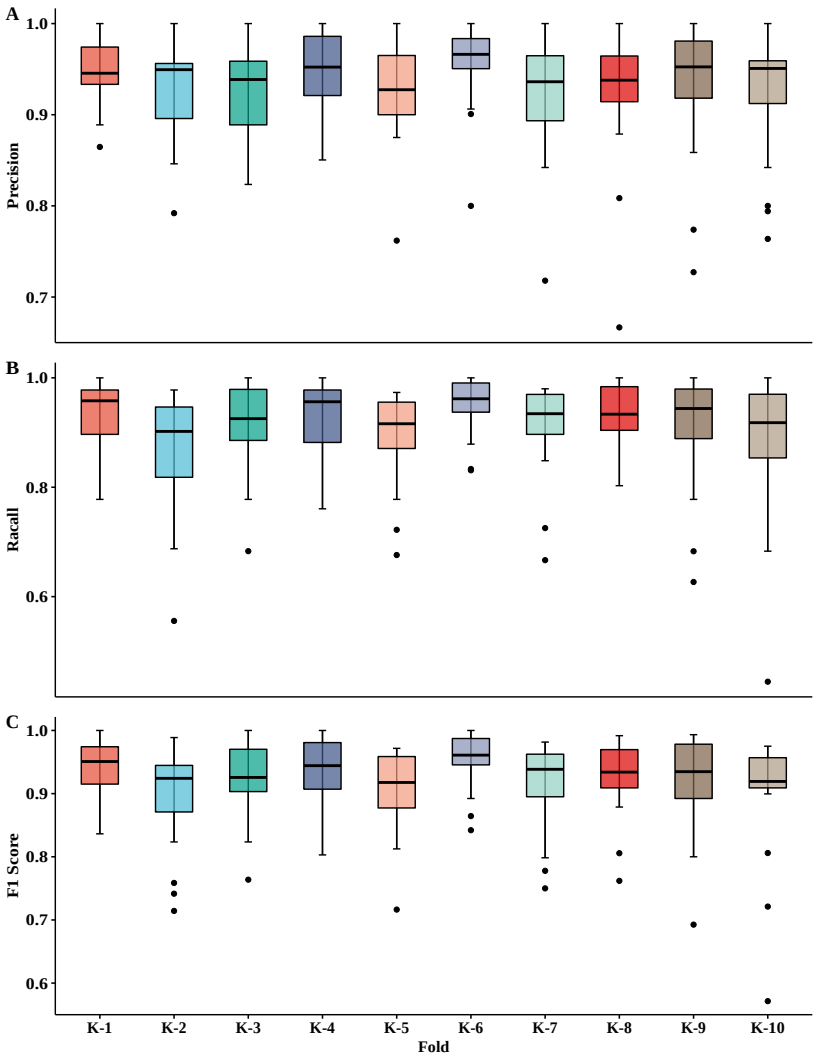

Figure 4.5: Boxplot of k-Fold crossvalidation analysis shows the distribution and possible difference between folds, which is not observed in any scores. Most of the box has outlier points and the medians are mainly in the range of 0.8 to 1.0, which is a good score for these models. A. Shows the Precision results by fold. B. Shows the Recall results by fold. C. Show the F1 score results by fold. Whiskers represent the associated error.

4.4 Minimum viable product for annotation and extraction of legal entities in the Chilean environmental field

A product to be obtained is a web application that allows visualising a possible use of the custom-trained model. For the above, Streamlit was used to develop a simple but functional application that shows how the model can recognise entities, label them and extract the sentences associated with them.

Thus, a hybrid strategy extracted the sentences associated with the recognised entities. This is because the trained model does not have the `parser` component, which is the one in charge of obtaining the grammatical and syntactic relations between the words of a sentence. Given that, the long pre-trained model of `spaCy` from which the parser component was loaded was used. However, the other components were left out. Finally, from the custom-trained model, the `tok2vec` and `ner` components (tokeniser and entity recognition) were loaded, indicating that each one should receive a copy and listen of the tokeniser with which they had been trained. With that, a pipeline was built, leaving the following components: `tok2vec`, `ner`, and `parser`.

Figure 4.6 shows a screenshot of the application running locally. The functions with which the application was provided are:

- The annotate of an environmental legal text where the application brings a text by default (Figure 4.7), with a box to pass to the web application unknown text.
- At the same time, it shows the `TAGs` that annotate the model (Figure 4.8) and the pipeline's structure (Figure 4.9).
- Also, it evaluates the model's performance with the unknown text; for this, a confusion matrix analysis is performed (Figure 4.10), and a summary table with the scores obtained is provided (Figure 4.11). It should be noted that these scores are generally lower than those presented in section 4.3.2. However, there is a discrepancy between the annotation of the text that is delivered to the application and the performance evaluation results. This fact will be discussed in the discussion chapter.

- Regarding the extraction of entities with their related sentences (Figure 4.12), two methods are shown where one is more straightforward than the other, which deliver different results, in some cases, one being more explanatory than the other regarding the context in which the tagged entity is found (Figures 4.13 and 4.14).

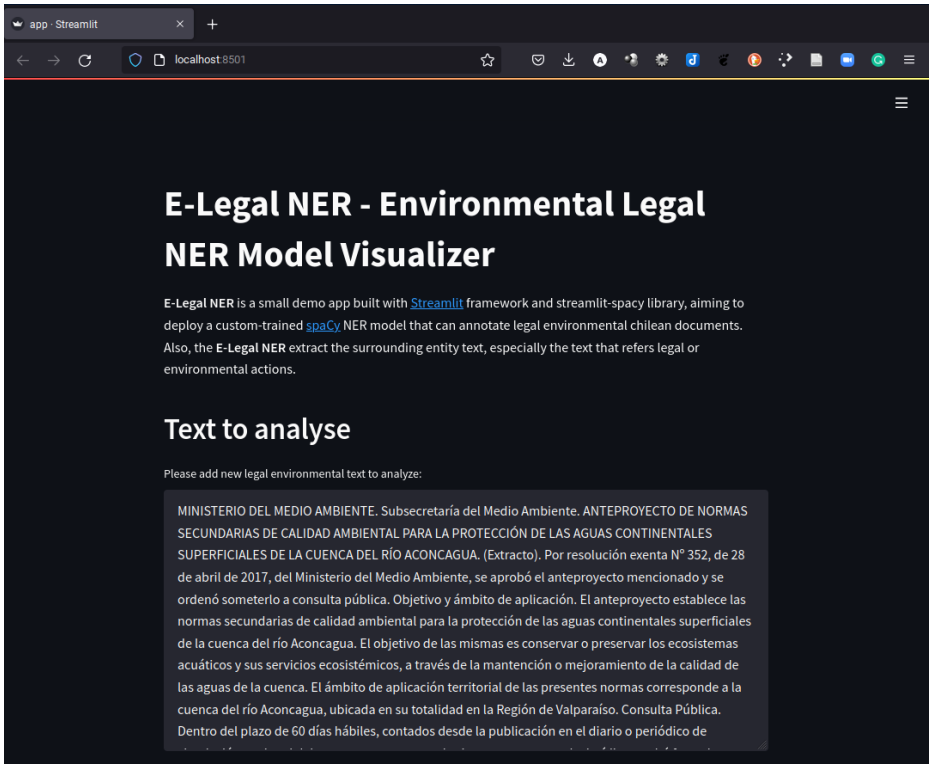

Figure 4.6: Web application developed with Streamlit running locally.

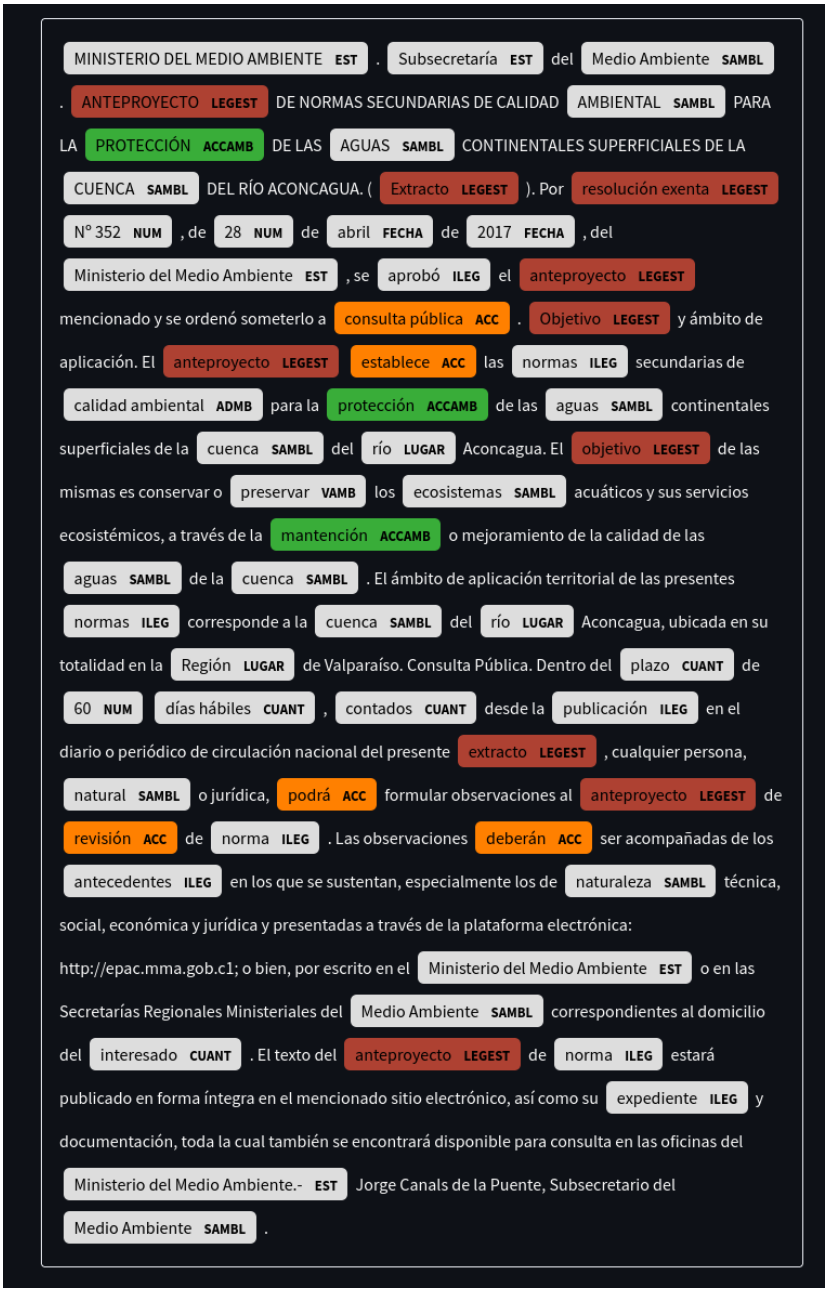

Figure 4.7: Presentation of the annotated text passed to the web application. Colours can be manipulated at will.

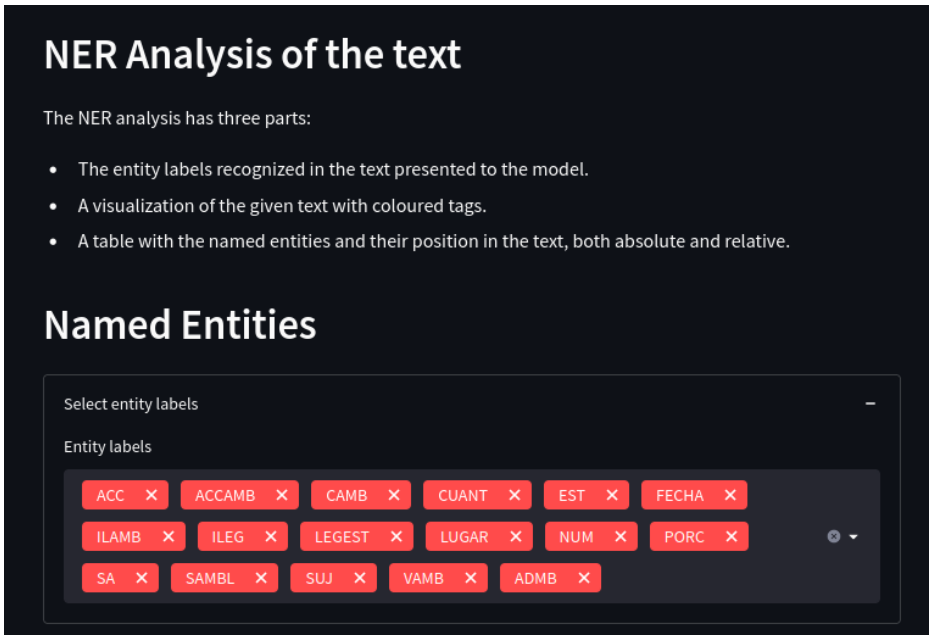

Figure 4.8: Box showing the labels that the model recognise by default. This TAGs can be removed from the analysis.

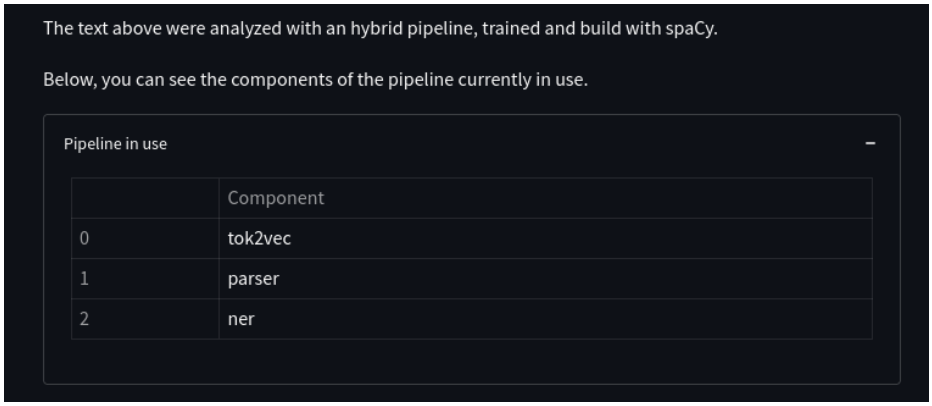

Figure 4.9: Table showing the pipeline's components.

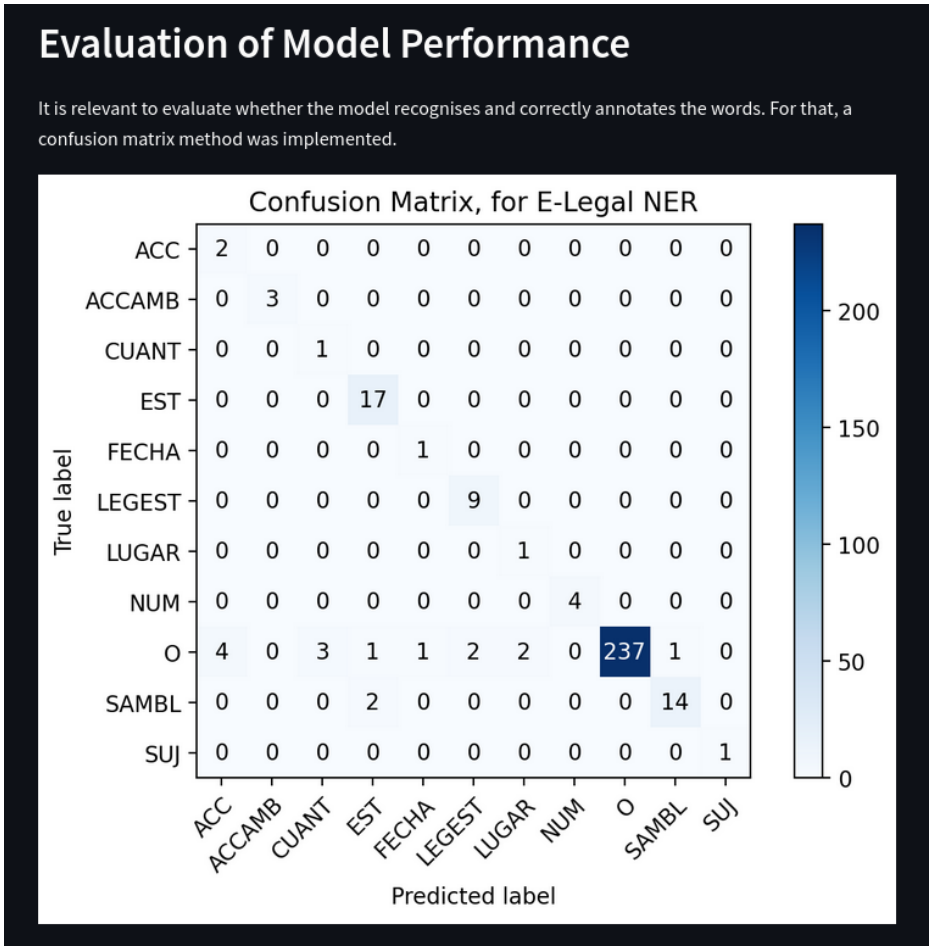

Figure 4.10: The confusion matrix provided by the application evaluates the model performance. It can be noted that the model suffers some confusion, especially with the `TAGs ACC` and `CUANT`. True and predicted labels match in the matrix diagonal.

Resume table with the scores over the tested text:

	Precision	Recall	F1
EST	0.8000	0.8000	0.8000
SAMBL	0.9231	0.9231	0.9231
LEGEST	0.8000	0.8889	0.8421
ACCAMB	1.0000	0.7500	0.8571
NUM	1.0000	1.0000	1.0000
FECHA	0.5000	1.0000	0.6667
ILEG	0.0000	0.0000	0.0000
ACC	0.4000	1.0000	0.5714
ADMB	0.0000	0.0000	0.0000
LUGAR	0.3333	1.0000	0.5000
VAMB	0.0000	0.0000	0.0000
CUANT	0.3333	1.0000	0.5000
SUJ	1.0000	1.0000	1.0000

Figure 4.11: Table with the model Precision, Recall and F1 scores for unknown text.

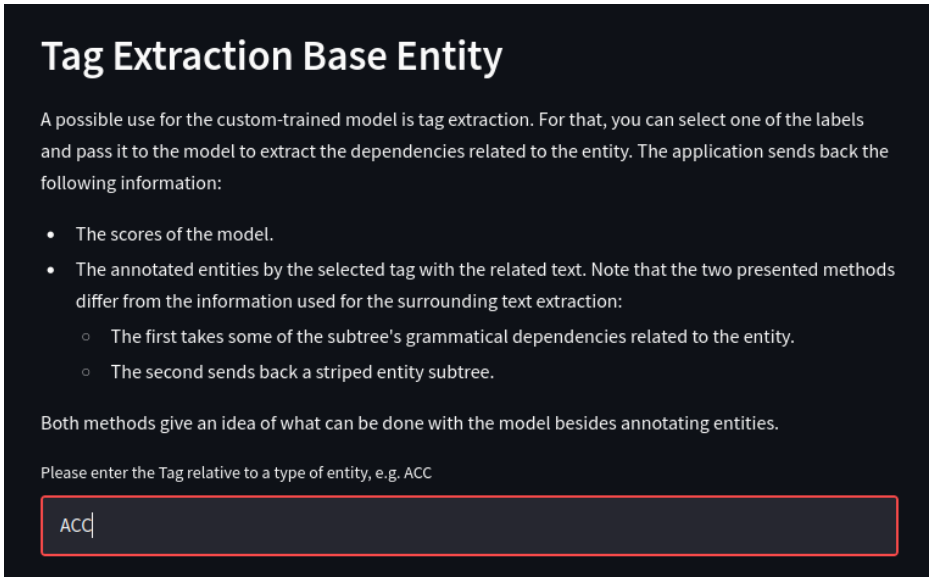

Figure 4.12: `TAG`s extraction box, where the user can ask the application which `TAG` to extract.

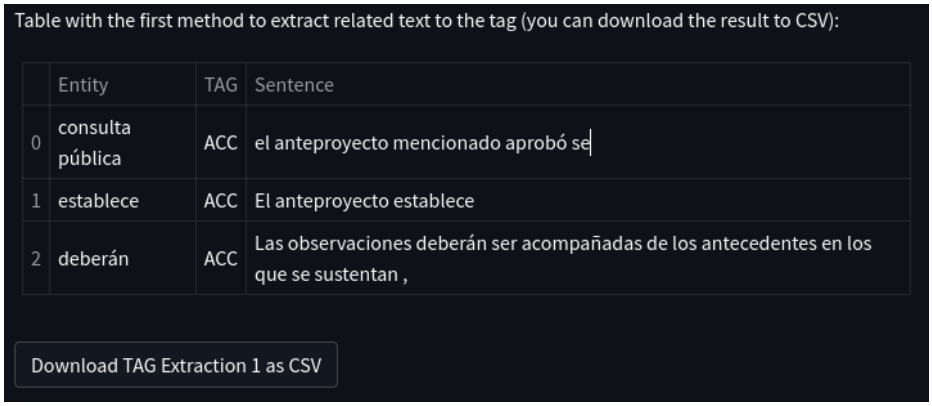

Figure 4.13: Tabulated data retrieved from the application with the entity, the `TAG` and the related sentence for the first method proposed.

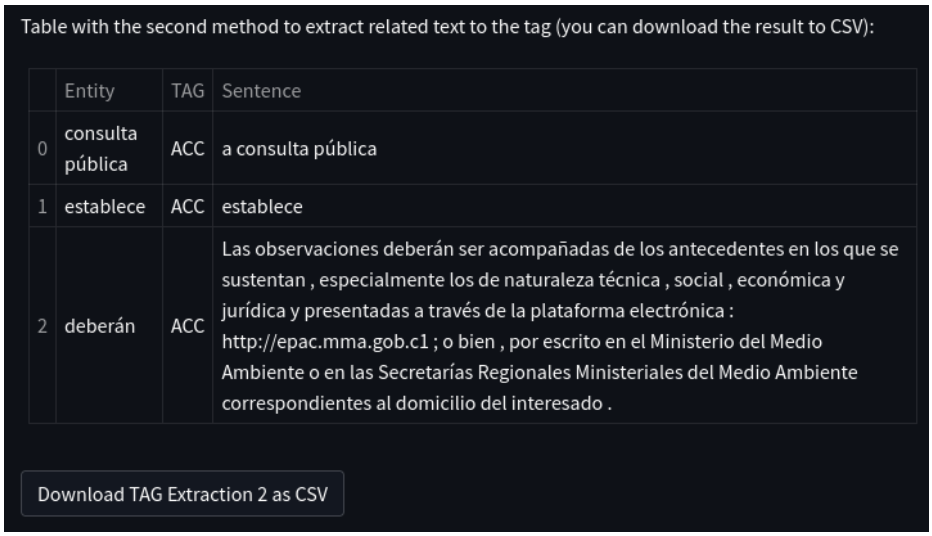

Figure 4.14: Tabulated data retrieved from the application with the entity, the `TAG` and the related sentence for the second method proposed.

Chapter 5

Discussion

There are two general things to discuss until complete the whole process of work in a Natural Language Processing project. First, in the book "Human-in-the-loop Machine Learning" from Monarch (2021), it is proposed that 90% of the machine learning applications today are powered by supervised machine learning. That implies that humans spend thousands of hours telling or annotating how to interpret the information used in training different kinds of models. Construct the necessary training data and then iterate over the process to fine-tune this data, entailing a human in the epicentre of any machine learning that is not settled in a self-supervised learning mechanism. That was the same experience in the first part of this project. Not only is the quantity of the data relevant to training a specific Named Entity Recognition (NER) model, but it is also how this data is collected, annotated, and labels selected, among several small decisions that a human has and needs to make. This teaches that humans' role in the entire machine learning process should never be underestimated and that each project requires special human capital training since it will notably influence subsequent results through biases introduced in the training data.

The second general thing to discuss relates to this kind of project requiring advice from experts in the field to achieve, transforming it into a multidisciplinary or even an interdisciplinary initiative. In the case of this thesis, the author had previous knowledge in the environmental field, which includes the review of laws, decrees and norms that are in use in Chile. That knowledge facilitates selecting the words and phrases that pass to engross the ontology necessary to train a specific custom NER model.

Specifically, using `spaCy` in production environments is relatively straightforward if the pre-trained models provided as Python packages will be used, and it does not imply any particular difficulties. Nevertheless, for specific requirements in the NLP field, for example, NER in Spanish and the relatively broad options present in the `spaCy` framework will challenge the overall strategy. Moreover, an experience that arose from work done, any project of this nature needs to be carefully planned and requires the experience of professionals in the field that knows the tools involved. However, that does not imply that the use of `spaCy` in production can be a problem but presents challenges in planning the training to achieve the desired results.

The broad options that `spaCy` has gives room for constant improvement to the

NER model. Some of these improvements relate to the fine-tuning of words used in each label, implying an iteration of the whole training process. In contrast, others are related to the word embeddings built with Gensim, which always welcome more data, implying a broad collection of it and so improves the number of words with vectors, only 45% in training had vectors.

The above opens the option to concatenate in a pipeline software all the training process, which was not done in this thesis. That will partially automate the process and delivers improved versions of the custom-trained NER model for use in production.

More in-depth from the different model versions obtained during the work, the overall results are pretty good compared to published results from `spaCy` (see Table 5.1). For example, the F1 score for the large model reports a value of 89.77. In contrast, version 7.0.9 reported in this thesis has a value for F1 of 94.88 (see section 4.3.2). However, when the custom-trained model processes unknown environmental legal text, its accuracy evaluations range between 80.0 to 90.0 for the same score. From the above issue, some overfitting in the training data could exist. Notwithstanding, the results are still quite good.

In the web application functions, there is a discrepancy between the annotation of the text that is delivered to the application and the performance evaluation results. The above can be due to two things:

- The pipeline used in the annotation is not the same as the one used for performance evaluation.
- There may be an error in transforming the annotated text to the `BILUO` format since, although the entities are recognized, the span and the labelling of some entities may be lost, generating discrepancies between the two pipelines. Specifically, the `TAG ACC` is reported by the confusion matrix with two True annotations, but the pipeline for annotation reports three. In addition, looking at the console output, the entity *consulta pública* was not labelled in the text annotation. That fact should be the problem behind the discrepancy between pipelines.
- The code used for performance evaluation can be reviewed and improved, and

split the visualizing function from the `spacy_streamlit` to obtain the annotated text directly and use that data with the performance evaluation. However, to date, the text is again annotated for the evaluation, so the contrasted data is not precisely the same.

Table 5.1: spaCy's large pre-trained model (version 3.4.0) accuracy evaluation for Spanish language. Note, that the evaluation is over all the components in the model pipeline (parameter). Source: https://github.com/explosion/spacy-models/releases/tag/es_core_news_lg-3.4.0

Parameter	Parameter explanaiton	Value
TOKEN_ACC	Tokenization	100.00
TOKEN_P		99.89
TOKEN_R		99.95
TOKEN_F		99.92
POS_ACC	Part-of-speech tags (coarse grained tags, Token.pos)	98.52
MORPH_ACC	Morphological analysis	98.23
MORPH_MICRO_P		99.54
MORPH_MICRO_R		99.03
MORPH_MICRO_F		99.29
SENTS_P	Sentence segmentation (precision)	97.19
SENTS_R	Sentence segmentation (recall)	98.65
SENTS_F	Sentence segmentation (F-score)	97.91
DEP_UAS	Unlabeled dependencies	91.30
DEP_LAS	Labeled dependencies	88.19
TAG_ACC	Part-of-speech tags (fine grained tags, Token.tag)	96.15
LEMMA_ACC	Lemmatization	96.56
ENTS_P	Named entities (precision)	89.76
ENTS_R	Named entities (recall)	89.79
ENTS_F	Named entities (F-score)	89.77

Training models with `spaCy` as it was manifested have its difficulties but is a framework that allows put in production quickly the necessary components for online products that need text processing. The framework also has the flexibility to implement simple to complex under-the-hood workers, with specific custom-trained models that can work in the same pipeline.

Chapter 6

Conclussions

It can be concluded that this thesis work reaches the proposed general and specific objectives and allows the following few conclusions that answer the suggested research question of this thesis:

- First allowed to afford a natural language processing problem and internalize its challenges.
- It has the potential to put in value a custom-trained model with a specific framework (`spaCy`) in a language different from English that can be used in different applications.
- The accuracy scores of the custom-trained model were in the range expected for these models.
- It proves that the `spaCy` framework answer the concerns around its utility for production use.
- Finally, the resultant tool could be extended for further use in different kinds of applications.

In addition, it is necessary to point out that the experience gained is not restricted to NLP and can be applied in various fields due to the technology underneath which NLP lives are in the same line with other fields like, for example, computer vision problems.

References

Alpaydin, E. (2010). *Introduction to machine learning*. MIT Press. Retrieved from https://books.google.cl/books?id=4j9GAQAAIAAJ

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., … Farhan, L. (2021, March). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, *8*(1), 53. Retrieved from https://doi.org/10.1186/s40537-021-00444-8 doi: 10.1186/s40537-021-00444-8

Angelidis, I., Chalkidis, I., & Koubarakis, M. (2018). Named entity recognition, linking and generation for greek legislation. In *Jurix* (pp. 1–10).

Barratt, S., & Sharma, R. (2018, May). *Optimizing for generalization in machine learning with cross-validation gradients*. Retrieved from http://arxiv.org/abs/1805.07072v1;http://arxiv.org/pdf/1805.07072v1

Barriere, V., & Fouret, A. (2019). May i check again?–a simple but efficient way to generate and use contextual dictionaries for named entity recognition. application to french legal texts. *arXiv preprint arXiv:1909.03453*.

Becker, C., Hahn, N., He, B., Jabbar, H., Plesiak, M., Szabo, V., … Wagner, J. (2020, September). *Modern approaches in natural language processing*.

Benoit, K., Muhr, D., & Watanabe, K. (2021). stopwords: Multilingual stopword lists [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=stopwords (R package version 2.3)

Bishop, C. (2006). *Pattern recognition and machine learning*. Springer. Retrieved from https://books.google.cl/books?id=qWPwnQEACAAJ

Bloom, B. H. (1970, jul). Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, *13*(7), 422–426. Retrieved from https://doi.org/10.1145/362686.362692 doi: 10.1145/362686.362692

Bommarito II, M. J., Katz, D. M., & Detterman, E. M. (2021). Lexnlp: Natural language processing and information extraction for legal and regulatory texts. In *Research handbook on big data law*. Edward Elgar Publishing.

Breiman, L. (2001). Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, *16*(3), 199 – 231. Retrieved from https://doi.org/10.1214/ss/1009213726 doi: 10.1214/ss/1009213726

Brooks, R. (2022, January). *How Claude Shannon Helped Kick-start Machine*

Learning. Retrieved 2022-02-02, from https://spectrum.ieee.org/claude-shannon-information-theory

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). *Learning phrase representations using rnn encoder-decoder for statistical machine translation*. arXiv. Retrieved from https://arxiv.org/abs/1406.1078 doi: 10.48550/ARXIV.1406.1078

Chollet, F. (2021). *Deep learning with python* (Second Edition ed.). Manning Publications.

Costumero, R., Sánchez, J., García-Pedrero, Á., Rivera, D., Lillo, M., Gonzalo-Martín, C., & Menasalvas, E. (2017). Geography of legal water disputes in chile. *Journal of Maps*, *13*(1), 7–13. Retrieved from https://doi.org/10.1080/17445647.2016.1252803 doi: 10.1080/17445647.2016.1252803

Crawford, K., & Joler, V. (2018, sep 7). Anatomy of an ai system: The amazon echo as an anatomical map of human labor, data and planetary resources. *AI Now Institute and Share Lab*. https://anatomyof.ai.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q. V., Salakhutdinov, R., et al. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR, abs/1901.02860*. Retrieved from http://arxiv.org/abs/1901.02860

Dean, J. (2014). *Big data, data mining, and machine learning: Value creation for business leaders and practitioners*. Wiley. Retrieved from https://books.google.cl/books?id=pNp1AwAAQBAJ

Devlin, J., Chang, M.-W., Lee, K., Toutanova, K., et al. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dhar, P. (2020). The carbon impact of artificial intelligence. *Nature Machine Intelligence*, *2*, 423–425. doi: 10.1038/s42256-020-0219-9

Dozier, C., Kondadadi, R., Light, M., Vachher, A., Veeramachaneni, S., & Wudali, R. (2010). Named entity recognition and resolution in legal text. In *Semantic processing of legal texts* (pp. 27–43). Springer.

Dyer, C., Ballesteros, M., Ling, W., Matthews, A., Smith, N. A., et al. (2015, July). Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)* (pp. 334–343). Beijing, China: Association for Computational Linguistics. Retrieved from https://aclanthology.org/P15-1033 doi: 10.3115/v1/P15-1033

Firebanks-Quevedo, D., Planas, J., Buckingham, K., Taylor, C., Silva, D., Naydenova, G., & Zamora-Cristales, R. (2022). Using machine learning to identify incentives in forestry policy: Towards a new paradigm in policy analysis. *Forest Policy and Economics*, *134*, 102624. Retrieved from https://www.sciencedirect.com/science/article/pii/S1389934121002306 doi: 10.1016/j.forpol.2021.102624

Glaser, I., Waltl, B., & Matthes, F. (2018). Named entity recognition, extraction, and linking in german legal contracts. In *Iris: Internationales rechtsinformatik symposium* (pp. 325–334).

Goldberg, Y. (2017). *Neural network methods for natural language processing*. Morgan & Claypool.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. (http://www.deeplearningbook.org)

Google. (2017). *A history of machine learning*. Retrieved 2022-02-02, from https://cloud.withgoogle.com/build/data-analytics/explore-history-machine-learning/

Graves, A. (2013, 08). *Generating sequences with recurrent neural networks*. arXiv. Retrieved from https://arxiv.org/abs/1308.0850 doi: 10.48550/ARXIV.1308.0850

Gutiérrez-Fandiño, A., Armengol-Estapé, J., Gonzalez-Agirre, A., & Villegas, M. (2021). Spanish legalese language model and corpora. *arXiv preprint arXiv:2110.12201*.

Herrera, M., Candia, C., Rivera, D., Aitken, D., Brieba, D., Boettiger, C., … Godoy-Faúndez, A. (2019). Understanding water disputes in chile with text and data mining tools. *Water International*, *44*(3), 302–320. Retrieved from https://doi.org/10.1080/02508060.2019.1599774 doi: 10.1080/02508060.2019.1599774

Hochreiter, S., & Schmidhuber, J. (1997, 11). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780. Retrieved from https://doi.org/10.1162/neco.1997.9.8.1735 doi: 10.1162/neco.1997.9.8.1735

Honnibal, M. (2018). *Embed, encode, attend, predict*. Retrieved from https://raw.githubusercontent.com/explosion/talks/master/2018-04-12_Embed-Encode-Attend-Predict.pdf

Honnibal, M., & Johnson, M. (2015, September). An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 conference on empirical methods in natural language processing* (pp. 1373–1378). Lisbon, Portugal: Association for Computational Linguistics. Retrieved from https://aclanthology.org/D15-1162 doi: 10.18653/v1/D15-1162

Honnibal, M., & Montani, I. (2017). *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing*. (To appear)

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An introduction to statistical learning: with applications in r* (Second ed.). Springer US. Retrieved from https://books.google.cl/books?id=5dQ6EAAAQBAJ

Kaelbling, L. P., Littman, M. L., Moore, A. W., et al. (1996, May). Reinforcement learning: A survey. , 237–285. Retrieved from https://arxiv.org/abs/cs/9605103 doi: 10.48550/ARXIV.CS/9605103

Kar, P., K, B., Sriperumbudur, Jain, P., & Karnick, H. C. (2013, May). On the generalization ability of online learning algorithms for pairwise loss functions. , Jour-

nal of Machine Learning Research. Retrieved from https://arxiv.org/abs/1305.2505 doi: 10.48550/ARXIV.1305.2505

Kim, Y. (2014, September). *Convolutional neural networks for sentence classification*. Retrieved from http://arxiv.org/abs/1408.5882v2;http://arxiv.org/pdf/1408.5882v2

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., … Willing, C. (2016). Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides & B. Schmidt (Eds.), *Positioning and power in academic publishing: Players, agents and agendas* (pp. 87–90).

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., Dyer, C., et al. (2016, June). Neural architectures for named entity recognition. In *Proceedings of the 2016 conference of the north American chapter of the association for computational linguistics: Human language technologies* (pp. 260–270). San Diego, California: Association for Computational Linguistics. Retrieved from https://aclanthology.org/N16-1030 doi: 10.18653/v1/N16-1030

Langley, P. (2011). The changing science of machine learning. *Machine Learning*(82), 275–279. doi: 10.1007/s10994-011-5242-y

Leitner, E., Rehm, G., & Moreno-Schneider, J. (2019). Fine-grained named entity recognition in legal documents. In *International conference on semantic systems* (pp. 272–287).

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.*, *5*, 115–133. Retrieved from http://cds.cern.ch/record/427611

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, *2014*(239), 2.

Mikolov, T., Chen, K., Corrado, G., Dean, J., et al. (2013). *Efficient estimation of word representations in vector space*. arXiv. Retrieved from https://arxiv.org/abs/1301.3781 doi: 10.48550/ARXIV.1301.3781

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J., et al. (2013). *Distributed representations of words and phrases and their compositionality*. arXiv. Retrieved from https://arxiv.org/abs/1310.4546 doi: 10.48550/ARXIV.1310.4546

Mitchell, T. (1997). *Machine learning*. McGraw-Hill. Retrieved from https://books.google.cl/books?id=EoYBngEACAAJ

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2012). *Foundations of machine learning*. MIT Press. Retrieved from https://books.google.cl/books?id=maz6AQAAQBAJ

Monarch, R. M. (2021). *Human-in-the-loop machine learning* (First Edition ed.). Shelter Island, NY 11964: Manning Publications Co.

Moreno-Schneider, J., Rehm, G., Montiel-Ponsoda, E., Rodriguez-Doncel, V., Revenko, A., Karampatakis, S., … Maganza, F. (2020). Orchestrating nlp services for the legal domain. *arXiv preprint arXiv:2003.12900*.

Mullen, L. A., Benoit, K., Keyes, O., Selivanov, D., & Arnold, J. (2018). Fast, consistent

tokenization of natural language text. *Journal of Open Source Software*, *3*, 655. Retrieved from https://doi.org/10.21105/joss.00655 doi: 10.21105/joss.00655

Muller, B. (2022, March). *BERT 101 - State Of The Art NLP Model Explained*. Retrieved 2022-07-24, from https://huggingface.co/blog/bert-101

Mumcuoğlu, E., Öztürk, C. E., Ozaktas, H. M., & Koç, A. (2021). Natural language processing in law: Prediction of outcomes in the higher courts of turkey. *Information Processing & Management*, *58*(5), 102684.

Nivre, J. (2004). Incrementality in deterministic dependency parsing. In *Proceedings of the workshop on incremental parsing: Bringing engineering and cognition together* (pp. 50–57). USA: Association for Computational Linguistics.

Pascanu, R., Gulcehre, C., Cho, K., & Bengio, Y. (2014, 12). *How to construct deep recurrent neural networks*. Retrieved from https://arxiv.org/abs/1312.6026v5

Pavlyshenko, B. (2016, December). *Machine learning, linear and bayesian models for logistic regression in failure detection problems*. Retrieved from http://arxiv.org/abs/1612.05740v1;http://arxiv.org/pdf/1612.05740v1

Perner, P. (2011). *Machine learning and data mining in pattern recognition: 7th international conference, mldm 2011, new york, ny, usa, august 30-september 3, 2011proceedings*. Springer Berlin Heidelberg. Retrieved from https://books.google.cl/books?id=VIaNhcb91s4C

Qi, X., Silvestrov, S., & Nazir, T. (2016, May). *Linear classification of data with support vector machines and generalized support vector machines*. Retrieved from http://arxiv.org/abs/1606.05664v1;http://arxiv.org/pdf/1606.05664v1 doi: 10.1063/1.4972718

R Core Team. (2021). R: A language and environment for statistical computing [Computer software manual]. Vienna, Austria. Retrieved from https://www.R-project.org/

Rehurek, R., & Sojka, P. (2011). Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, *3*(2).

Russell, S., & Norvig, P. (2021). *Artificial intelligence: A modern approach, global edition*. Pearson Education. Retrieved from https://books.google.cl/books?id=cb0qEAAAQBAJ

Russell, S., Russell, S., Norvig, P., & Davis, E. (2010). *Artificial intelligence: A modern approach*. Prentice Hall. Retrieved from https://books.google.cl/books?id=8jZBksh-bUMC

S., S., Nowozin, S., & Wright, S. (2012). *Optimization for machine learning*. MIT Press. Retrieved from https://books.google.cl/books?id=JPQx7s2L1A8C

Samarawickrama, C., de Almeida, M., Perera, A. S., de Silva, N., & Ratnayaka, G. (2021). Identifying legal party members from legal opinion texts using natural language processing. *EasyChair, Tech. Rep*.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers.

IBM Journal of Research and Development, *3*(3), 210–229. doi: 10.1147/rd.33.0210

Sarle, W. S. (1994). Neural networks and statistical models. In *Proceedings of the nineteenth annual sas users group international conference,* (pp. 1538–1550). SAS Institute.

Schuster, M., & Paliwal, K. (1997, Nov). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673–2681. doi: 10.1109/78.650093

Sierra, G., et al. (2018). Event extraction from legal documents in spanish. In *1st workshop on language resources and technologies for the legal knowledge graph* (p. 36).

Socher, R., Lin, C. C.-Y., Ng, A. Y., Manning, C. D., et al. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on international conference on machine learning* (pp. 129–136). Madison, WI, USA: Omnipress.

Suvrit, S. (2016, May). *Directional statistics in machine learning: a brief review.* Retrieved from http://arxiv.org/abs/1605.00316v1;http://arxiv.org/pdf/1605.00316v1

Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual.* Scotts Valley, CA: CreateSpace.

Vasilev, I. (2019). *Advanced deep learning with python: Design and implement advanced next-generation ai solutions using tensorflow and pytorch.* Packt Publishing. Retrieved from https://books.google.cl/books?id=TxbEDwAAQBAJ

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … others (2017, December). *Attention is all you need.* Retrieved from http://arxiv.org/abs/1706.03762v5;http://arxiv.org/pdf/1706.03762v5

von Luxburg, U., & Schoelkopf, B. (2008, October). *Statistical learning theory: Models, concepts, and results.* Retrieved from http://arxiv.org/abs/0810.4752v1;http://arxiv.org/pdf/0810.4752v1

Wang, Z., Wu, Y., Lei, P., & Peng, C. (2020). Named entity recognition method of brazilian legal text based on pre-training model. In *Journal of physics: Conference series* (Vol. 1550, p. 032149).

Zhong, H., Xiao, C., Tu, C., Zhang, T., Liu, Z., & Sun, M. (2020). How does nlp benefit legal system: A summary of legal artificial intelligence. *arXiv preprint arXiv:2004.12158*.

List of acronyms

* AI: Artificial Intelligence
* ML: Machine Learning
* NLP: Natural Language Processing
* NN: Neural Networks
* ANN: Artificial Neural Networks
* CNN: Convolutional Neural Networks
* NER: Named Entity Recognition

Appendices

A Appendix: In-house system information

Client:

```
Operating System: Ubuntu 20.04.3 LTS x86_64
Kernel Version: 5.13.0-22-generic
Packages: 3595 (dpkg), 44 (flatpak), 24 (snap)
Shell: bash 5.0.17
KDE Plasma Version: 5.18.7
KDE Frameworks Version: 5.68.0
Qt Version: 5.12.8
CPU: AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx (8) @ 2.100GHz
GPU: AMD ATI Picasso
Memory: 29.3 GiB
```

Server:

```
OS: Ubuntu 20.04.3 LTS x86_64
Kernel: 5.13.0-27-generic
Packages: 1952 (dpkg), 9 (snap)
Shell: bash 5.0.17
Terminal: /dev/pts/0
CPU: AMD Ryzen Threadripper 1900X (16) @ 3.800GHz
GPU: AMD ATI Radeon Pro WX 5100
Memory: 64.2GiB
```

B Appendix: Preparing training data

Followed sequence for processing data

The procedure or algorithm used to prepare the data to train a spaCy model to track the following structure:

1. Import the requisite library
2. Build upon the `spaCy` Large Model
3. Load text document and key entities/TAGs files to build patterns (keyword dictionary).
4. Check text length
5. Import and apply `PhraseMatcher` function to keyword dictionary.
6. Check if the matcher is working.
7. Build the Corpus with sentences to annotate.
8. Check the Corpus length.
9. Build the patterns.
10. Creating an entity ruler with a blank *spaCy* model and the patterns.
11. Saving a `JSON` file with the training data and visualizing.
12. Transforming the data in `JSON` format to `DocBin` (`spaCy` format).

Using blank model with patterns

As part of the experimention a second iteration will be performed to prepare the training data.

The `words.csv` file was revised and improved, and six new TAGs ACCAMB, NUM, PLACE, PORC were added. The new TAGs are expected to recognize the different entities after training.

These new training data will be used in the `Experiment 7`, moving to version `7.0.9` of the model. The training process of this model includes the pre-training and the use of the weight vectors trained with `Gensim`.

For training, all available legal text data will be used, which is found in the file `texto_all_ldr.txt`. The file with legal text used previously was `decrees.txt`.

[1]:
```
#Import the requisite library
import spacy
import pandas as pd
import json
import random

#Build upon the spaCy Large Model
nlp = spacy.load("es_core_news_lg")
```

```
# the max length is increased. since it throws an error when processing the␣
 ↪entire text for training.
nlp.max_length = 1500000
```

[2]:
```
# File upload with decrees

with open ("./00-Data/texto_all_ldr.txt", "r") as f:
    text = f.read()

# File upload with keywords to build patterns

keyword_dict = pd.read_csv("./00-Data/words.csv") # se usa la version␣
 ↪actualizada del archivo.

# text length
len(text)
```

[2]:
```
1416885
```

[3]:
```
# The keywords are extracted from the dictionary and added to the matcher␣
 ↪together with the label

from spacy.matcher import PhraseMatcher

ACC_words = [nlp(text) for text in keyword_dict['ACC'].dropna(axis = 0)]
ADMB_words = [nlp(text) for text in keyword_dict['ADMB'].dropna(axis = 0)]
CAMB_words = [nlp(text) for text in keyword_dict['CAMB'].dropna(axis = 0)]
CUANT_words = [nlp(text) for text in keyword_dict['CUANT'].dropna(axis = 0)]
ILAMB_words = [nlp(text) for text in keyword_dict['ILAMB'].dropna(axis = 0)]
ILEG_words = [nlp(text) for text in keyword_dict['ILEG'].dropna(axis = 0)]
SA_words = [nlp(text) for text in keyword_dict['SA'].dropna(axis = 0)]
SAMBL_words = [nlp(text) for text in keyword_dict['SAMBL'].dropna(axis = 0)]
SUJ_words = [nlp(text) for text in keyword_dict['SUJ'].dropna(axis = 0)]
VAMB_words = [nlp(text) for text in keyword_dict['VAMB'].dropna(axis = 0)]
EST_words = [nlp(text) for text in keyword_dict['EST'].dropna(axis = 0)]
LEGEST_words = [nlp(text) for text in keyword_dict['LEGEST'].dropna(axis = 0)]
## new TAGs
NUM_words = [nlp(text) for text in keyword_dict['NUM'].dropna(axis = 0)]
FECHA_words = [nlp(text) for text in keyword_dict['FECHA'].dropna(axis = 0)]
LUGAR_words = [nlp(text) for text in keyword_dict['LUGAR'].dropna(axis = 0)]
ACCAMB_words = [nlp(text) for text in keyword_dict['ACCAMB'].dropna(axis = 0)]
PORC_words = [nlp(text) for text in keyword_dict['PORC'].dropna(axis = 0)]
#ORG_words = [nlp(text) for text in keyword_dict['ORG'].dropna(axis = 0)]

matcher = PhraseMatcher(nlp.vocab)
matcher.add('ACC', None, *ACC_words)
matcher.add('ADMB', None, *ADMB_words)
matcher.add('CAMB', None, *CAMB_words)
matcher.add('CUANT', None, *CUANT_words)
matcher.add('ILAMB', None, *ILAMB_words)
```

```
matcher.add('ILEG', None, *ILEG_words)
matcher.add('SA', None, *SA_words)
matcher.add('SAMBL', None, *SAMBL_words)
matcher.add('SUJ', None, *SUJ_words)
matcher.add('VAMB', None, *VAMB_words)
matcher.add('EST', None, *EST_words)
matcher.add('LEGEST', None, *LEGEST_words)
## new TAGs
matcher.add('NUM', None, *NUM_words)
matcher.add('FECHA', None, *FECHA_words)
matcher.add('LUGAR', None, *LUGAR_words)
matcher.add('ACCAMB', None, *ACCAMB_words)
matcher.add('PORC', None, *PORC_words)
#matcher.add('ORG', None, *ORG_words)
```

[4]:
```
# The PhraseMatcher result for the TAG ACC is displayed
ACCAMB_words[0:15]
```

[4]:
```
[Emisión,
 Incidente,
 Daño,
 Impacto,
 Perdida,
 Superación,
 Cambios,
 Presencia,
 Conservación,
 Protección,
 Alcance,
 Compensación,
 Sobrepasadas,
 Uso,
 Homologación]
```

[5]:
```
corpus = []

doc = nlp(text)
for sent in doc.sents:
    corpus.append(sent.text)

# longitud del corpus (sentencias)
len(corpus)
```

[5]:
```
6593
```

[6]:
```
print(corpus[0:5])
```

```
['LEY NÚM.', '19.300\n\nTITULO I\n\nDisposiciones Generales\n\nArtículo 1°.- El
derecho a vivir en un medio ambiente libre de contaminación, la protección del
medio ambiente, la preservación de la naturaleza y la conservación del
```

patrimonio ambiental se regularán por las disposiciones de esta ley, sin perjuicio de lo que otras normas legales establezcan sobre la materia.', '\n\nArtículo 2°.- Para todos los efectos legales, se entenderá por:\n\na) Biodiversidad o Diversidad Biológica: la variabilidad de los organismos vivos, que forman parte de todos los ecosistemas terrestres y acuáticos.', 'Incluye la diversidad dentro de una misma especie, entre especies y entre ecosistemas;\n\na bis) Biotecnología: se entiende toda aplicación tecnológica que utilice sistemas biológicos y organismos vivos o sus derivados para la creación o modificación de productos o procesos para usos específicos;\n\na ter) Cambio Climático: se entiende un cambio de clima atribuido directa o indirectamente a la actividad humana que altera la composición de la atmósfera mundial y que se suma a la variabilidad natural del clima observada durante períodos de tiempo comparables;\n\nb) Conservación del Patrimonio Ambiental: el uso y aprovechamiento racionales o la reparación, en su caso, de los componentes del medio ambiente, especialmente aquellos propios del país que sean únicos, escasos o representativos, con el objeto de asegurar su permanencia y su capacidad de regeneración;\n\nc) Contaminación: la presencia en el ambiente de sustancias, elementos, energía o combinación de ellos, en concentraciones o concentraciones y permanencia superiores o inferiores, según corresponda, a las establecidas en la legislación vigente;\n\nd) Contaminante: todo elemento, compuesto, sustancia, derivado químico o biológico, energía, radiación, vibración, ruido, luminosidad artificial o una combinación de ellos, cuya presencia en el ambiente, en ciertos niveles, concentraciones o períodos de tiempo, pueda constituir un riesgo a la salud de las personas, a la calidad de vida de la población, a la preservación de la naturaleza o a la conservación del patrimonio ambiental;\n\ne) Daño Ambiental: toda pérdida, disminución, detrimento o menoscabo significativo inferido al medio ambiente o a uno o más de sus componentes;\n\nf) Declaración de Impacto Ambiental: el documento descriptivo de una actividad o proyecto que se pretende realizar, o de las modificaciones que se le introducirán, otorgado bajo juramento por el respectivo titular, cuyo contenido permite al organismo competente evaluar si su impacto ambiental se ajusta a las normas ambientales vigentes;\n\ng) Desarrollo Sustentable: el proceso de mejoramiento sostenido y equitativo de la calidad de vida de las personas, fundado en medidas apropiadas de conservación y protección del medio ambiente, de manera de no comprometer las expectativas de las generaciones futuras;\n\nh) Educación Ambiental: proceso permanente de carácter interdisciplinario, destinado a la formación de una ciudadanía que reconozca valores, aclare conceptos y desarrolle las habilidades y las actitudes necesarias para una convivencia armónica entre seres humanos, su cultura y su medio bio-físico circundante;\n\nh bis) Efecto Sinérgico: aquel que se produce cuando el efecto conjunto de la presencia simultánea de varios agentes supone una incidencia ambiental mayor que el efecto suma de las incidencias individuales contempladas aisladamente;\n\ni) Estudio de Impacto Ambiental: el documento que describe pormenorizadamente las características de un proyecto o actividad que se pretenda llevar a cabo o su modificación.', 'Debe proporcionar antecedentes fundados para la predicción, identificación e interpretación de su impacto ambiental y describir la o las acciones que ejecutará para impedir o minimizar sus efectos significativamente adversos;\n\ni bis).']

[7]:
```
# applying the matcher to the text by sentence and obtaining the patterns
# example of how to build the pattern

#patterns = [
#                {"label": "ACC", "pattern": "ESTABLECE"}
#            ]

patterns = []
matches = matcher(doc)
for match_id, start, end in matches:
    rule_id = nlp.vocab.strings[match_id]  # acquiring the unique ID, i.e. 'LEGEST'
    span = doc[start : end]  # acquiring the range of the match in the document
    patterns.append({"label":rule_id, "pattern":span.text}) # adding by id and span

#keywords = "\n".join(f'{i[0]} {i[1]} ({j})' for i,j in Counter(d).items())
```

[8]:
```
print(patterns[0:10])
```

```
[{'label': 'LEGEST', 'pattern': 'LEY'}, {'label': 'NUM', 'pattern': 'NÚM.'},
{'label': 'NUM', 'pattern': 'NÚM. 19.300'}, {'label': 'LEGEST', 'pattern':
'TITULO'}, {'label': 'NUM', 'pattern': 'I'}, {'label': 'LEGEST', 'pattern':
'Disposiciones Generales'}, {'label': 'LEGEST', 'pattern': 'Artículo'},
{'label': 'NUM', 'pattern': '1'}, {'label': 'NUM', 'pattern': '1°'}, {'label':
'LEGEST', 'pattern': 'derecho'}]
```

[ ]:
```
from spacy import displacy

# The rules are added and applied to the base model.

# function to save training data to json
def save_data(file, data):
    with open (file, "w", encoding = "utf-8") as f:
        json.dump(data, f, indent = 4)

# function without encoding
def save_data2(file, data):
    with open (file, "w", encoding = "utf-8") as f:
        json.dump(data, f, ensure_ascii = False, indent = 4)


# Building on the blank spaCy model

nlp = spacy.blank("es")

#Creating an Entity Ruler
ruler = nlp.add_pipe("entity_ruler")

# Adding patterns as rules
```

```
ruler.add_patterns(patterns)

# creating the training database
TRAIN_DATA = []

# iterating over the corpus again
for sentence in corpus:
    doc = nlp(sentence)
    displacy.render(doc, jupyter=True, style='ent')
    #rremember, that the entities need to be a dictionary of index 1 of the␣
 ↪list, so it needs to be an empty list
    entities = []
# extracting entities
    for ent in doc.ents:

        # adding the entities in the correct format for training data
        entities.append([ent.start_char, ent.end_char, ent.label_])
    TRAIN_DATA.append([sentence, {"entities": entities}])

# It is saved in json format for later use or to load in other environments.
save_data("./01-Training-Data/TRAIN_DATA_All-220312.json", TRAIN_DATA)
save_data2("./01-Training-Data/TRAIN_DATA_All-220312-No-utf8.json", TRAIN_DATA)
#print (TRAIN_DATA)
```

[10]:
```
print (len(TRAIN_DATA))
```

```
6593
```

[11]:
```
# transforming the data in json format to DocBin (spaCy format)


import pandas as pd
from tqdm import tqdm
from spacy.tokens import DocBin

def load_data(file):
    with open(file, "r", encoding="utf-8") as f:
              data = json.load(f)
    return(data)

# the created json is loaded
TRAIN_DATA = load_data('./01-Training-Data/TRAIN_DATA_All-220312.json')

nlp = spacy.blank("es") # A new spaCy model is loaded
db = DocBin() # a DocBin object is created

for text, annotations in TRAIN_DATA: # training data in the above format
    doc = nlp(text) # a doc object is created from the text
    ents = []
    for start, end, label in annotations["entities"]: # adding character indices
```

```
        span = doc.char_span(start, end, label=label)
        ents.append(span)
    doc.ents = ents # the text is tagged with the entities
    db.add(doc)

db.to_disk("./01-Training-Data/train.spacy") # saving the DocBin object to disk
```

[ ]:
```
db = DocBin()
for text, annotations in training_data:
    doc = nlp(text)
    ents = []
    for start, end, label in annotations:
        span = doc.char_span(start, end, label=label)
        ents.append(span)
    doc.ents = ents
    db.add(doc)
db.to_disk("./train.spacy")
```

[31]:
```
# SpaCy Usage Example offsets_to_biluo_tags to transform from json format to␣
 ↪IOB, to then pass to pandas.

import spacy
from spacy.training import offsets_to_biluo_tags, biluo_tags_to_spans, Example
from spacy import util

TRAIN_DATA = [
    ("¿Quién es Pedro Pedreros?", {"entities": [(10, 24, "PERSON")]}),
    ("Me gusta Santiago y Puerto Montt", {"entities": [(9, 17, "LOC"), (20, 32,␣
 ↪"LOC")]}),
]

print (TRAIN_DATA)


nlp = spacy.load("es_core_news_lg")
docs = []
for text, annot in TRAIN_DATA:
    doc = nlp(text)
    token = doc
    ent = offsets_to_biluo_tags(token, annot["entities"])
    docs.append((text,{"entities":(ent)}))

#biluo_tags_to_spans(docs, tags)
print(docs)

bidata = []
for sent, tag in docs:
    doc = nlp (sent)
    for text in doc:
        token = text.text        bidata.append(token)
```

```
print (bidata)

tags = []
for sent, ent in docs:
    ls = list(ent["entities"])
    tags.append(ls)
print (tags)

#df = pd.DataFrame (bidata, tags, columns = ['col1', 'col2'])
#print (df)
ent = []
for sent, tag in docs:
    #text = sent.split()
    ent.append(tag["entities"])

print (bidata, ent)



biluo_data = []
for text, annot in docs:
    doc = nlp(text)
    gold = Example.from_dict(doc, annot)
    biluo_data.append(gold)

#print(biluo_data)

#df = pd.DataFrame.from_dict(biluo_data, orient="columns")
#df
```

```
[('¿Quién es Pedro Pedreros?', {'entities': [(10, 24, 'PERSON')]}), ('Me gusta
Santiago y Puerto Montt', {'entities': [(9, 17, 'LOC'), (20, 32, 'LOC')]})]
[('¿Quién es Pedro Pedreros?', {'entities': ['O', 'O', 'O', 'B-PERSON',
'L-PERSON', 'O']}), ('Me gusta Santiago y Puerto Montt', {'entities': ['O', 'O',
'U-LOC', 'O', 'B-LOC', 'L-LOC']})]
['¿', 'Quién', 'es', 'Pedro', 'Pedreros', '?', 'Me', 'gusta', 'Santiago', 'y',
'Puerto', 'Montt']
[['O', 'O', 'O', 'B-PERSON', 'L-PERSON', 'O'], ['O', 'O', 'U-LOC', 'O', 'B-LOC',
'L-LOC']]
['¿', 'Quién', 'es', 'Pedro', 'Pedreros', '?', 'Me', 'gusta', 'Santiago', 'y',
'Puerto', 'Montt'] [['O', 'O', 'O', 'B-PERSON', 'L-PERSON', 'O'], ['O', 'O',
'U-LOC', 'O', 'B-LOC', 'L-LOC']]
```

[ ]:
```
# Transforming training data in json to BILUO for use in Keras
import json
import pandas as pd
from pandas import json_normalize
import spacy
from spacy.training import offsets_to_biluo_tags, biluo_tags_to_spans, Example
```

```
# function to save and load training data from json to BILUO
# function without encoding
def save_data2(file, data):
    with open (file, "w", encoding = "utf-8") as f:
        json.dump(data, f, ensure_ascii = False, indent = 4)

def load_data(file):
    with open(file, "r", encoding="utf-8") as f:
              data = json.load(f)
    return(data)
```

[34]:
```
# the created json is loaded
TRAIN_DATA = load_data('./01-Training-Data/TRAIN_DATA_All-220312.json')

nlp = spacy.load("es_core_news_lg")
docs = []
for text, annot in TRAIN_DATA:
    doc = nlp(text)
    #spacy.training.offsets_to_biluo_tags(nlp.make_doc(text), annot["entities"])
    ent = offsets_to_biluo_tags(doc, annot["entities"])
    docs.append((text,{"entities":(ent)}))

#biluo_tags_to_spans(docs, tags)
#print(docs)

bidata = []
for sent, tag in docs:
    doc = nlp (sent)
    for text in doc:
        token = text.text        bidata.append(token)
#print (bidata)

tags = []
for sent, ent in docs:
    ls = list(ent["entities"])
    tags.append(ls)
#print (tags)

#df = pd.DataFrame (bidata, tags, columns = ['col1', 'col2'])
#print (df)

ent = []
for sent, tag in docs:
    #text = sent.split()
    ent.append(tag["entities"])

#print (bidata, ent)

#biluo_data = []
#for text, annot in docs:
```

```
#     doc = nlp(text)
#     gold = Example.from_dict(doc, annot)
#     biluo_data.append(gold)


# dictionary of lists
databi = {'text': bidata, 'tag': ent}

#df = pd.DataFrame(databi)

#df.to_csv("./01-Training-Data/train-BILUO.csv", header=False, index=False)

#biluo_data[0:2]
# guardando en disco
save_data2("./01-Training-Data/train-BILUO.json", databi)
```

C Appendix: Generating word-vectors with Gensim/-Word2vec

Using the examples from the `Gensim` tutorial, we want to train a model using word2vec to later be integrated into the spaCy pipeline.

Preparing the environment

[2]:
```
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s',␣
↪level=logging.INFO)
```

Loading libraries and data

The first model trained with `Gensim` only includes decrees. Given the poor results as in the training in spaCy, it was considered to incorporate all the legal text compiled to date, which incorporates laws, decrees and resolutions linked to the environmental legal field.

[11]:
```
from gensim.test.utils import datapath
from gensim import utils

class MyCorpus:
    """An iterator that produces sentences (lists of str)."""

    def __iter__(self):
        corpus_path = '00-Data/texto_all_ldr.txt' # a txt with laws, decrees␣
↪and resolutions linked to the environmental legal field is loaded
        for line in open(corpus_path):
            # assume there is one document per line, tokens separated by␣
↪whitespace
            yield utils.simple_preprocess(line)
```

The model is trained using the text

The model is trained with an N of 300 dimensions of the N-dimensional space (default = 100), and in addition, all those words that have less than 10 repetitions in the text are removed (default = 5).

[35]:
```
import gensim.models

sentences = MyCorpus()
model = gensim.models.Word2Vec(sentences=sentences, min_count=3,␣
↪vector_size=300, workers=8, seed=79, epochs=20)
```

```
2022-01-14 23:49:20,482 : INFO : collecting all words and their counts
2022-01-14 23:49:20,484 : INFO : PROGRESS: at sentence #0, processed 0 words,
keeping 0 word types
2022-01-14 23:49:20,676 : INFO : PROGRESS: at sentence #10000, processed 164387
words, keeping 8589 word types
2022-01-14 23:49:20,707 : INFO : collected 9298 word types from a corpus of
190472 raw words and 11927 sentences
2022-01-14 23:49:20,708 : INFO : Creating a fresh vocabulary
2022-01-14 23:49:20,721 : INFO : Word2Vec lifecycle event {'msg':
'effective_min_count=3 retains 4519 unique words (48.601849860184984%% of
original 9298, drops 4779)', 'datetime': '2022-01-14T23:49:20.721086', 'gensim':
'4.1.2', 'python': '3.8.10 (default, Nov 26 2021, 20:14:08) \n[GCC 9.3.0]',
'platform': 'Linux-5.11.0-46-generic-x86_64-with-glibc2.29', 'event':
'prepare_vocab'}
2022-01-14 23:49:20,721 : INFO : Word2Vec lifecycle event {'msg':
'effective_min_count=3 leaves 184245 word corpus (96.73075307656768%% of
original 190472, drops 6227)', 'datetime': '2022-01-14T23:49:20.721647',
'gensim': '4.1.2', 'python': '3.8.10 (default, Nov 26 2021, 20:14:08) \n[GCC
9.3.0]', 'platform': 'Linux-5.11.0-46-generic-x86_64-with-glibc2.29', 'event':
'prepare_vocab'}
2022-01-14 23:49:20,743 : INFO : deleting the raw counts dictionary of 9298
items
2022-01-14 23:49:20,744 : INFO : sample=0.001 downsamples 33 most-common words
2022-01-14 23:49:20,744 : INFO : Word2Vec lifecycle event {'msg': 'downsampling
leaves estimated 126195.21034856439 word corpus (68.5%% of prior 184245)',
'datetime': '2022-01-14T23:49:20.744497', 'gensim': '4.1.2', 'python': '3.8.10
(default, Nov 26 2021, 20:14:08) \n[GCC 9.3.0]', 'platform':
'Linux-5.11.0-46-generic-x86_64-with-glibc2.29', 'event': 'prepare_vocab'}
2022-01-14 23:49:20,781 : INFO : estimated required memory for 4519 words and
300 dimensions: 13105100 bytes
2022-01-14 23:49:20,782 : INFO : resetting layer weights
2022-01-14 23:49:20,787 : INFO : Word2Vec lifecycle event {'update': False,
'trim_rule': 'None', 'datetime': '2022-01-14T23:49:20.787556', 'gensim':
'4.1.2', 'python': '3.8.10 (default, Nov 26 2021, 20:14:08) \n[GCC 9.3.0]',
'platform': 'Linux-5.11.0-46-generic-x86_64-with-glibc2.29', 'event':
'build_vocab'}
2022-01-14 23:49:20,788 : INFO : Word2Vec lifecycle event {'msg': 'training
model with 8 workers on 4519 vocabulary and 300 features, using sg=0 hs=0
sample=0.001 negative=5 window=5 shrink_windows=True', 'datetime':
'2022-01-14T23:49:20.788258', 'gensim': '4.1.2', 'python': '3.8.10 (default, Nov
26 2021, 20:14:08) \n[GCC 9.3.0]', 'platform':
'Linux-5.11.0-46-generic-x86_64-with-glibc2.29', 'event': 'train'}
2022-01-14 23:49:21,096 : INFO : worker thread finished; awaiting finish of 7
more threads
2022-01-14 23:49:21,104 : INFO : worker thread finished; awaiting finish of 6
more threads
2022-01-14 23:49:21,112 : INFO : worker thread finished; awaiting finish of 5
more threads
2022-01-14 23:49:21,113 : INFO : worker thread finished; awaiting finish of 4
more threads
```

```
2022-01-14 23:49:21,114 : INFO : worker thread finished; awaiting finish of 3
more threads
2022-01-14 23:49:21,115 : INFO : worker thread finished; awaiting finish of 2
more threads
2022-01-14 23:49:21,116 : INFO : worker thread finished; awaiting finish of 1
more threads
2022-01-14 23:49:21,117 : INFO : worker thread finished; awaiting finish of 0
more threads
2022-01-14 23:49:21,118 : INFO : EPOCH - 1 : training on 190472 raw words
(126260 effective words) took 0.3s, 385135 effective words/s
2022-01-14 23:49:21,508 : INFO : worker thread finished; awaiting finish of 7
more threads
2022-01-14 23:49:21,512 : INFO : worker thread finished; awaiting finish of 6
more threads
2022-01-14 23:49:21,520 : INFO : worker thread finished; awaiting finish of 5
more threads
2022-01-14 23:49:21,521 : INFO : worker thread finished; awaiting finish of 4
more threads
2022-01-14 23:49:21,527 : INFO : worker thread finished; awaiting finish of 3
more threads
2022-01-14 23:49:21,531 : INFO : worker thread finished; awaiting finish of 2
more threads
2022-01-14 23:49:21,532 : INFO : worker thread finished; awaiting finish of 1
more threads
2022-01-14 23:49:21,533 : INFO : worker thread finished; awaiting finish of 0
more threads

...OUTPUT TRUNCATED...

2022-01-14 23:49:27,203 : INFO : EPOCH - 19 : training on 190472 raw words
(126370 effective words) took 0.4s, 322846 effective words/s
2022-01-14 23:49:27,504 : INFO : worker thread finished; awaiting finish of 7
more threads
2022-01-14 23:49:27,508 : INFO : worker thread finished; awaiting finish of 6
more threads
2022-01-14 23:49:27,509 : INFO : worker thread finished; awaiting finish of 5
more threads
2022-01-14 23:49:27,510 : INFO : worker thread finished; awaiting finish of 4
more threads
2022-01-14 23:49:27,515 : INFO : worker thread finished; awaiting finish of 3
more threads
2022-01-14 23:49:27,518 : INFO : worker thread finished; awaiting finish of 2
more threads
2022-01-14 23:49:27,522 : INFO : worker thread finished; awaiting finish of 1
more threads
2022-01-14 23:49:27,525 : INFO : worker thread finished; awaiting finish of 0
more threads
2022-01-14 23:49:27,525 : INFO : EPOCH - 20 : training on 190472 raw words
(126244 effective words) took 0.3s, 394800 effective words/s
2022-01-14 23:49:27,526 : INFO : Word2Vec lifecycle event {'msg': 'training on
```

```
3809440 raw words (2524663 effective words) took 6.7s, 374721 effective
words/s', 'datetime': '2022-01-14T23:49:27.526146', 'gensim': '4.1.2', 'python':
'3.8.10 (default, Nov 26 2021, 20:14:08) \n[GCC 9.3.0]', 'platform':
'Linux-5.11.0-46-generic-x86_64-with-glibc2.29', 'event': 'train'}
2022-01-14 23:49:27,526 : INFO : Word2Vec lifecycle event {'params':
'Word2Vec(vocab=4519, vector_size=300, alpha=0.025)', 'datetime':
'2022-01-14T23:49:27.526669', 'gensim': '4.1.2', 'python': '3.8.10 (default, Nov
26 2021, 20:14:08) \n[GCC 9.3.0]', 'platform':
'Linux-5.11.0-46-generic-x86_64-with-glibc2.29', 'event': 'created'}
```

Obtaining the vector corresponding to a word that is common to the model:

[13]:
```
vec_art = model.wv['artículo']
vec_art
```

[13]:
```
array([-7.56538093e-01, -2.67952919e-01,  2.12408364e-01, -3.06433737e-02,
        7.24331617e-01,  2.30615929e-01,  7.64467657e-01, -3.50750446e-01,
        5.59174299e-01,  3.23003709e-01,  6.58398867e-01,  6.31558478e-01,
       -1.32274926e+00, -1.13901830e+00,  1.93764362e-02,  2.54746992e-02,
       -3.46313506e-01, -5.02166927e-01,  3.97940308e-01, -1.66866854e-01,
       -5.64845979e-01, -3.20573062e-01, -4.12516356e-01,  1.57046854e-01,
        1.36163855e+00,  2.03398004e-01,  3.06929857e-01,  1.56452432e-01,
       -1.52951762e-01,  4.22827750e-01, -6.44364297e-01, -4.90315795e-01,
 ...OUTPUT TRUNCATED...
        9.72442329e-01,  4.37452830e-02, -8.71309578e-01,  1.40619123e+00,
       -2.51317620e-01, -3.50453109e-01, -9.13374543e-01,  1.03986561e+00,
        4.09335583e-01,  1.92796454e-01,  3.79026085e-01, -5.84970355e-01],
      dtype=float32)
```

Reviewing vocabulary present in the model:

[36]:
```
for index, word in enumerate(model.wv.index_to_key):
    if index == 11:
        break
    print(f"word #{index}/{len(model.wv.index_to_key)} is {word}")
```

```
word #0/4519 is de
word #1/4519 is la
word #2/4519 is el
word #3/4519 is en
word #4/4519 is del
word #5/4519 is que
word #6/4519 is los
word #7/4519 is las
word #8/4519 is se
word #9/4519 is para
word #10/4519 is artículo
```

Checking the similarity between pairs of words with the trained model, it can be seen that an `article` looks more like a `decree` than a `law`, and not at all with `pollutant`:

[37]:
```
pairs = [
    ('artículo', 'ley'),
    ('artículo', 'decreto'),
    ('artículo', 'resolución'),
    ('artículo', 'ambiental'),
    ('artículo', 'contaminante'),
]
for w1, w2 in pairs:
    print('%r\t%r\t%.2f' % (w1, w2, model.wv.similarity(w1, w2)))
```

```
'artículo'      'ley'   0.43
'artículo'      'decreto'       0.45
'artículo'      'resolución'    0.23
'artículo'      'ambiental'     0.27
'artículo'      'contaminante'  0.10
```

[38]:
```
model.wv.most_similar(positive=['contaminante'])
```

[38]:
```
[('efluente', 0.7263919115066528),
 ('elemento', 0.7239376306533813),
 ('balance', 0.6955969333648682),
 ('parámetro', 0.6875048279762268),
 ('porcentaje', 0.682728111743927),
 ('so', 0.6723634004592896),
 ('arsénico', 0.6719986200332642),
 ('medir', 0.6626269817352295),
 ('cálculo', 0.6621272563934326),
 ('magnitud', 0.6558148264884949)]
```

[39]:
```
model.wv.most_similar(positive=['vida'])
```

[39]:
```
[('biodiversidad', 0.764335572719574),
 ('población', 0.7608422040939331),
 ('astronómica', 0.756157398223877),
 ('tutelar', 0.7206318974494934),
 ('contraparte', 0.7189157605171204),
 ('cielo', 0.7182424664497375),
 ('integridad', 0.7174928784370422),
 ('humana', 0.7090886235237122),
 ('riesgo', 0.7060478925704956),
 ('conservación', 0.694452166557312)]
```

[50]:
```
model.wv.most_similar(positive=['artículo'])
```

[50]:
```
[('inciso', 0.733634889125824),
 ('letra', 0.7036660313606262),
 ('código', 0.6807164549827576),
 ('artículos', 0.6473606824874878),
 ('párrafo', 0.642257809638977),
 ('art', 0.6310014128684998),
```

```
 ('letras', 0.6270051002502441),
 ('señaladas', 0.6098313927650452),
 ('bis', 0.6054291725158691),
 ('señalados', 0.5988930463790894)]
```

[54]:
```
model.wv.most_similar(positive=['regirá'])
```

[54]:
```
[('regirán', 0.755047082901001),
 ('entenderá', 0.679348349571228),
 ('refieren', 0.6486899256706238),
 ('exceptúan', 0.6401525735855103),
 ('aplicará', 0.6281970143318176),
 ('sancionará', 0.6183983087539673),
 ('aplicarán', 0.6077778935432434),
 ('refiere', 0.5966508984565735),
 ('indicarán', 0.5957207083702087),
 ('diligencia', 0.5951999425888062)]
```

Saving the model to disk

[52]:
```
from gensim.test.utils import common_texts, get_tmpfile
from gensim.models import Word2Vec

model.save("02-Gensim-models/gensim-model-002.model.bin")

#model = Word2Vec(common_texts, size=100, window=5, min_count=1, workers=4)
model.wv.save_word2vec_format("02-Gensim-models/gensim-model-002.txt")
```

```
2022-01-15 09:06:47,190 : INFO : Word2Vec lifecycle event {'fname_or_handle':
'02-Gensim-models/gensim-model-002.model.bin', 'separately': 'None',
'sep_limit': 10485760, 'ignore': frozenset(), 'datetime':
'2022-01-15T09:06:47.190026', 'gensim': '4.1.2', 'python': '3.8.10 (default, Nov
26 2021, 20:14:08) \n[GCC 9.3.0]', 'platform':
'Linux-5.11.0-46-generic-x86_64-with-glibc2.29', 'event': 'saving'}
2022-01-15 09:06:47,191 : INFO : not storing attribute cum_table
2022-01-15 09:06:47,204 : INFO : saved 02-Gensim-models/gensim-
model-002.model.bin
2022-01-15 09:06:47,207 : INFO : storing 4519x300 projection weights into
02-Gensim-models/gensim-model-002.txt
```

[43]:
```
!gzip ./02-Gensim-models/gensim-model-002.txt
```

Loading the model into spaCy for common use

spaCy loaded a total of 4,519 vectors contained in the word2vec model. Compared to the 300,000 vectors contained in the long spaCy model, much more environmental legislation would be required to reach a comparable value. This may be insufficient to be able to generate a change in the training of the model.

[44]:
```
!python -m spacy init vectors es ./02-Gensim-models/gensim-model-002.txt.gz␣
↪--name model002.model ./gensim_models
```

```
 Creating blank nlp object for language 'es'
[2022-01-14 23:53:11,082] [INFO] Reading vectors from 02-Gensim-models/gensim-
model-002.txt.gz
4519it [00:00, 14792.23it/s]
[2022-01-14 23:53:11,390] [INFO] Loaded vectors from 02-Gensim-models/gensim-
model-002.txt.gz
 Successfully converted 4519 vectors
 Saved nlp object with vectors to output directory. You can now use

the path to it in your config as the 'vectors' setting in [initialize].
/home/anibal/gitsources/tesis-estadistica/03-Experimentos/Ejem-1/gensim_models
```

[45]:
```
import spacy
```

[46]:
```
gensim_model = spacy.load("./gensim_models")
```

[47]:
```
tokens = gensim_model("Artículo 21.- Aquí hay una sentencia. El proyecto deberá␣
↪ser presentado a la Subsecretaría antes del plazo de 10 días hábiles. Por lo␣
↪demás, se deberá mitigar el efecto de las emisiones al medio ambiente de␣
↪acuerdo a lo señalado en el artículo 3 de la Norma de Emisiones de 2012. Que␣
↪los niveles y periodo de exposición a los contaminantes serán regulados por␣
↪la normativa vigente. Para cada medición realizada, se elegirá el promedio␣
↪de los valores medidos. Artículo 3.- La presente norma será de aplicación␣
↪nacional.")
```

You can see that the word2vec model was trained on only lowercase words, so it doesn't know the word 'Artículo'. Also, numbers and some stopwords were removed. This was done automatically when loading the text.

[48]:
```
for token in tokens:
    print (token.text, token.has_vector, token.vector_norm, token.is_oov)
```

```
Artículo False 0.0 True
21.- False 0.0 True
Aquí False 0.0 True
hay True 1.3766278 False
una True 11.793155 False
sentencia True 4.3536572 False
. False 0.0 True
El False 0.0 True
proyecto True 13.025288 False
deberá True 16.490122 False
ser True 11.243809 False
presentado True 0.99033886 False
a False 0.0 True
la True 8.280686 False
Subsecretaría False 0.0 True
antes True 5.015717 False
```

```
del True 9.070202 False
...OUTPUT TRUNCATED.
```

D Appendix: Training the model with spaCy

This appendix shows the configuration files used to train the NER model and shows the runtime obtained from a Jupyter notebook.

```
title: "NER legal pipeline (Named Entity Recognition)"
description: "Experimento 7 para entrenar un modelo NER con spaCy que reconozca␣
 ↪estructuras y terminos legales en base a Decretos. Se preparó un diccionario␣
 ↪de terminos y se anotó una base de datos de entrenamiento con spaCy␣
 ↪(TRAIN_DATA_Dec.json, ver Data ejemplos)."
# Se mantienen las notas de cada apartado.

# Version 0.0.5
# TRAIN_DATA_All-220312.json

# Se puede hacer referencia a las variables en project.yml usando $vars.var_name
vars:
  name: "ner_exp_7"
  lang: "es"
  train: "train.json"
  dev: "dev.json"
  version: "0.0.9" # se modifica acorde
  # Configura tu ID de GPU, -1 es CPU
  gpu_id: -1
  # Modelo de vectores para entrenamiento con vectores
  vectors_model: "es_core_news_lg"

# Estos son los directorios que necesita el proyecto. La CLI del proyecto se␣
 ↪asegurará de que
# siempre existan.
directories: ["assets", "corpus", "configs", "training", "scripts", "packages"]

# Activos que deben descargarse o estar disponibles en el directorio. Los␣
 ↪enviaremos con el
# proyecto para que no sea necesario descargarlos.
assets:
  - dest: "assets/train.json"
    description: "Training data converted from the v2 `train_ner.py` example␣
 ↪with `srsly.write_json(̈train.json¨, TRAIN_DATA)`"
  - dest: "assets/dev.json"
    description: "Development data"

#Los flujos de trabajo son secuencias de comandos (ver más abajo) que se␣
 ↪ejecutan en orden. Puede
# ejecutarlos a través de "ejecución de proyecto espacial [workflow]". Si las␣
 ↪entradas /
# salidas de un comando no han cambiado, no se volverá a ejecutar.
workflows:
  all:
    - convert
    - create-config
```

```
    - train
    # - train-with-vectors
    - evaluate

# Comandos de proyecto, especificados en un estilo similar a los archivos de␣
 ↪configuración de CI
# (por ejemplo, Azure Pipelines). El nombre es el nombre del comando que le␣
 ↪permite activar el
# comando a través de "proyecto espacial ejecutar [comando] [ruta]". El mensaje␣
 ↪de ayuda es
# opcional y se muestra al ejecutar "spacy project run [comando opcional] [ruta]␣
 ↪--help".
commands:
  - name: "download"
    help: "Download a spaCy model with pretrained vectors"
    script:
      - "python -m spacy download $vars.vectors_model"

  - name: "convert"
    help: "Convert the data to spaCy's binary format"
    script:
      - "python scripts/convert.py $vars.lang assets/$vars.train corpus/train.
 ↪spacy"
      - "python scripts/convert.py $vars.lang assets/$vars.dev corpus/dev.spacy"
    deps:
      - "assets/$vars.train"
      - "assets/$vars.dev"
      - "scripts/convert.py"
    outputs:
      - "corpus/train.spacy"
      - "corpus/dev.spacy"

  - name: "create-config"
    help: "Create a new config with an NER pipeline component"
    script:
      - "python -m spacy init config --lang $vars.lang --pipeline ner configs/
 ↪config.cfg --force"
    outputs:
      - "configs/config.cfg"
  - name: "train"
    help: "Train the NER model"
    script:
      - "python -m spacy train configs/config.cfg --output training/ --paths.
 ↪train corpus/train.spacy --paths.dev corpus/dev.spacy --training.
 ↪eval_frequency 10 --training.patience 50 --gpu-id $vars.gpu_id"
    deps:
      - "configs/config.cfg"
      - "corpus/train.spacy"
      - "corpus/dev.spacy"
    outputs:
      - "training/model-best"
```

```
 - name: "train-with-vectors"
   help: "Train the NER model with vectors"
   script:
     - "python -m spacy train configs/config.cfg --output training/ --paths.
↪train corpus/train.spacy --paths.dev corpus/dev.spacy --training.
↪eval_frequency 10 --training.patience 50 --gpu-id $vars.gpu_id --initialize.
↪vectors $vars.vectors_model --components.tok2vec.model.embed.
↪include_static_vectors true"
   deps:
     - "configs/config.cfg"
     - "corpus/train.spacy"
     - "corpus/dev.spacy"
   outputs:
     - "training/model-best"

 - name: "evaluate"
   help: "Evaluate the model and export metrics"
   script:
     - "python -m spacy evaluate training/model-best corpus/dev.spacy --output␣
↪training/metrics.json"
   deps:
     - "corpus/dev.spacy"
     - "training/model-best"
   outputs:
     - "training/metrics.json"

 - name: package
   help: "Package the trained model as a pip package"
   script:
     - "python -m spacy package training/model-best packages --name $vars.name␣
↪--version $vars.version --force"
   deps:
     - "training/model-best"
   outputs_no_cache:
     - "packages/$vars.lang_$vars.name-$vars.version/dist/$vars.lang_$vars.
↪name-$vars.version.tar.gz"

 - name: visualize-model
   help: Visualize the model's output interactively using Streamlit
   script:
     - "streamlit run scripts/visualize_model.py training/model-best Decreto Nº␣
↪11, de fecha 8 de abril de 2015, del Ministerio de Salud.Que de acuerdo a la␣
↪ley Nº 19.300, el Estado tiene por función dictar normas secundarias de␣
↪calidad ambiental para regular la presencia de contaminantes en el medio␣
↪ambiente. DISPOSICIONES  GENERALES. La presente norma de emisión establece la␣
↪concentración máxima de contaminantes. TÍTULO I. Párrafo 1º. Artículo 9º.-␣
↪Responsabilidad extendida del productor. Artículo 29.- Educación ambiental. El␣
↪Ministerio diseñará e implementará programas de educación ambiental, formal e␣
↪informal, destinados a transmitir conocimientos y crear conciencia en la␣
↪comunidad sobre la prevención en la generación de residuos. Artículo 38.-␣
↪Fiscalización y seguimiento. Corresponderá a la Superintendencia la␣
↪fiscalización del cumplimiento. TITULO V. Del Fondo de Protección Ambiental.␣
↪Artículo 66.- El Ministerio del Medio Ambiente tendrá a su cargo la␣
↪administración de un Fondo de Protección Ambiental."
```

```
    deps:
      - "scripts/visualize_model.py"
      - "training/model-best"
```

```
[paths]
raw_text = null
pretrain = "./corpus/pretrain/"
train = null
dev = null
vectors = "./gensim-models/"
init_tok2vec = null

[system]
gpu_allocator = null
seed = 0

[nlp]
lang = "es"
pipeline = ["tok2vec","ner"]
batch_size = 1000
disabled = []
before_creation = null
after_creation = null
after_pipeline_creation = null
tokenizer = "@tokenizers":"spacy.Tokenizer.v1"

[components]

[components.ner]
factory = "ner"
moves = null
update_with_oracle_cut_size = 100

[components.ner.model]
@architectures = "spacy.TransitionBasedParser.v2"
state_type = "ner"
extra_state_tokens = false
hidden_width = 64
maxout_pieces = 2
use_upper = true
nO = null

[components.ner.model.tok2vec]
@architectures = "spacy.Tok2VecListener.v1"
width = $components.tok2vec.model.encode.width
upstream = "*"

[components.tok2vec]
factory = "tok2vec"
```

```
[components.tok2vec.model]
@architectures = "spacy.Tok2Vec.v2"

[components.tok2vec.model.embed]
@architectures = "spacy.MultiHashEmbed.v2"
width = $components.tok2vec.model.encode.width
attrs = ["NORM","PREFIX","SUFFIX","SHAPE"]
rows = [5000,2500,2500,2500]
include_static_vectors = false

[components.tok2vec.model.encode]
@architectures = "spacy.MaxoutWindowEncoder.v2"
width = 96
depth = 4
window_size = 1
maxout_pieces = 3

[corpora]

[corpora.dev]
@readers = "spacy.Corpus.v1"
path = $paths.dev
max_length = 0
gold_preproc = false
limit = 0
augmenter = null

[corpora.pretrain]
@readers = "spacy.JsonlCorpus.v1"
path = $paths.pretrain
min_length = 0
max_length = 0
limit = 0

[corpora.train]
@readers = "spacy.Corpus.v1"
path = $paths.train
max_length = 2000
gold_preproc = false
limit = 0
augmenter = null

[training]
dev_corpus = "corpora.dev"
train_corpus = "corpora.train"
seed = $system.seed
gpu_allocator = $system.gpu_allocator
dropout = 0.1
accumulate_gradient = 1
```

```
patience = 1600
max_epochs = 0
max_steps = 20000
eval_frequency = 200
frozen_components = []
before_to_disk = null

[training.batcher]
@batchers = "spacy.batch_by_words.v1"
discard_oversize = false
tolerance = 0.2
get_length = null

[training.batcher.size]
@schedules = "compounding.v1"
start = 100
stop = 1000
compound = 1.001
t = 0.0

[training.logger]
@loggers = "spacy.ConsoleLogger.v1"
progress_bar = false

[training.optimizer]
@optimizers = "Adam.v1"
beta1 = 0.9
beta2 = 0.999
L2_is_weight_decay = true
L2 = 0.01
grad_clip = 1.0
use_averages = false
eps = 0.00000001
learn_rate = 0.001

[training.score_weights]
ents_per_type = null
ents_f = 1.0
ents_p = 0.0
ents_r = 0.0

[pretraining]
max_epochs = 1000
dropout = 0.2
n_save_every = null
#n_save_epoch = 10
component = "tok2vec"
layer = ""
corpus = "corpora.pretrain"
```

```
[pretraining.batcher]
@batchers = "spacy.batch_by_words.v1"
size = 3000
discard_oversize = false
tolerance = 0.2
get_length = null

[pretraining.objective]
@architectures = "spacy.PretrainVectors.v1"
maxout_pieces = 3
hidden_size = 300
loss = "cosine"

[pretraining.optimizer]
@optimizers = "Adam.v1"
beta1 = 0.9
beta2 = 0.999
L2_is_weight_decay = true
L2 = 0.01
grad_clip = 1.0
use_averages = true
eps = 0.00000001
learn_rate = 0.001

[initialize]
vectors = $paths.vectors
init_tok2vec = $paths.init_tok2vec
vocab_data = null
lookups = null
before_init = null
after_init = null

[initialize.components]

[initialize.tokenizer]
```

```
[paths]
raw_text = null
pretrain = "./corpus/pretrain.spacy"
train = "./corpus/"
dev = "./corpus/"
vectors = "./gensim-models/"
init_tok2vec = "./pretrain/model999.bin"


[system]
gpu_allocator = null
seed = 79
```

```
[nlp]
lang = "es"
pipeline = ["tok2vec","ner"]
batch_size = 1000
disabled = []
before_creation = null
after_creation = null
after_pipeline_creation = null
tokenizer = "@tokenizers":"spacy.Tokenizer.v1"

[components]

[components.ner]
factory = "ner"
moves = null
update_with_oracle_cut_size = 100

[components.ner.model]
@architectures = "spacy.TransitionBasedParser.v2"
state_type = "ner"
extra_state_tokens = false
hidden_width = 64
maxout_pieces = 2
use_upper = true
nO = null

[components.ner.model.tok2vec]
@architectures = "spacy.Tok2VecListener.v1"
width = $components.tok2vec.model.encode.width
upstream = "*"

[components.tok2vec]
factory = "tok2vec"

[components.tok2vec.model]
@architectures = "spacy.Tok2Vec.v2"

[components.tok2vec.model.embed]
@architectures = "spacy.MultiHashEmbed.v2"
width = $components.tok2vec.model.encode.width
attrs = ["NORM","PREFIX","SUFFIX","SHAPE"]
rows = [5000,2500,2500,2500]
include_static_vectors = true

[components.tok2vec.model.encode]
@architectures = "spacy.MaxoutWindowEncoder.v2"
width = 96
depth = 4
window_size = 1
maxout_pieces = 3
```

```
[corpora]

[corpora.dev]
@readers = "spacy.Corpus.v1"
path = $paths.dev
max_length = 0
gold_preproc = false
limit = 0
augmenter = null

[corpora.train]
@readers = "spacy.Corpus.v1"
path = $paths.train
max_length = 2000
gold_preproc = false
limit = 0
augmenter = null

[training]
dev_corpus = "corpora.dev"
train_corpus = "corpora.train"
seed = $system.seed
gpu_allocator = $system.gpu_allocator
dropout = 0.1
accumulate_gradient = 1
patience = 1600
max_epochs = 0
max_steps = 20000
eval_frequency = 200
frozen_components = []
before_to_disk = null

[training.batcher]
@batchers = "spacy.batch_by_words.v1"
discard_oversize = false
tolerance = 0.2
get_length = null

[training.batcher.size]
@schedules = "compounding.v1"
start = 100
stop = 1000
compound = 1.001
t = 0.0

[training.logger]
@loggers = "spacy.ConsoleLogger.v1"
progress_bar = false
```

```
[training.optimizer]
@optimizers = "Adam.v1"
beta1 = 0.9
beta2 = 0.999
L2_is_weight_decay = true
L2 = 0.01
grad_clip = 1.0
use_averages = false
eps = 0.00000001
learn_rate = 0.001

[training.score_weights]
ents_per_type = null
ents_f = 1.0
ents_p = 0.0
ents_r = 0.0

[pretraining]


[initialize]
vectors = $paths.vectors
init_tok2vec = $paths.init_tok2vec
vocab_data = null
lookups = null
before_init = null
after_init = null


[initialize.components]

[initialize.tokenizer]
```

Trainning NER model with spaCy

[1]:
```
!python -m spacy debug data ./configs/config.cfg
```

```
============================ Data file validation
============================
 Corpus is loadable
 Pipeline can be initialized with data

=============================== Training stats
===============================
Language: es
```

```
Training pipeline: tok2vec, ner
32442 training docs
32442 evaluation docs
 13013 training examples also in evaluation data

============================== Vocab & Vectors

==============================
 1157725 total word(s) in the data (16814 unique)
 4519 vectors (4519 unique keys, 300 dimensions)
 524415 words in training data without vectors (0.45%)

========================== Named Entity Recognition

==========================
 17 label(s)
0 missing value(s) (tokens with '-' label)
 128 entity span(s) with punctuation
 Good amount of examples for all labels
 Examples without occurrences available for all labels
 No entities consisting of or starting/ending with whitespace
Entity spans consisting of or starting/ending with punctuation can not be
trained with a noise level > 0.

================================== Summary

==================================
 5 checks passed
 3 warnings
```

[3]:
```
!python -m spacy project run train
```

```
=================================== train

===================================
Running command: /home/anibal/gitsources/tesis-
estadistica/03-Experimentos/.env/bin/python -m spacy train configs/config.cfg
--output training/ --paths.train corpus/train.spacy --paths.dev corpus/dev.spacy
--training.eval_frequency 10 --training.patience 50 --gpu-id -1
 Using CPU
 To switch to GPU 0, use the option: --gpu-id 0

============================ Initializing pipeline

============================
[W Module.cpp:482] Warning: Disabling benchmark mode for MIOpen is NOT
supported. Overriding value to True (function operator())
[2022-03-17 16:09:09,955] [INFO] Set up nlp object from config
```

```
[2022-03-17 16:09:09,962] [INFO] Pipeline: ['tok2vec', 'ner']
[2022-03-17 16:09:09,964] [INFO] Created vocabulary
[2022-03-17 16:09:10,024] [INFO] Added vectors: ./gensim-models/
[2022-03-17 16:09:10,024] [INFO] Finished initializing nlp object
[2022-03-17 16:09:15,342] [INFO] Initialized pipeline components: ['tok2vec',
'ner']
 Initialized pipeline


============================== Training pipeline

==============================
 Pipeline: ['tok2vec', 'ner']
 Initial learn rate: 0.001
```

E	#	LOSS TOK2VEC	LOSS NER	ENTS_F	ENTS_P	ENTS_R	SCORE
0	0	0.00	58.64	0.03	8.24	0.02	0.00
0	10	2.45	601.75	0.00	0.00	0.00	0.00
0	20	11.30	271.40	0.00	0.00	0.00	0.00
0	30	9.30	257.58	0.02	28.57	0.01	0.00
0	40	5.52	211.41	0.11	35.71	0.06	0.00
0	50	5.84	199.01	1.65	26.38	0.85	0.02
0	60	6.60	205.78	10.52	30.18	6.37	0.11
0	70	12.90	301.87	11.53	33.77	6.95	0.12
0	80	8.73	211.12	13.90	19.50	10.80	0.14
0	90	10.05	215.74	20.99	59.64	12.74	0.21
0	100	7.54	183.49	30.80	48.47	22.57	0.31
0	110	51.46	159.17	33.44	47.85	25.70	0.33
0	120	2360.30	616.20	33.32	44.51	26.62	0.33
0	130	1909.83	345.85	36.64	55.73	27.29	0.37
0	140	8.40	185.85	41.45	55.44	33.09	0.41
0	150	10.51	176.07	48.82	65.00	39.09	0.49
0	160	11.41	159.06	45.87	50.34	42.12	0.46
0	170	10.73	145.78	47.15	62.98	37.67	0.47
0	180	9.20	121.54	52.07	61.39	45.21	0.52
0	190	12.38	139.70	52.88	62.11	46.04	0.53
0	200	14.90	142.57	56.35	67.09	48.58	0.56
0	210	15.02	166.01	60.15	67.07	54.52	0.60
0	220	11.64	123.81	62.60	70.51	56.29	0.63
0	230	61.89	209.84	61.49	66.72	57.02	0.61
0	240	22.61	212.19	62.53	67.02	58.60	0.63
0	250	22.82	163.06	68.89	74.24	64.25	0.69
0	260	17.65	142.98	69.36	75.42	64.20	0.69
0	270	465.17	227.89	68.53	72.81	64.73	0.69
0	280	111.28	109.81	70.77	79.39	63.84	0.71
0	290	13.53	108.49	72.12	76.65	68.09	0.72
0	300	16.94	117.58	74.02	79.19	69.49	0.74
0	310	16.93	114.10	71.72	77.18	66.98	0.72
0	320	29.03	173.65	75.89	78.66	73.32	0.76
0	330	16.15	117.87	78.24	82.11	74.71	0.78
0	340	14.79	93.92	76.90	79.69	74.29	0.77

0	350	15.75	103.73	78.40	82.72	74.51	0.78
0	360	36.90	62.06	78.45	82.79	74.54	0.78
0	370	16.14	98.08	76.81	80.18	73.71	0.77
0	380	28.54	129.87	80.69	84.32	77.36	0.81
0	390	23.83	121.57	81.51	85.32	78.03	0.82
0	400	17.24	101.79	79.15	82.70	75.90	0.79
0	410	17.28	95.05	79.15	82.31	76.23	0.79
0	420	33.22	162.30	80.44	84.80	76.51	0.80
0	430	20.18	100.26	84.28	86.44	82.22	0.84
0	440	15.99	88.65	83.94	85.84	82.11	0.84
0	450	28.68	88.02	84.06	85.58	82.59	0.84
0	460	18.46	102.38	85.11	87.07	83.23	0.85
0	470	12.31	61.07	84.43	88.35	80.83	0.84
0	480	41.50	80.32	85.03	86.66	83.46	0.85
0	490	19.95	97.20	85.69	87.11	84.31	0.86
0	500	19.67	100.85	83.04	84.15	81.95	0.83
0	510	12.41	66.56	85.31	88.13	82.67	0.85
0	520	24.13	116.10	85.25	85.18	85.32	0.85
0	530	20.74	103.32	87.52	89.83	85.32	0.88
0	540	18.90	109.62	87.39	88.61	86.20	0.87
0	550	28.59	127.31	85.63	85.80	85.45	0.86
0	560	257.25	120.72	87.47	89.39	85.63	0.87
0	570	32.34	156.15	87.75	89.44	86.13	0.88
0	580	14.35	71.89	88.55	89.23	87.89	0.89
0	590	19.80	96.14	87.87	89.96	85.87	0.88
0	600	234.46	114.99	87.61	88.76	86.49	0.88
0	610	18.51	90.84	89.37	90.81	87.97	0.89
0	620	15.48	81.08	89.80	89.99	89.61	0.90
0	630	17.53	76.49	87.94	90.70	85.33	0.88
0	640	19.68	56.30	88.64	90.60	86.76	0.89
0	650	20.51	102.32	86.92	89.83	84.19	0.87
0	660	23.87	111.81	89.38	90.64	88.16	0.89
0	670	17.05	88.33	90.38	91.97	88.84	0.90
0	680	13.89	65.06	90.93	91.88	90.00	0.91
0	690	18.38	84.55	90.71	91.52	89.90	0.91
0	700	21.48	94.67	90.86	92.41	89.37	0.91
0	710	24.22	112.78	91.32	91.53	91.10	0.91
0	720	14.12	60.12	91.04	90.99	91.09	0.91
0	730	13.92	65.03	91.07	93.11	89.11	0.91
0	740	14256.97	238.09	92.44	94.19	90.75	0.92
0	750	16.69	66.37	92.38	92.59	92.17	0.92
0	760	18.03	73.86	92.60	93.39	91.82	0.93
0	770	17.11	65.64	92.34	93.15	91.54	0.92
0	780	17.34	73.36	92.48	94.60	90.47	0.92
0	790	15.86	61.81	92.64	93.01	92.27	0.93
0	800	21.13	97.71	92.68	93.42	91.94	0.93
0	810	17.27	72.37	92.08	92.79	91.38	0.92
0	820	297.59	186.79	92.64	93.07	92.22	0.93
0	830	20.67	78.76	92.65	93.36	91.96	0.93
0	840	16.26	66.63	93.19	93.89	92.50	0.93

```
  0     850         11.98     40.65   93.12   94.54   91.74    0.93
  0     860         22.62     86.49   92.87   93.11   92.63    0.93
  0     870         26.58     91.46   93.38   93.57   93.19    0.93
  0     880         20.07     72.18   91.77   94.34   89.34    0.92
  0     890         24.69     83.21   92.68   92.44   92.91    0.93
  0     900         20.82     77.72   92.91   94.36   91.51    0.93
  0     910         24.60     77.46   93.77   94.42   93.12    0.94
  0     920         19.06     66.24   93.96   94.65   93.28    0.94
  0     930         15.81     57.59   94.28   94.27   94.30    0.94
  0     940         21.89     68.99   94.88   95.44   94.33    0.95
  0     950         26.67     76.93   93.71   94.09   93.32    0.94
  0     960         34.37    117.52   94.24   94.27   94.21    0.94
  0     970         14.82     51.81   94.71   94.98   94.44    0.95
  0     980         20.73     70.63   93.24   93.05   93.43    0.93
  0     990         19.06     68.95   93.22   94.10   92.36    0.93
 Saved pipeline to output directory
training/model-last
```

[ ]:

[4]:
```
!python -m spacy project run evaluate
```

```
========================================= evaluate

=====================================
Running command: /home/anibal/gitsources/tesis-
estadistica/03-Experimentos/.env/bin/python -m spacy evaluate training/model-
best corpus/dev.spacy --output training/metrics.json
[W Module.cpp:482] Warning: Disabling benchmark mode for MIOpen is NOT
supported. Overriding value to True (function operator())
 Using CPU
 To switch to GPU 0, use the option: --gpu-id 0

========================================= Results

=====================================

TOK     100.00
NER P   95.44
NER R   94.33
NER F   94.88
SPEED   25270


================================== NER (per type)

================================

              P       R       F
```

```
LEGEST    96.97   96.97   96.97
NUM       98.14   98.62   98.38
SAMBL     89.50   93.91   91.65
CAMB      95.47   92.99   94.21
ACCAMB    91.40   93.36   92.37
ILEG      96.88   95.22   96.04
ACC       97.43   95.62   96.51
ADMB      96.61   82.61   89.06
SUJ       94.39   91.42   92.88
CUANT     97.17   97.15   97.16
LUGAR     90.79   91.25   91.02
SA        95.92   50.00   65.73
EST       92.67   92.88   92.78
ILAMB     86.83   75.90   81.00
VAMB      94.41   70.31   80.60
PORC     100.00   89.29   94.34
FECHA     91.11   92.09   91.60

 Saved results to training/metrics.json
```

[5]:
```
!python -m spacy project run package
```

```
================================== package

==================================
Running command: /home/anibal/gitsources/tesis-
estadistica/03-Experimentos/.env/bin/python -m spacy package training/model-best
packages --name ner_exp_7 --version 0.0.9 --force
 Building package artifacts: sdist
 Loaded meta.json from file
training/model-best/meta.json
 Successfully created package 'es_ner_exp_7-0.0.9'
packages/es_ner_exp_7-0.0.9
running sdist
running egg_info
creating es_ner_exp_7.egg-info
writing es_ner_exp_7.egg-info/PKG-INFO
writing dependency_links to es_ner_exp_7.egg-info/dependency_links.txt
writing entry points to es_ner_exp_7.egg-info/entry_points.txt
writing requirements to es_ner_exp_7.egg-info/requires.txt
writing top-level names to es_ner_exp_7.egg-info/top_level.txt
writing manifest file 'es_ner_exp_7.egg-info/SOURCES.txt'
reading manifest file 'es_ner_exp_7.egg-info/SOURCES.txt'
reading manifest template 'MANIFEST.in'
warning: no files found matching 'LICENSE'
writing manifest file 'es_ner_exp_7.egg-info/SOURCES.txt'
warning: sdist: standard file not found: should have one of README, README.rst,
README.txt, README.md
```

```
running check
warning: check: missing required meta-data: url

warning: check: missing meta-data: either (author and author_email) or
(maintainer and maintainer_email) must be supplied

creating es_ner_exp_7-0.0.9
creating es_ner_exp_7-0.0.9/es_ner_exp_7
creating es_ner_exp_7-0.0.9/es_ner_exp_7.egg-info
creating es_ner_exp_7-0.0.9/es_ner_exp_7/es_ner_exp_7-0.0.9
creating es_ner_exp_7-0.0.9/es_ner_exp_7/es_ner_exp_7-0.0.9/ner
creating es_ner_exp_7-0.0.9/es_ner_exp_7/es_ner_exp_7-0.0.9/tok2vec
creating es_ner_exp_7-0.0.9/es_ner_exp_7/es_ner_exp_7-0.0.9/vocab
copying files to es_ner_exp_7-0.0.9…
copying MANIFEST.in -> es_ner_exp_7-0.0.9
copying meta.json -> es_ner_exp_7-0.0.9
copying setup.py -> es_ner_exp_7-0.0.9
copying es_ner_exp_7/__init__.py -> es_ner_exp_7-0.0.9/es_ner_exp_7
copying es_ner_exp_7/meta.json -> es_ner_exp_7-0.0.9/es_ner_exp_7
copying es_ner_exp_7.egg-info/PKG-INFO -> es_ner_exp_7-0.0.9/es_ner_exp_7.egg-
info
copying es_ner_exp_7.egg-info/SOURCES.txt ->
es_ner_exp_7-0.0.9/es_ner_exp_7.egg-info
copying es_ner_exp_7.egg-info/dependency_links.txt ->
es_ner_exp_7-0.0.9/es_ner_exp_7.egg-info
copying es_ner_exp_7.egg-info/entry_points.txt ->
es_ner_exp_7-0.0.9/es_ner_exp_7.egg-info
copying es_ner_exp_7.egg-info/not-zip-safe ->
es_ner_exp_7-0.0.9/es_ner_exp_7.egg-info
copying es_ner_exp_7.egg-info/requires.txt ->
es_ner_exp_7-0.0.9/es_ner_exp_7.egg-info
copying es_ner_exp_7.egg-info/top_level.txt ->
es_ner_exp_7-0.0.9/es_ner_exp_7.egg-info
copying es_ner_exp_7/es_ner_exp_7-0.0.9/config.cfg ->
es_ner_exp_7-0.0.9/es_ner_exp_7/es_ner_exp_7-0.0.9
copying es_ner_exp_7/es_ner_exp_7-0.0.9/meta.json ->
es_ner_exp_7-0.0.9/es_ner_exp_7/es_ner_exp_7-0.0.9
copying es_ner_exp_7/es_ner_exp_7-0.0.9/tokenizer ->
es_ner_exp_7-0.0.9/es_ner_exp_7/es_ner_exp_7-0.0.9
copying es_ner_exp_7/es_ner_exp_7-0.0.9/ner/cfg ->
es_ner_exp_7-0.0.9/es_ner_exp_7/es_ner_exp_7-0.0.9/ner
copying es_ner_exp_7/es_ner_exp_7-0.0.9/ner/model ->
es_ner_exp_7-0.0.9/es_ner_exp_7/es_ner_exp_7-0.0.9/ner
copying es_ner_exp_7/es_ner_exp_7-0.0.9/ner/moves ->
es_ner_exp_7-0.0.9/es_ner_exp_7/es_ner_exp_7-0.0.9/ner
copying es_ner_exp_7/es_ner_exp_7-0.0.9/tok2vec/cfg ->
es_ner_exp_7-0.0.9/es_ner_exp_7/es_ner_exp_7-0.0.9/tok2vec
copying es_ner_exp_7/es_ner_exp_7-0.0.9/tok2vec/model ->
es_ner_exp_7-0.0.9/es_ner_exp_7/es_ner_exp_7-0.0.9/tok2vec
copying es_ner_exp_7/es_ner_exp_7-0.0.9/vocab/key2row ->
```

```
es_ner_exp_7-0.0.9/es_ner_exp_7/es_ner_exp_7-0.0.9/vocab
copying es_ner_exp_7/es_ner_exp_7-0.0.9/vocab/lookups.bin ->
es_ner_exp_7-0.0.9/es_ner_exp_7/es_ner_exp_7-0.0.9/vocab
copying es_ner_exp_7/es_ner_exp_7-0.0.9/vocab/strings.json ->
es_ner_exp_7-0.0.9/es_ner_exp_7/es_ner_exp_7-0.0.9/vocab
copying es_ner_exp_7/es_ner_exp_7-0.0.9/vocab/vectors ->
es_ner_exp_7-0.0.9/es_ner_exp_7/es_ner_exp_7-0.0.9/vocab
Writing es_ner_exp_7-0.0.9/setup.cfg
creating dist
Creating tar archive
removing 'es_ner_exp_7-0.0.9' (and everything under it)
  Successfully created zipped Python package
packages/es_ner_exp_7-0.0.9/dist/es_ner_exp_7-0.0.9.tar.gz
```

[ ]:

E Appendix: K-Fold Crossvalidation data preparation

- This notebook shows how the training data was prepared for K-Fold cross-validate analysis.
- It was separated a 10% of the data for test purposes.
- The ratio between train and dev data was 9:1.
- The files were transformed to spaCy's binary format and sourced to the training pipeline.

E.1 K-Fold Data Split

Importing the training dataset This dataset corresponds to the complete set used in previous experiments.

[1]:
```
%load_ext rpy2.ipython
```

[15]:
```
%%R
setwd("/home/anibal/gitsources/tesis-estadistica/03-Experimentos/Exp-10")
library(tidyverse)
data_dir <- "./data/texto_all_ldr.txt"
tmp <- read_file(data_dir)
text1 <- tibble(read_lines(tmp, skip_empty_rows = TRUE))

text1
```

```
# A tibble: 8,141 × 1
   `read_lines(tmp, skip_empty_rows = TRUE)`
   <chr>
 1 LEY NÚM. 19.300
 2 TITULO I
 3 Disposiciones Generales
```

```
 4 Artículo 1°.- El derecho a vivir en un medio ambiente libre de contaminación…
 5 Artículo 2°.- Para todos los efectos legales, se entenderá por:
 6 a) Biodiversidad o Diversidad Biológica: la variabilidad de los organismos v…
 7 a bis) Biotecnología: se entiende toda aplicación tecnológica que utilice si…
 8 a ter) Cambio Climático: se entiende un cambio de clima atribuido directa o …
 9 b) Conservación del Patrimonio Ambiental: el uso y aprovechamiento racionale…
10 c) Contaminación: la presencia en el ambiente de sustancias, elementos, ener…
# … with 8,131 more rows
```

Shuffling and separate the 10% of the data for test

[43]:
```
%%R

shufData <-text1[sample(nrow(text1)), ]
splitIdx <- sample(1:nrow(shufData), nrow(shufData)*0.1, replace = F)
testSplit <- shufData[splitIdx, ]
ksplit <- shufData[-splitIdx, ]
write.table(testSplit,
            "./data/test_split.txt",
            sep=";",
            col.names = FALSE,
            row.names = FALSE)
folds <- cut(seq(1, nrow(ksplit)), breaks=10, labels=FALSE)
testSplit
```

```
# A tibble: 814 × 1
   `read_lines(tmp, skip_empty_rows = TRUE)`
   <chr>
 1 "    Anótese, publíquese y comuníquese.- Francisco Echeverría Ellsworth, Dir…
 2 "            parte de los gases emitidos por chimeneas utilizadas para esos"
 3 "Manganeso            0,3 (mg/L)        4,8 (g/día)"
 4 "Selenio          mg/L      0,02"
 5 "Aluminio                mg/L      Al              1"
 6 "    b) Si la infracción cometida perjudica gravemente el cauce, y siempre q…
 7 "NCh 2313/14: Aguas Residuales - Métodos de análisis Parte 14: Determinación…
 8 "b) Desde el año 2003, 34 ton/año."
 9 "h) Fomentar y facilitar la participación ciudadana en la evaluación de proy…
10 "                      (CVAFS)"
# … with 804 more rows
```

Iterating to split and write train and dev data

[46]:
```
%%R
for(i in 1:10){
    #Segement your data by fold using the which() function
    testIdx <- which(folds==i, arr.ind=TRUE)
    testData <- ksplit[testIdx, ]
    trainData <- ksplit[-testIdx, ]
    write.table(testData,
                sep = ";",
                file = paste0("./data/dev_",i,".txt"),
```

```
                              row.names = F,
                              col.names = F,
                              quote = F)
    write.table(trainData,
                sep = ";",
                file = paste0("./data/train_",i,".txt"),
                              row.names = F,
                              col.names = F,
                              quote = F)
    #Use the test and train data partitions however you desire...
}
```

E.2 Annotate and transform the data to spaCy's format

[1]:
```
import json
import spacy
import pandas as pd
import numpy as np
import rpy2
from tqdm import tqdm
from spacy.tokens import DocBin
```

[3]:
```
# Using the spaCy large Model to prepare matcher and corpuses.
nlp = spacy.load("es_core_news_lg")

nlp.max_length = 1500000 # improving maximum length to use in memory
```

[3]:
```
# functio to importing json format

def load_data(file):
    with open(file, "r", encoding="utf-8") as f:
              data =  json.load(f)
    return(data)

def load_txt(file):
    with open(file, "r", encoding="utf-8") as f:
        data = f.read()
    return(data)
```

[4]:
```
# function to save training data to json
def save_data(file, data):
    with open (file, "w", encoding = "utf-8") as f:
        json.dump(data, f, indent = 4)

# function without encoding
def save_data2(file, data):
    with open (file, "w", encoding = "utf-8") as f:
        json.dump(data, f, ensure_ascii = False, indent = 4)
```

```
# function to save a txt file
def save_txt(file, data):
    with open (file, "w", encoding = "utf-8") as f:
        f.writelines(data) # It need to be writelines
```

Loading files The files are from the above steps in R, and are stored in `data` folder.

[6]:
```
# assign directory
direc = '/home/anibal/gitsources/tesis-estadistica/03-Experimentos/Exp-10/data/'

# iterate over dev files
all_dev = []
for i in range(1,11):
    all_dev.append(load_txt(direc + "dev_" + str(i) + ".txt"))

# iterate over train files
all_train = []
for i in range(1,11):
    all_train.append(load_txt(direc + "train_" + str(i) + ".txt"))
```

[44]:
```
# length of the first list in the lists

print(len(all_train[4]))
print(len(all_dev[9]))
```

```
1147795
132450
```

Preparing train and dev data to annotate with PhraseMatcher (imported code)

[8]:
```
# Importing keywords to annotate in data:
keyword_dict = pd.read_csv("./data/words.csv")
```

[9]:
```
# The keywords are extracted from the dictionary and added to the matcher with␣
 ↪their tag

from spacy.matcher import PhraseMatcher

ACC_words = [nlp(text) for text in keyword_dict['ACC'].dropna(axis = 0)]
ADMB_words = [nlp(text) for text in keyword_dict['ADMB'].dropna(axis = 0)]
CAMB_words = [nlp(text) for text in keyword_dict['CAMB'].dropna(axis = 0)]
CUANT_words = [nlp(text) for text in keyword_dict['CUANT'].dropna(axis = 0)]
ILAMB_words = [nlp(text) for text in keyword_dict['ILAMB'].dropna(axis = 0)]
ILEG_words = [nlp(text) for text in keyword_dict['ILEG'].dropna(axis = 0)]
SA_words = [nlp(text) for text in keyword_dict['SA'].dropna(axis = 0)]
SAMBL_words = [nlp(text) for text in keyword_dict['SAMBL'].dropna(axis = 0)]
SUJ_words = [nlp(text) for text in keyword_dict['SUJ'].dropna(axis = 0)]
VAMB_words = [nlp(text) for text in keyword_dict['VAMB'].dropna(axis = 0)]
```

```
EST_words = [nlp(text) for text in keyword_dict['EST'].dropna(axis = 0)]
LEGEST_words = [nlp(text) for text in keyword_dict['LEGEST'].dropna(axis = 0)]
## TAGs nuevos
NUM_words = [nlp(text) for text in keyword_dict['NUM'].dropna(axis = 0)]
FECHA_words = [nlp(text) for text in keyword_dict['FECHA'].dropna(axis = 0)]
LUGAR_words = [nlp(text) for text in keyword_dict['LUGAR'].dropna(axis = 0)]
ACCAMB_words = [nlp(text) for text in keyword_dict['ACCAMB'].dropna(axis = 0)]
PORC_words = [nlp(text) for text in keyword_dict['PORC'].dropna(axis = 0)]
#ORG_words = [nlp(text) for text in keyword_dict['ORG'].dropna(axis = 0)]

matcher = PhraseMatcher(nlp.vocab)
matcher.add('ACC', None, *ACC_words)
matcher.add('ADMB', None, *ADMB_words)
matcher.add('CAMB', None, *CAMB_words)
matcher.add('CUANT', None, *CUANT_words)
matcher.add('ILAMB', None, *ILAMB_words)
matcher.add('ILEG', None, *ILEG_words)
matcher.add('SA', None, *SA_words)
matcher.add('SAMBL', None, *SAMBL_words)
matcher.add('SUJ', None, *SUJ_words)
matcher.add('VAMB', None, *VAMB_words)
matcher.add('EST', None, *EST_words)
matcher.add('LEGEST', None, *LEGEST_words)
## TAGs nuevos
matcher.add('NUM', None, *NUM_words)
matcher.add('FECHA', None, *FECHA_words)
matcher.add('LUGAR', None, *LUGAR_words)
matcher.add('ACCAMB', None, *ACCAMB_words)
matcher.add('PORC', None, *PORC_words)
#matcher.add('ORG', None, *ORG_words)
```

[87]:
```
# Iterationg to get the sentences from train and dev lists and creating the␣
 ↪Corpuses. (several minutes)

corp_dev = []
patterns_dev = []
for i in range(10):
    corp_dev.append([])
    patterns_dev.append([])
    doc1 = nlp(all_dev[i])  # access to dev list of lists and process with␣
 ↪spaCy lg model
    for sent in doc1.sents:
        corp_dev[i].append(sent.text) # get sentences
    matches1 = matcher(doc1)
    for match_id, start, end in matches1:
        rule_id = nlp.vocab.strings[match_id]  # acquiring the unique ID, i.e.␣
 ↪'LEGEST'
        span = doc1[start : end]  # getting the range of the match in the␣
 ↪document
```

```
        patterns_dev[i].append({"label":rule_id, "pattern":span.text})  #␣
↪adding by ID and span
```

[89]:
```
# Getting the length of the list of lists. All have different lengths
print(len(corp_dev))
print(len(corp_dev[8]))
print(len(patterns_dev[3]))
```

```
10
585
4373
```

[93]:
```
corp_train = []
patterns_train = []
for j in range(10):
    corp_train.append([])
    patterns_train.append([])
    doc2 = nlp(all_train[j]) # access of train list of lists and process with␣
↪spaCy lg model
    for sent in doc2.sents:
        corp_train[j].append(sent.text) # get sentences
    matches2 = matcher(doc2)
    for match_id, start, end in matches2:
        rule_id = nlp.vocab.strings[match_id]  # acquiring the unique ID, i.e.␣
↪'LEGEST'
        span = doc2[start : end]  # getting the range of the match in the␣
↪document
        patterns_train[j].append({"label":rule_id, "pattern":span.text})  #␣
↪adding by ID and span
```

[96]:
```
# Getting the length of the list of lists.
print(len(corp_train))
print(len(corp_train[7]))
print(len(patterns_train[5]))
len(patterns_train)
```

```
10
5331
41106
```

[96]: 10

Train data Apply rules to corpus and transform train data to json compatible format. The saved json file didn't preserv the list folding.

[97]:
```
#from spacy import displacy

#Creating the training database
TRAIN = []
```

```
for h in range(10):

    TRAIN.append([])

    # The rules are added and applied to the base model.
    # Building on the blank spaCy model
    nlp1 = spacy.blank("es")

    # Creating an Entity Ruler
    ruler1 = nlp1.add_pipe("entity_ruler")
    # Adding patterns as rules
    ruler1.add_patterns(patterns_train[h])

    # iterating over the corpus again
    for sentence in corp_train[h]:

       # Creating spaCy doc
        doc = nlp1(sentence)
        #displacy.render(doc, jupyter=True, style='ent')

        # remember, that the entities need to be a dictionary of index 1 of the␣
 ↪list, so it needs to be an empty list
        entities = []

        # extracting entities
        for ent in doc.ents:
           # adding the entities in the correct format for training data
            entities.append([ent.start_char, ent.end_char, ent.label_])

        TRAIN[h].append([sentence, {"entities": entities}])

print(len(TRAIN))
```

```
10
```

[101]:
```
# assign directory
dire = '/home/anibal/gitsources/tesis-estadistica/03-Experimentos/Exp-10/'

# It is saved in json format for later use or to load in other environments.
for l in range(len(TRAIN)):
    save_data(dire + "./assets/TRAIN_" + str(l) + ".json", TRAIN[l])
    save_data2(dire + "./assets/TRAIN_No-utf8_" + str(l) + ".json", TRAIN[l])
```

E.3 Dev data

Apply rules to corpus and transform dev data to json compatible format. The saved json file didn't preserv the list folding.

[102]:
```
#Creating the developing database
DEV = []

for h in range(10):

    DEV.append([])

    # The rules are added and applied to the base model.
    # Building on the blank spaCy model
    nlp2 = spacy.blank("es")

    # Creating an Entity Ruler
    ruler2 = nlp2.add_pipe("entity_ruler")
    # Adding patterns as rules
    ruler2.add_patterns(patterns_dev[h])

    # iterating over the corpus again
    for sentence in corp_dev[h]:

       # Creating spaCy doc
        doc = nlp2(sentence)
        #displacy.render(doc, jupyter=True, style='ent')

        # remember, that the entities need to be a dictionary of index 1 of the␣
 ↪list, so it needs to be an empty list
        entities = []

        # extracting entities
        for ent in doc.ents:
           # adding the entities in the correct format for training data
            entities.append([ent.start_char, ent.end_char, ent.label_])

        DEV[h].append([sentence, {"entities": entities}])

print(len(DEV))
```

```
10
```

[105]:
```
# assign directory
dire = '/home/anibal/gitsources/tesis-estadistica/03-Experimentos/Exp-10/'

# It is saved in json format for later use or to load in other environments.
for l in range(len(DEV)):
    save_data(dire + "./assets/DEV_" + str(l) + ".json", DEV[l])
    save_data2(dire + "./assets/DEV_No-utf8_" + str(l) + ".json", DEV[l])
```

E.4 Converting files to binary format

It was used directly the lists TRAIN and DEV loaded in memory. Also, was saved the json files in the assets folder. Those files when loaded need it to split in ten folds by sentence.

[108]:
```
# Need it to put in a loop for good


# assign directory
d = '/home/anibal/gitsources/tesis-estadistica/03-Experimentos/Exp-10/'

# Converting to binary format (spaCy)
for h in range(10):
    nlp3 = spacy.blank("es") # Load a new blank spaCy model
    db1 = DocBin() # se crea un objeto DocBin

    for text, annotations in TRAIN[h]: # datos de entrenamiento en el formato␣
 ↪anterior
        doc = nlp3(text) # se crea un objeto doc desde el texto
        ents = []
        for start, end, label in annotations["entities"]: # agregando indices␣
 ↪de caracteres
            span = doc.char_span(start, end, label=label)
            ents.append(span)
        doc.ents = ents # se etiqueta el texto con las entidades
        db1.add(doc)

    db1.to_disk(d + "./corpus/train_" + str(h) + ".spacy") # guardando a disco␣
 ↪el objeto DocBin


    nlp4 = spacy.blank("es") # Se carga un nuevo spaCy model
    db2 = DocBin() # se crea un objeto DocBin

    for text, annotations in DEV[h]: # datos de entrenamiento en el formato␣
 ↪anterior
        doc = nlp4(text) # se crea un objeto doc desde el texto
        ents = []
        for start, end, label in annotations["entities"]: # agregando indices␣
 ↪de caracteres
            span = doc.char_span(start, end, label=label)
            ents.append(span)
        doc.ents = ents # se etiqueta el texto con las entidades
        db2.add(doc)

    db2.to_disk(d + "./corpus/dev_" + str(h) + ".spacy") # guardando a disco el␣
 ↪objeto DocBin
```

E.5 Test data converting

[5]:
```
# assign directory
direc = '/home/anibal/gitsources/tesis-estadistica/03-Experimentos/Exp-10/data/'

# load test text data

testText = load_txt(direc + "test_split.txt")
```

[6]:
```
print(len(testText))
```

```
134475
```

[7]:
```
# Using the spaCy large Model to prepare matcher and corpuses.
nlp = spacy.load("es_core_news_lg")

nlp.max_length = 1500000 # improving maximum length to use in memory
```

[8]:
```
# Importing keywords to annotate in data:
keyword_dict = pd.read_csv("./data/words.csv")
```

[9]:
```
# The keywords are extracted from the dictionary and added to the matcher with␣
 ↪their tag

from spacy.matcher import PhraseMatcher

ACC_words = [nlp(text) for text in keyword_dict['ACC'].dropna(axis = 0)]
ADMB_words = [nlp(text) for text in keyword_dict['ADMB'].dropna(axis = 0)]
CAMB_words = [nlp(text) for text in keyword_dict['CAMB'].dropna(axis = 0)]
CUANT_words = [nlp(text) for text in keyword_dict['CUANT'].dropna(axis = 0)]
ILAMB_words = [nlp(text) for text in keyword_dict['ILAMB'].dropna(axis = 0)]
ILEG_words = [nlp(text) for text in keyword_dict['ILEG'].dropna(axis = 0)]
SA_words = [nlp(text) for text in keyword_dict['SA'].dropna(axis = 0)]
SAMBL_words = [nlp(text) for text in keyword_dict['SAMBL'].dropna(axis = 0)]
SUJ_words = [nlp(text) for text in keyword_dict['SUJ'].dropna(axis = 0)]
VAMB_words = [nlp(text) for text in keyword_dict['VAMB'].dropna(axis = 0)]
EST_words = [nlp(text) for text in keyword_dict['EST'].dropna(axis = 0)]
LEGEST_words = [nlp(text) for text in keyword_dict['LEGEST'].dropna(axis = 0)]
## TAGs nuevos
NUM_words = [nlp(text) for text in keyword_dict['NUM'].dropna(axis = 0)]
FECHA_words = [nlp(text) for text in keyword_dict['FECHA'].dropna(axis = 0)]
LUGAR_words = [nlp(text) for text in keyword_dict['LUGAR'].dropna(axis = 0)]
ACCAMB_words = [nlp(text) for text in keyword_dict['ACCAMB'].dropna(axis = 0)]
PORC_words = [nlp(text) for text in keyword_dict['PORC'].dropna(axis = 0)]
#ORG_words = [nlp(text) for text in keyword_dict['ORG'].dropna(axis = 0)]

matcher = PhraseMatcher(nlp.vocab)
matcher.add('ACC', None, *ACC_words)
matcher.add('ADMB', None, *ADMB_words)
matcher.add('CAMB', None, *CAMB_words)
```

```
matcher.add('CUANT', None, *CUANT_words)
matcher.add('ILAMB', None, *ILAMB_words)
matcher.add('ILEG', None, *ILEG_words)
matcher.add('SA', None, *SA_words)
matcher.add('SAMBL', None, *SAMBL_words)
matcher.add('SUJ', None, *SUJ_words)
matcher.add('VAMB', None, *VAMB_words)
matcher.add('EST', None, *EST_words)
matcher.add('LEGEST', None, *LEGEST_words)
## TAGs nuevos
matcher.add('NUM', None, *NUM_words)
matcher.add('FECHA', None, *FECHA_words)
matcher.add('LUGAR', None, *LUGAR_words)
matcher.add('ACCAMB', None, *ACCAMB_words)
matcher.add('PORC', None, *PORC_words)
#matcher.add('ORG', None, *ORG_words)
```

[10]:
```
corpus = []

doc = nlp(testText)
for sent in doc.sents:
    corpus.append(sent.text)

# longitud del corpus (sentencias)
len(corpus)
```

[10]:
```
602
```

[12]:
```
# Creating PATTERNS for train and dev data

patterns = []
matches = matcher(doc)
for match_id, start, end in matches:
    rule_id = nlp.vocab.strings[match_id]  # acquiring the unique ID, i.e. 'LEGEST'
    span = doc[start : end]  # getting the range of the match in the document
    patterns.append({"label":rule_id, "pattern":span.text})  # adding by ID and span
```

[13]:
```
# assign directory
d = '/home/anibal/gitsources/tesis-estadistica/03-Experimentos/Exp-10/'

# The rules are added and applied to the base model.

# Building on the blank spaCy model
nlp2 = spacy.blank("es")

# Creating an Entity Ruler
ruler2 = nlp2.add_pipe("entity_ruler")
```

```
# Adding patterns as rules
ruler2.add_patterns(patterns)

# Creating the training database
TEST = []

# iterating over the corpus again
for sentence in corpus:
    doc = nlp2(sentence)
    # remember, that the entities need to be a dictionary of index 1 of the␣
 ↪list, so it needs to be an empty list
    entities = []

    # extracting entities
    for ent in doc.ents:
       # adding the entities in the correct format for training data
        entities.append([ent.start_char, ent.end_char, ent.label_])

    TEST.append([sentence, {"entities": entities}])

# It is saved in json format for later use or to load in other environments.
save_data(d + "./assets/TEST.json", TEST)
save_data2(d + "./assets/TEST-No-utf8.json", TEST)
```

[14]:
```
# Transforming test data to binary format

nlp5 = spacy.blank("es")
db = DocBin()

for text, annotations in TEST:
    doc = nlp5(text)
    ents = []
    for start, end, label in annotations["entities"]:
        span = doc.char_span(start, end, label=label)
        ents.append(span)
    doc.ents = ents
    db.add(doc)

db.to_disk(d + "./corpus/test.spacy")
```

[ ]:
```
```

F Appendix: K-Fold Crossvalidation training

This notebook is an excerpt of the entire training process and shows the training of the first datasets.

[44]:
```
!python -m spacy debug data ./configs/config.cfg
```

```
============================ Data file validation

============================
 Corpus is loadable
 Pipeline can be initialized with data

=============================== Training stats

===============================
Language: es
Training pipeline: tok2vec, ner
5300 training docs
615 evaluation docs
 50 training examples also in evaluation data

============================== Vocab & Vectors

==============================
 207994 total word(s) in the data (12090 unique)
 4519 vectors (4519 unique keys, 300 dimensions)
 82487 words in training data without vectors (0.40%)

========================== Named Entity Recognition

==========================
 17 label(s)
0 missing value(s) (tokens with '-' label)
 50 entity span(s) with punctuation
 Good amount of examples for all labels
 Examples without occurrences available for all labels
 No entities consisting of or starting/ending with whitespace
Entity spans consisting of or starting/ending with punctuation can not be
trained with a noise level > 0.

================================== Summary

==================================
 5 checks passed
 3 warnings
```

[47]:
```
!python -m spacy project run train
```

```
================================== train

==================================
```

```
Running command: /home/anibal/gitsources/tesis-
estadistica/03-Experimentos/.env/bin/python -m spacy train configs/config.cfg
--output training/ --paths.train corpus/train_0.spacy --paths.dev
corpus/dev_0.spacy --training.eval_frequency 10 --training.patience 100 --gpu-id
-1
 Using CPU
 To switch to GPU 0, use the option: --gpu-id 0

============================ Initializing pipeline

============================
[W Module.cpp:482] Warning: Disabling benchmark mode for MIOpen is NOT
supported. Overriding value to True (function operator())
[2022-06-27 15:57:39,264] [INFO] Set up nlp object from config
[2022-06-27 15:57:39,271] [INFO] Pipeline: ['tok2vec', 'ner']
[2022-06-27 15:57:39,273] [INFO] Created vocabulary
[2022-06-27 15:57:39,333] [INFO] Added vectors: ./gensim-models/
[2022-06-27 15:57:39,333] [INFO] Finished initializing nlp object
[2022-06-27 15:57:43,650] [INFO] Initialized pipeline components: ['tok2vec',
'ner']
 Initialized pipeline

============================== Training pipeline

==============================
 Pipeline: ['tok2vec', 'ner']
 Initial learn rate: 0.001
E    #       LOSS TOK2VEC  LOSS NER  ENTS_F  ENTS_P  ENTS_R  SCORE
---  ------  ------------  --------  ------  ------  ------  ------
  0       0          0.00     59.56    2.01    1.21    5.97    0.02
  0      10          2.95    908.55    0.00    0.00    0.00    0.00
  0      20          9.95    331.72    0.00    0.00    0.00    0.00
  0      30          5.43    225.39    0.00    0.00    0.00    0.00
  0      40          5.80    249.08    0.00    0.00    0.00    0.00
  0      50          5.15    180.22    0.41   16.00    0.21    0.00
  0      60          5.81    211.74   15.61   60.96    8.95    0.16
  0      70          7.62    216.03   14.70   46.77    8.72    0.15
  0      80          6.44    192.53   26.61   59.67   17.12    0.27
  0      90          7.78    213.47   33.83   52.75   24.90    0.34
  0     100         14.64    287.79   38.94   54.02   30.45    0.39
... Truncated output ...
  1    1500         41.66     88.24   96.82   97.08   96.57    0.97
  1    1510         16.13     48.39   96.73   96.63   96.83    0.97
  1    1520         34.08     80.37   96.69   96.60   96.78    0.97
  1    1530         25.21     67.29   96.26   96.40   96.13    0.96
  1    1540         25.00     65.59   96.38   96.31   96.44    0.96
  1    1550         29.86     86.56   96.22   96.52   95.92    0.96
  1    1560         33.56     86.39   96.06   95.86   96.26    0.96
  1    1570         41.33     87.85   95.92   95.92   95.92    0.96
  1    1580         27.73     86.70   95.72   95.95   95.50    0.96
```

```
  1    1590         33.54     79.62   96.41   96.22   96.60    0.96
  1    1600         22.21     74.79   96.49   96.61   96.36    0.96
 Saved pipeline to output directory
training/model-last
```

[48]:
```
!python -m spacy project run evaluate
```

```
================================== evaluate

==================================
Running command: /home/anibal/gitsources/tesis-
estadistica/03-Experimentos/.env/bin/python -m spacy evaluate training/model-
best corpus/test.spacy --output training/metrics_0.0.1.json
[W Module.cpp:482] Warning: Disabling benchmark mode for MIOpen is NOT
supported. Overriding value to True (function operator())
 Using CPU
 To switch to GPU 0, use the option: --gpu-id 0

================================== Results

==================================

TOK     100.00
NER P   95.85
NER R   95.39
NER F   95.62
SPEED   26125


============================== NER (per type)

==============================

              P       R       F
SAMBL     91.76   91.26   91.51
CAMB      95.16   92.27   93.69
NUM       97.81   97.19   97.50
CUANT     98.66   98.33   98.50
LEGEST    97.59   97.98   97.79
FECHA     94.08   96.95   95.50
ACC       95.80   98.49   97.12
ILAMB     86.47   80.99   83.64
ILEG      97.42   97.42   97.42
SUJ       94.55   89.67   92.05
ACCAMB    95.31   94.86   95.08
EST       93.84   95.80   94.81
LUGAR     93.62   97.78   95.65
SA        88.89   88.89   88.89
VAMB      93.33   77.78   84.85
```

```
PORC     100.00   100.00   100.00
ADMB      90.00    81.82    85.71

 Saved results to training/metrics_0.0.1.json
```

```
[ ]:
```

Wilcoxon Test results

Tables 1, 3 and 5 shows the output for the Wilcoxon Test from the k-Fold Cross-validation analysis.

Table 1: Wilcoxon test for Precision score from k-Fold Cross-validation analysis.

Score	$group_1$	$group_2$	n_1	n_2	statistic	p	p.adj	signif.
Precision	K-1	K-2	17	17	166.5	0.459	1.000	ns
Precision	K-1	K-3	17	17	167.0	0.449	1.000	ns
Precision	K-1	K-4	17	17	131.0	0.654	1.000	ns
Precision	K-1	K-5	17	17	191.0	0.113	1.000	ns
Precision	K-1	K-6	17	17	96.5	0.102	1.000	ns
Precision	K-1	K-7	17	17	175.0	0.301	1.000	ns
Precision	K-1	K-8	17	17	165.5	0.480	1.000	ns
Precision	K-1	K-9	17	17	138.5	0.850	1.000	ns
Precision	K-1	K-10	17	17	163.0	0.535	1.000	ns
Precision	K-2	K-3	17	17	148.0	0.918	1.000	ns
Precision	K-2	K-4	17	17	117.5	0.361	1.000	ns
Precision	K-2	K-5	17	17	163.5	0.524	1.000	ns
Precision	K-2	K-6	17	17	83.0	0.035	1.000	ns
Precision	K-2	K-7	17	17	159.0	0.630	1.000	ns
Precision	K-2	K-8	17	17	145.5	0.986	1.000	ns
Precision	K-2	K-9	17	17	126.0	0.535	1.000	ns
Precision	K-2	K-10	17	17	143.0	0.973	1.000	ns
Precision	K-3	K-4	17	17	115.0	0.317	1.000	ns
Precision	K-3	K-5	17	17	154.0	0.757	1.000	ns
Precision	K-3	K-6	17	17	85.5	0.044	1.000	ns
Precision	K-3	K-7	17	17	150.5	0.850	1.000	ns
Precision	K-3	K-8	17	17	138.5	0.850	1.000	ns
Precision	K-3	K-9	17	17	125.5	0.524	1.000	ns
Precision	K-3	K-10	17	17	146.5	0.959	1.000	ns
Precision	K-4	K-5	17	17	191.0	0.113	1.000	ns
Precision	K-4	K-6	17	17	122.5	0.458	1.000	ns
Precision	K-4	K-7	17	17	182.5	0.196	1.000	ns
Precision	K-4	K-8	17	17	173.0	0.334	1.000	ns
Precision	K-4	K-9	17	17	159.5	0.617	1.000	ns
Precision	K-4	K-10	17	17	175.5	0.293	1.000	ns
Precision	K-5	K-6	17	17	74.5	0.017	0.747	ns
Precision	K-5	K-7	17	17	133.5	0.718	1.000	ns
Precision	K-5	K-8	17	17	121.5	0.438	1.000	ns
Precision	K-5	K-9	17	17	110.5	0.249	1.000	ns
Precision	K-5	K-10	17	17	130.5	0.642	1.000	ns
Precision	K-6	K-7	17	17	209.5	0.026	1.000	ns
Precision	K-6	K-8	17	17	199.5	0.060	1.000	ns
Precision	K-6	K-9	17	17	183.0	0.190	1.000	ns
Precision	K-6	K-10	17	17	211.0	0.023	1.000	ns
Precision	K-7	K-8	17	17	137.5	0.823	1.000	ns
Precision	K-7	K-9	17	17	118.5	0.380	1.000	ns
Precision	K-7	K-10	17	17	139.5	0.877	1.000	ns
Precision	K-8	K-9	17	17	129.5	0.617	1.000	ns
Precision	K-8	K-10	17	17	147.5	0.931	1.000	ns
Precision	K-9	K-10	17	17	165.5	0.480	1.000	ns

Table 3: Wilcoxon test for Recall score from k-Fold Cross-validation analysis at 95% confidence.

Score	$group_1$	$group_2$	n_1	n_2	statistic	p	p.adj	signif.
Recall	K-1	K-2	17	17	198.5	0.065	1.000	ns
Recall	K-1	K-3	17	17	149.0	0.890	1.000	ns
Recall	K-1	K-4	17	17	146.5	0.959	1.000	ns
Recall	K-1	K-5	17	17	201.0	0.054	1.000	ns
Recall	K-1	K-6	17	17	117.5	0.361	1.000	ns
Recall	K-1	K-7	17	17	167.5	0.438	1.000	ns
Recall	K-1	K-8	17	17	134.5	0.743	1.000	ns
Recall	K-1	K-9	17	17	152.5	0.796	1.000	ns
Recall	K-1	K-10	17	17	174.5	0.309	1.000	ns
Recall	K-2	K-3	17	17	100.5	0.134	1.000	ns
Recall	K-2	K-4	17	17	90.5	0.065	1.000	ns
Recall	K-2	K-5	17	17	141.0	0.918	1.000	ns
Recall	K-2	K-6	17	17	70.5	0.011	0.508	ns
Recall	K-2	K-7	17	17	108.0	0.215	1.000	ns
Recall	K-2	K-8	17	17	83.5	0.037	1.000	ns
Recall	K-2	K-9	17	17	98.5	0.117	1.000	ns
Recall	K-2	K-10	17	17	127.5	0.570	1.000	ns
Recall	K-3	K-4	17	17	140.0	0.890	1.000	ns
Recall	K-3	K-5	17	17	185.5	0.163	1.000	ns
Recall	K-3	K-6	17	17	114.0	0.301	1.000	ns
Recall	K-3	K-7	17	17	156.0	0.705	1.000	ns
Recall	K-3	K-8	17	17	128.0	0.581	1.000	ns
Recall	K-3	K-9	17	17	142.5	0.959	1.000	ns
Recall	K-3	K-10	17	17	168.5	0.418	1.000	ns
Recall	K-4	K-5	17	17	198.0	0.068	1.000	ns
Recall	K-4	K-6	17	17	119.5	0.399	1.000	ns
Recall	K-4	K-7	17	17	166.0	0.469	1.000	ns
Recall	K-4	K-8	17	17	137.0	0.809	1.000	ns
Recall	K-4	K-9	17	17	148.5	0.904	1.000	ns
Recall	K-4	K-10	17	17	177.0	0.270	1.000	ns
Recall	K-5	K-6	17	17	72.0	0.012	0.531	ns
Recall	K-5	K-7	17	17	112.5	0.278	1.000	ns
Recall	K-5	K-8	17	17	91.5	0.070	1.000	ns
Recall	K-5	K-9	17	17	109.0	0.228	1.000	ns
Recall	K-5	K-10	17	17	122.5	0.459	1.000	ns
Recall	K-6	K-7	17	17	193.0	0.098	1.000	ns
Recall	K-6	K-8	17	17	163.0	0.535	1.000	ns
Recall	K-6	K-9	17	17	174.5	0.309	1.000	ns
Recall	K-6	K-10	17	17	194.5	0.088	1.000	ns
Recall	K-7	K-8	17	17	119.5	0.399	1.000	ns
Recall	K-7	K-9	17	17	128.5	0.593	1.000	ns
Recall	K-7	K-10	17	17	155.5	0.718	1.000	ns
Recall	K-8	K-9	17	17	155.5	0.717	1.000	ns
Recall	K-8	K-10	17	17	181.5	0.209	1.000	ns
Recall	K-9	K-10	17	17	166.5	0.459	1.000	ns

Table 5: Wilcoxon test for F1 score from k-Fold Cross-validation analysis.

Score	$group_1$	$group_2$	n_1	n_2	statistic	p	p.adj	signif.
F1	K-1	K-2	17	17	194.0	0.092	1.000	ns
F1	K-1	K-3	17	17	163.0	0.535	1.000	ns
F1	K-1	K-4	17	17	142.0	0.945	1.000	ns
F1	K-1	K-5	17	17	191.5	0.109	1.000	ns
F1	K-1	K-6	17	17	108.5	0.221	1.000	ns
F1	K-1	K-7	17	17	168.5	0.418	1.000	ns
F1	K-1	K-8	17	17	157.0	0.683	1.000	ns
F1	K-1	K-9	17	17	159.0	0.634	1.000	ns
F1	K-1	K-10	17	17	182.5	0.196	1.000	ns
F1	K-2	K-3	17	17	114.5	0.310	1.000	ns
F1	K-2	K-4	17	17	97.0	0.105	1.000	ns
F1	K-2	K-5	17	17	140.5	0.904	1.000	ns
F1	K-2	K-6	17	17	71.5	0.013	0.562	ns
F1	K-2	K-7	17	17	115.5	0.326	1.000	ns
F1	K-2	K-8	17	17	105.0	0.179	1.000	ns
F1	K-2	K-9	17	17	116.0	0.339	1.000	ns
F1	K-2	K-10	17	17	134.0	0.734	1.000	ns
F1	K-3	K-4	17	17	125.0	0.513	1.000	ns
F1	K-3	K-5	17	17	176.0	0.286	1.000	ns
F1	K-3	K-6	17	17	102.5	0.153	1.000	ns
F1	K-3	K-7	17	17	152.0	0.812	1.000	ns
F1	K-3	K-8	17	17	138.0	0.838	1.000	ns
F1	K-3	K-9	17	17	144.0	1.000	1.000	ns
F1	K-3	K-10	17	17	167.0	0.454	1.000	ns
F1	K-4	K-5	17	17	191.5	0.109	1.000	ns
F1	K-4	K-6	17	17	118.0	0.370	1.000	ns
F1	K-4	K-7	17	17	174.0	0.322	1.000	ns
F1	K-4	K-8	17	17	160.5	0.593	1.000	ns
F1	K-4	K-9	17	17	168.5	0.418	1.000	ns
F1	K-4	K-10	17	17	184.0	0.182	1.000	ns
F1	K-5	K-6	17	17	75.0	0.016	0.720	ns
F1	K-5	K-7	17	17	121.5	0.438	1.000	ns
F1	K-5	K-8	17	17	107.0	0.205	1.000	ns
F1	K-5	K-9	17	17	119.0	0.394	1.000	ns
F1	K-5	K-10	17	17	134.0	0.734	1.000	ns
F1	K-6	K-7	17	17	202.0	0.049	1.000	ns
F1	K-6	K-8	17	17	182.5	0.196	1.000	ns
F1	K-6	K-9	17	17	190.5	0.117	1.000	ns
F1	K-6	K-10	17	17	206.5	0.034	1.000	ns
F1	K-7	K-8	17	17	131.0	0.658	1.000	ns
F1	K-7	K-9	17	17	135.0	0.760	1.000	ns
F1	K-7	K-10	17	17	159.5	0.617	1.000	ns
F1	K-8	K-9	17	17	147.0	0.946	1.000	ns
F1	K-8	K-10	17	17	175.5	0.293	1.000	ns
F1	K-9	K-10	17	17	169.0	0.413	1.000	ns