UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia “Galileo Galilei”

Master Degree in Physics of Data

Final Dissertation

Multilingual taxonomic text classification in an incremental number of languages

Thesis supervisor

Prof. Nicolò Navarin

Candidate

Giulia Campesan

Academic Year 2021/2022

Abstract

Taxonomic text classification is the Natural Language Processing (NLP) branch that aims at classifying text into a hierarchically organized schema. Its applications space from pure research to industrial and commercial purposes. In this thesis, in particular, the attention is focused on the process of developing a multilingual model for hierarchically classifying text independently from its idiom. Comparative analysis is performed to evaluate whose embedding techniques and feature selection methods perform better. Moreover, since in real-life scenarios a multilingual model may be required to support new languages over time, we implement and benchmark a set of techniques, among which are two Continual Learning algorithms, to sequentially extend our network to an incremental number of languages. The experiments carried out display both the solidity and criticalities of the current model and set the basis for further research.

Contents

List of Figures

List of Tables

Introduction

Taxonomic text classification is the Natural Language Processing (NLP) set of techniques that aims at assigning a set of predefined labels to open-ended text, with the peculiarity that labels are organized in a taxonomy where the top categories are broad and they branch out into more specific ones the deeper we go in the taxonomy. It has been gaining attention in recent years due to its commercial application in audience segmentation, in which the Interactive Advertising Bureau (IAB) taxonomy has provided a shared nomenclature for the segments.

Despite representing a milestone in programmatic advertising, the IAB taxonomy consents to collect sensitive personal data of the user and to exploit them for targeted advertising, in contrast with the General Data Protection Regulation (GDPR) European law. To provide a starting point for a privacy-compliant targeted advertisement pipeline, in this work we provide a customized version of the IAB taxonomy, that hides categories related to sensitive data.

The present work deals with the implementation of a multilingual taxonomic text classification algorithm, that is able to categorize web pages in an arbitrary language as one of the nodes in our taxonomy. We propose and implement a deep learning pipeline for multilingual taxonomic text classification, including data collection, data cleaning, features extraction, classification and post-processing.

Moreover, as in real-life scenarios a multilingual model may need to be extended to new languages over time, we focus on the implementation and comparison of different techniques for dealing with the sequential integration of new idioms. In particular, we propose two Continual Learning (CL) techniques as an alternative to re-training from scratch, as its computational cost gets fast unsustainable with the number of required languages, and to fine-tuning on new data only, as this approach dramatically suffers from Catastrophic Forgetting (CF).

The main contribution of this thesis is the extension of the Elastic Weight Consolidation CL algorithm to a taxonomic classifier, thanks to the computation of the approximated Fisher Information matrix for multivariate distributions. Moreover, we implement the temperature scaling algorithm for the calibration of neural networks in the case of multi-output classifiers.

The work illustrated in this thesis originated as a collaboration between the University of Padua and ID Ward Ltd. following the need of extending the existing taxonomic text classifier to the multilingual case, for the goal of extrapolating user interests based on their browsing habits, while staying privacy compliant.

The thesis is structured as follows:

In chapter 1, we present the general task of taxonomic text classification, then we deal with the data collection and data cleaning phase and we conclude with the analysis of the collected data.

In chapter 2, we give an overview of the vector encoding techniques for textual data and then we focus on modern deep learning models for contextual embeddings, such as Bidirectional Encoder Representation from Transformers (BERT), that represents the feature extraction step of our pipeline.

Going to chapter 3, we expose the metrics that will be used to evaluate our models and the probability-based post-processing sorting algorithm, that grants that the predictions of the network correspond to proper nodes of our taxonomy.

In chapter 4, we report the results of the preliminary experiments for the embedder and the text slice selection, that we exploit in the implementation of our deep neural network baseline model. After its training phase, we evaluate it, at first in the Italian and English test dataset.

In chapter 5, we inspect the accuracy-confidence imbalance typical of modern deep neural networks and we exploit the temperature scaling algorithm to obtain a calibrated model.

In chapter 6, we deal with the extension over time of a BERT-based multilingual classifier to an incremental number of languages. In particular, we introduce the framework of Continual Learning and we present two algorithms: Elastic Weight Consolidation and a Knowledge Distillation (KD) inspired method, the Teacher-Student framework. We compare their performance to new language data only fine-tuning and to the training from scratch on the full dataset, taken as an upper bound.

Chapter 1

Dataset

In this chapter, after introducing the general framework of text classification, we present our specific task of taxonomic text classification. Therefore, in section 1.2, we deal with the concept of taxonomy and with the specific IAB taxonomy structure, that has been the starting point for our own taxonomic categories. We then present our dataset in 1.4, both explaining the data collection and data cleaning phases and analyzing its distribution among the categories.

1.1 Text Classification

The Natural Language Processing (NLP) text classification task aims at mapping any form of text, such as articles, sentences or words, to predefined labels, whose nature depends on the application area. Then, the text classification setting can be formulated as the task of assigning a boolean value to each pair $(d_j, c_i) \in \mathcal{D} \times \mathcal{C}$, where $\mathcal{D}$ is a collection of pieces of text and $\mathcal{C} = c_1, \ldots, c_{|\mathcal{C}|}$ is a set of predefined categories. A value of ***True*** assigned to (d_j, c_i) indicates the decision to label the document d_j as belonging to the class c_i, while a value of ***False*** indicates the decision of not categorising d_j as c_i. In particular, if $|\mathcal{C}| = 2$ we are in the framework of binary classification, whereas if $|\mathcal{C}| > 2$ our problem is in the set multi-class text classification. The general task of text classification algorithms is to approximate an unknown target function $\Phi : \mathcal{D} \times \mathcal{C} \to \{\mathbf{True}, \mathbf{False}\}$ that sets the ground-truth for how documents should be correctly classifies. The function Φ, in this case, is called a classifier and we mark as $\tilde{c}_i$ the prediction for the input d_j, that may or may not coincide with the ground-truth labelling.

1.1.1 Multi-label and multi-output text classification

As mentioned above, text classification methods can be clustered based on the number of categories among which the assigned one can be chosen.

Another possible distinction can be performed based on the number k of categories out of $|\mathcal{C}|$ as which each article d_j can be labelled. If $k > 1$ we are in the multi-label text classification framework, in which we are allowing the output categories to be not mutually exclusive.

Finally, in a classification problem, we can have multiple target output variables, i.e. we treat our classifier as able to estimate a series of M target functions $(f_1, \ldots, f_M)$ to predict the corresponding k labels $(\tilde{c}_1, \ldots, \tilde{c}_M)$, in which each $\tilde{c}_i$ can be a vector if $k > 1$.

1.1.2 Taxonomic text classification

The taxonomic text classification goal is to individuate the location of a particular piece of text in a hierarchical schema. For example, an article dealing with an alpine skiing Olympic competition would be labelled hierarchically as 'Sport', 'Olympic Sports', 'Winter Olympic Sports'.

When it is performed by a classifier, classification can be faced in two ways: either we have a single output multi-class network in which there is a specific label for each taxonomy path (in a localist encoding approach) or we can have a multi-output classifier with one distinct predicted value for each layer of the taxonomy (exploiting a distributed encoding setting).

Given a taxonomy, the second approach allows encoding of all the possible paths with much fewer neurons, as it allows to label a truncated path with a 'Nan' flag in exceeding layers and, in general, to have a unique encoding for shared children nodes. In this second approach, the various layers' predictions can be treated as independent or, on the contrary, as correlated. In the first case, we do not impose any a priori hierarchical consistency between tiers, i.e. between taxonomic layers, meaning that we could produce an overall output that does not coincide with any path in the taxonomy. This choice requires a post-processing phase in which coherence is enforced. On the other hand, we could choose to impose coherence in the classifier itself, i.e. by imposing children layers' predictions to be consistent with their parents outcomes. This approach, anyway, is particularly prone to error propagation issues: an error on the first layer would reflect on all the following ones, preventing any possible correction. For this reason, the approach followed in this work is the shallow one, letting the network learn the correlations among the tiers but without any constrained congruity.

1.2 IAB Taxonomy

In this section, we introduce the Interactive Advertising Bureau (IAB) taxonomy, which represented a milestone in programmatic advertising and that will be the starting point for the shaping of our own taxonomy.

Historically, data vendors, Data Management Platforms (DMPs) and analytics providers have used custom taxonomies to describe and segment their audiences, with little or no standardization across vendors. This results in sometimes drastically different descriptions even for similar data, where vendor A could describe a segment as "Auto-intenders" and vendor B describes a similarly segmented audience as "In-market for cars". Comparability is also made more difficult due to the fact that in house taxonomies reflect a wide range of audience segmentation approaches, including demographic (age, gender, household income, etc), interest-based, purchase-intent based, psychographic, and many more.

With the introduction of IAB Tech Lab's Audience Taxonomy 1.0, the industry now has

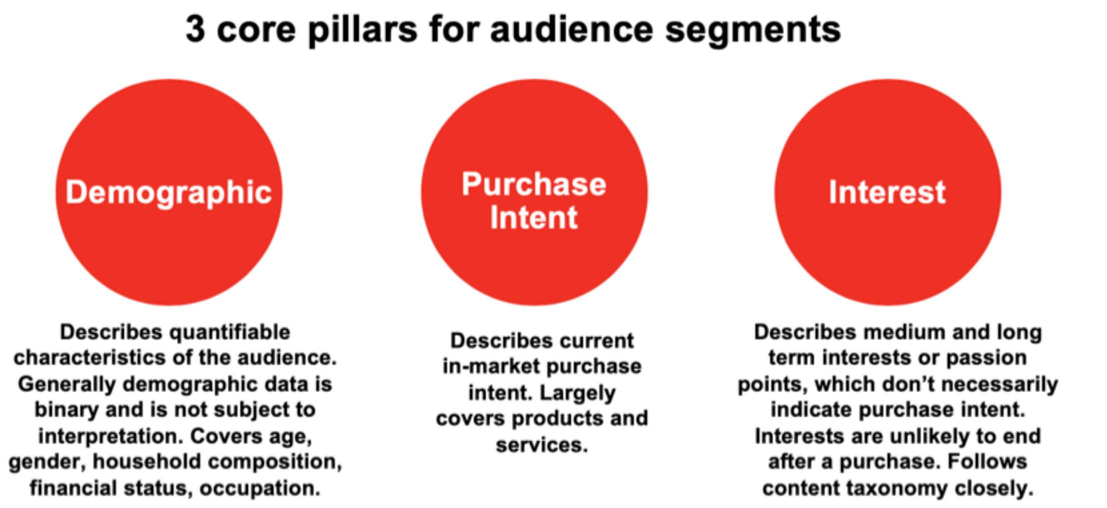

3 core pillars for audience segments
Demographic
Describes quantifiable characteristics of the audience. Generally demographic data is binary and is not subject to interpretation. Covers age, gender, household composition, financial status, occupation.
Purchase Intent
Describes current in-market purchase intent. Largely covers products and services.
Interest
Describes medium and long term interests or passion points, which don't necessarily indicate purchase intent. Interests are unlikely to end after a purchase. Follows content taxonomy closely.

Figure 1.1: Tier 1 categories in IAB taxonomy

a common nomenclature for audience segment names to improve comparability of data across different providers. It is a key element in IAB Tech Lab's Data Transparency Standard, which facilitates consistent labelling of audience data by first-party and third-party sources. The Audience Taxonomy also provides a mechanism to make segmentation approaches much clearer (categorically) by introducing Tier 1 level labelling that designates whether the segment describes attributes that are demographic, interest-based, or purchase-intent based[14]. Original IAB Tier 1 categories are depicted in 1.1. For this work, we focus on the interest sub-taxonomy, neglecting the demographic and purchase intent pillars. From now on, we number the layers of the taxonomy taking into account only the interest segment, which is built upon 4 hierarchical layers. Its first tier (T1) consists of 29 different interests area, such as Sports, Travel, Academic interests, Shopping, some of whose branch out into more specific nodes as we go deeper in the taxonomy. The second tier (T2) is made up of 264 sub-categories, that give rise to the 173 tier 3 (T3) classes. Finally, in tier 4 (T4), we find 27 nodes distributed among 3 of the 27 top layer categories. However, it has to be noticed that some of the nodes in this taxonomy are linked to sensitive subjects: having *Medical Health* or *Elections* would end up targeting the user based on sensitive personal data. For this reason, we have applied a reshaping to the taxonomy to have a model that is unable to tackle sensitive personal data.

1.3 Taxonomy customization

We obtain our own taxonomy starting from the IAB interest segment, removing the categories that are marked as 'personal sensitive data' by the GDPR[7]:

- personal data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs;
- trade-union membership;
- genetic data, biometric data processed solely to identify a human being;
- health-related data;

- data concerning a person's sex life or sexual orientation.

Moreover, also categories that may represent potential triggers, such as *Gambling*, *Smoking*, etc. are shadowed. The removal has to be treated with care: removing from the dataset all the articles that are labelled as one of these topics would lead to a model that is, in fact, not able to recognize them and that would simply assign them to a random category. In order to have a model that can discern touchy subjects, we assign to those particular articles the label 'Other' in all 4 tiers.

1.4 Dataset

1.4.1 Data collection

The request that our dataset should distribute following an imposed taxonomy requires an accurate data collection phase, that permits to obtain a sufficiently large and diverse set of articles and their related taxonomic labels. Despite the automatized classification of a random article into a broad taxonomy remaining challenging, the task of collecting a series of related articles for a given node is rather simple. The most straightforward method is to enter a query consisting of the name of the category in a modern search engine such as Google. This can be provided by various third-party services, among which we select SERP API, which allows the retrieval of up to 100 page results for up to 10.000 queries a month, which fits the needs of this thesis. This crawler can obtain the top Google results for an arbitrary list of topics, that in our case consists of all the possible paths of the taxonomy. After the crawl, we obtain, for each path, a list of raw links that we use to retrieve the HyperText Markup Language (HTML) file. The links we actually obtain are a subset of the retrieved ones, as the crawler will retain only those links that are included in the initial SERP list. Therefore, the maximum number of links per category is 100. After collecting the web pages, we need to extract the information contained in the HTML files, which are not ready to be used for training since they contain tags and non-textual objects, such as images and video. To retrieve the cleaned text in the page we use the BeautifulSoup package from Python.

1.4.2 Data Cleaning

The cleaned text from an HTML page is a good starting point for the text classification task, but it still needs to go through a data cleaning phase to ensure higher data quality. At first, the text is lowercased, as most embedders do not distinguish between cased and uncased words. Punctuation and special characters, such as '\n', '\t' are removed. To bring the focus on the relevant words for the topic of the article, all the stopwords, i.e. the most popular words in the considered languages, such as prepositions, adverbs, articles, etc. are discarded. Finally, the text goes through lemmatization, in which inflected words are reduced to their root form. For example, the words *cooked*, *cooking*, *cooker* are mapped to the common root *cook*.

1.4.3 Data Overview

After the data cleaning phase, we can have an overview of the data statistics and of its distribution among the various levels of the taxonomy. The Italian dataset is built of

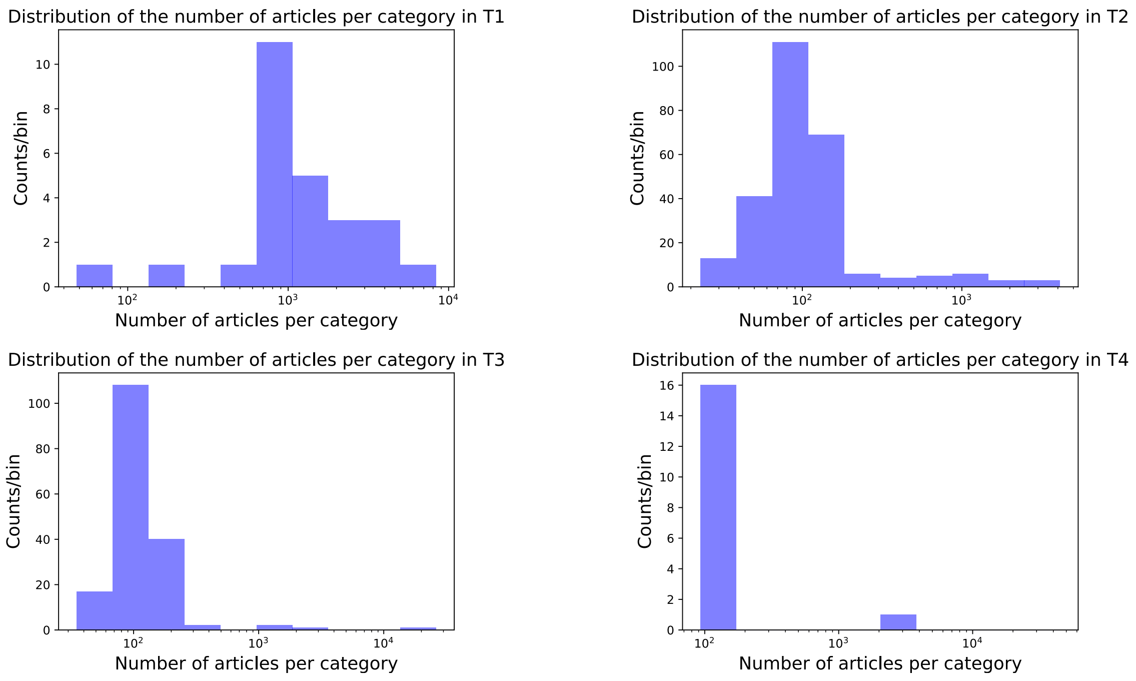


Figure 1.2: Histograms of the number of articles per category in each tier: Tier 1 (top left), Tier 2 (top right), Tier 3 (bottom left), Tier 4 (bottom right)

23, 749 HTMLs, the English one of 21, 859, 25, 067 for Spanish and 26, 796 for Swedish. For each sub-dataset in a specific language, we reserve its 20% for validation and another 10% for testing.

The average number of articles per category has a strong dependency on the depth of the tier in the taxonomy: in particular, articles category are assigned via inheritance, meaning that all documents from child nodes will count in the statistics of their parent node. Moreover, not all parent nodes have the same number of child nodes, and this is again reflected in the distribution of the number of articles among categories. We can see the histogram of the number of articles per category in figure 1.2. What we observe is that the mode of the distribution is around 10^3 for Tier 1, but it drops to 10^2 for subsequent tiers: this is justified by the inheritance concept exposed before. Moreover, we can see that for both Tier 3 and Tier 4 we have an outlier, represented by a single bin with particularly high counts: this is the 'Nan' category for the corresponding Tier, as most of the articles have a taxonomic depth lower than Tier 3.

We display in figure 1.3 the ratio of articles per category for the specific categories of, respectively, the first and the fourth tiers.

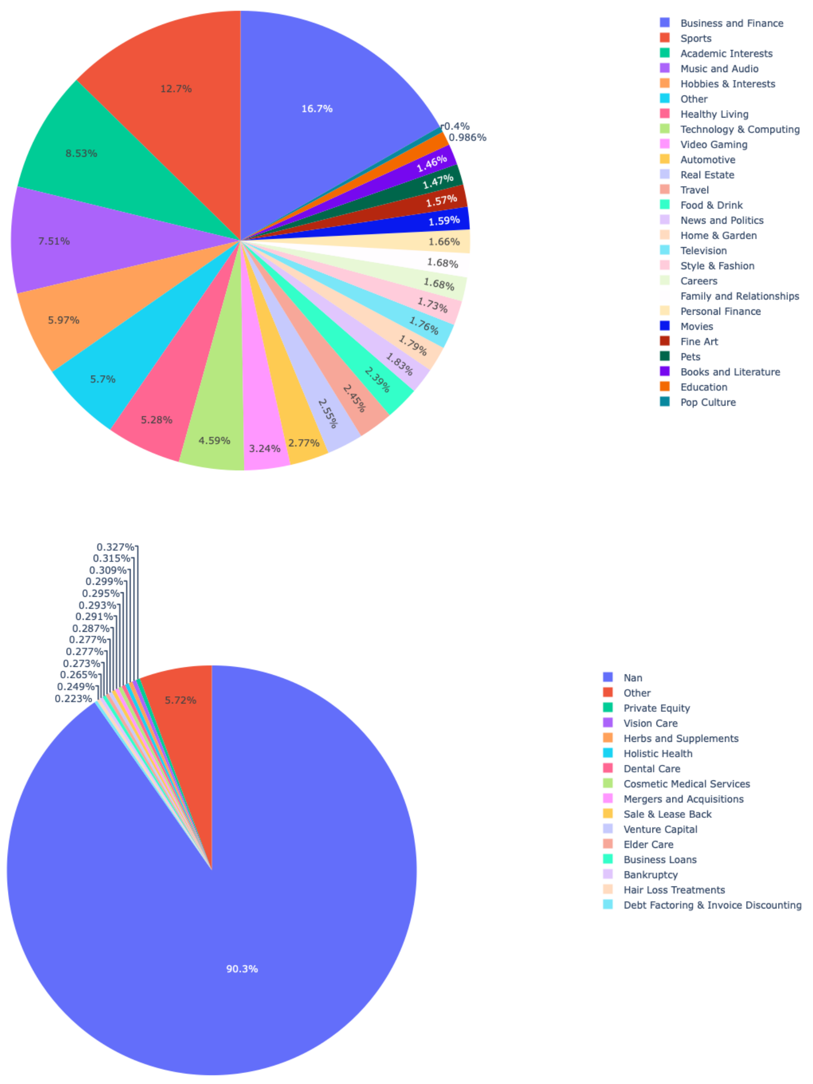


Figure 1.3: Distribution of articles among specific categories for Tier 1 (top) and Tier 4 (bottom)

Chapter 2

Word Embeddings

Word Embeddings count among the techniques for numerical encoding of words, which represents the core of feature extraction procedure in NLP. In particular, they build a vector representation of terms, in order to exploit vector algebraic operations and properties, such as distance, that allow encoding similarities. The most naive procedure for word-vector mapping consists of one-hot encoding each word in a $\mathbb{R}^V$ basis vector, being $|V|$ the size of the vocabulary. This leads to severe sparsity and prevents the encoding from being performed into a fixed-length vector as the vocabulary size enlarges[21]. The key property of embedding algorithms is then the ability to map semantic similar words close to each other[25], leading to a continuous representation in a vector space $\mathbb{R}^D$, being D independent from the vocabulary size and much smaller than it. This property relies on the so-called distributional hypothesis, claiming that words with similar surrounding words (context) share the same meaning. A complete definition of word embeddings describes them as dense, distributed, fixed-length word vectors, built using word co-occurrence statistics as per the distributional hypothesis[1]. The most famous example of the capability of embeddings to capture semantic analogies by algebraic operations, such as translation, is expressed by the equation $vec('King') - vec('Man') + vec('Woman') \approx vec('Queen')$[24]. A visual representation of these vector properties of embedded spaces is offered in figure 2.1. The notation used here implies that $vec('Word')$ is the embedded vector representation of Word and the approximation in the equation indicates that the result of the operations is a vector that is closer to the representation of Queen than of any other word.

Word embeddings can be clustered into two main classes, depending on the type of algo-

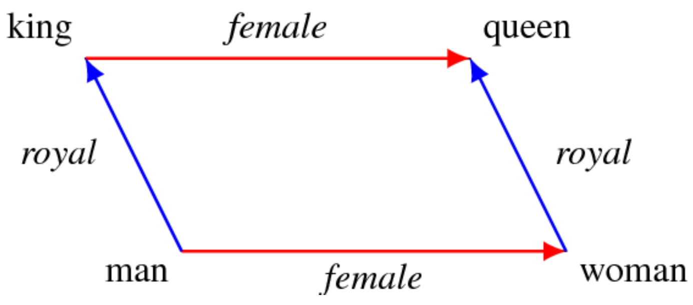


Figure 2.1: Word embedding vector representation

rithm used to build them: prediction-based models and count-based models. Techniques that rely upon neural networks are grouped into the *prediction-based* cluster since they usually learn to try to predict the following word based on its neighbours, whereas *count-based* methods exploit word-context co-occurrence statistics. Both prediction-based and count-based algorithms belong to the unsupervised learning framework.
Among the count-based class, it is worth mentioning GloVe[31], which has been the first to consider the ratios of co-occurrences instead of raw counts. In the prediction-based group, some break-through models are the continuous bag-of-words (CBOW)[25] and skip-gram designs (SG)[26], creating the Word2Vec representation, and the *FastText* algorithm presented in [3] and [15].

All the techniques in the list above provide non-contextual word vectors: each word is mapped to a single vector, despite it may have different meanings in different contexts, e.g. the word *right* in 'go right at the roundabout' and 'you are always right' would be mapped to the same vector. To provide contextual meaning to embeddings, LSTM-based or Attention-based architectures, like Bidirectional Encoder Representations from Transformers (BERT), can be exploited. These architectures belong to the wider field of Deep Learning models for text representation. In section 2.3, an overview of this kind of techniques is given, starting from feed-forward neural networks and leading to the explanation of state-of-the-art models.

2.1 Count-based embedders

Count-based embedders rely on word-context co-occurrence distribution to retrieve the vector representation. In fact, word-word co-occurrence statistics are the first source of information for any unsupervised language model. Here, we focus our interest on the Global Vectors for model representation (GloVe) algorithm, which represented a break-though innovation in its field via the intuition that it is the ratio of co-occurrence with a third word to determine the strength of the relation between a word and the context (i.e. another word in the vocabulary).

2.1.1 GloVe

Given a word-context framework, we can set a quantitative description of it by introducing the word counts matrix X, having as entries X_{ij} the counts of how many times the word j appears in the context of i. As it could be inferred, the index i and j run from 1 to the size of the vocabulary $|V|$. Then, it is possible to obtain the probability of the words j and i to appear jointly as $P_{ij} = P(j|i) = X_{ij}/X_i$, in which $X_i = \sum_j X_{ij}$ are the cumulative occurrences of any word j in the context of i. Having set the framework, we now show how the ratio between the co-occurrence probabilities of i and j with a third word k reveals the strength of their relation[31]. Let us consider an example that can give insight into the process. In the field of information technology, we focus our attention on the words $i = \textit{signal}$ and $j = \textit{gate}$. If the third word considered k is *smoke*, that is related to i but not to j, the ratio P_{ik}/P_{jk} should be high. On the other hand, when dealing with $k = \textit{house}$, which shares context with *gate* but not with *signal*, the same ratio should be much lower. Finally, when taking into account *electronic*, P_{ik}/P_{jk} should approach 1. This allows building a shared context between a couple of words via inspecting their co-occurrences ratio. This basic idea is then inserted in the least squares problem for a log-linear model setting and gives us the cost

$$J = \sum_{i,j}^{|V|} f(X_{ij}) \left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - log(X_{ij}) \right)^2 ,$$

where f is a weighting function and $w, \tilde{w}$ are word and context vectors. Full details about the model derivation can be found in [31].

After giving the idea behind GloVe, as a representative of count-based embedders, we deal with the class of prediction-based embedding models.

2.2 Prediction-based embedders

Prediction-based models are neural-based embedders, that instead of exploiting global information, such as corpus-wide co-occurrences distribution, exploit the local context of words.

2.2.1 Word2Vec

One of the two most popular algorithms, proposed by Mikolov et al.[25], are the Continuous Bag Of Words (CBOW) and the Skip-Gram, which groups into the Word2Vec embedding models. For both algorithms, the framework is given by a sequence of words $(w_1, w_2, \ldots, w_T)$ and a single-layer neural network. In the case of CBOW, the objective of training is to learn to guess the k_{th} word w_k in the sequence based on the surrounding ones $(w_1, \ldots, w_{k-1}, w_{k+1}, \ldots, w_T)$. In the other hand, Skip-Gram aims at predicting the context, i.e. words before and after a word, given the central item. In both cases, learning is achieved by maximizing the average log-probability of subsequent terms, that is formulated, taking Skip-gram as an example, as[26]:

$$\frac{1}{T} \sum_{t=1}^{T} \left(\sum_{j=-c}^{c} \log p \left(w_{t+j} | w_t \right) \right) \tag{2.1}$$

where $c = c(w_t)$ is (half) the size of the context window. An efficient simplified representation of the two can be seen in fig **mettere figura**

This overall structure gave rise also to another popular model, proposed by [3] and [15], and known as the Facebook FastText library, which will be mastered in the next section.

2.2.2 FastText

The FastText model develops starting from one of the drawbacks of the Skip-gram model: it considers words as its atomic unit, neglecting their internal structures. This leads to a potential loss of information, in particular for languages that make extent usage of composed words, such as Finnish and Turkish, in which sub-parts of a single word could help in building the representation of out-of-corpus words[1]. Morphological representation of words is achieved by considering each word in the corpus as a bag of $n - grams$, i.e. as the set of its sub-sequences of size n. In order to allow to infer morphological patterns, such as prefixes and suffixes, two special characters, $<$ and $>$, are introduced, to mark, respectively, words beginning and end. Taking as example the word $w = stars$ and $n = 3$, its tri-gram representation is $< st$, sta, tar, $ar >$ and the full sequence $< star >$. Then, its Bag Of Characters (BOC) $\mathcal{G}$ will be given by all its $n - grams$ with $3 \leq n \leq 6$. This allows to build the full word vector as the sum of its BOC items vectors $\mathbf{z}_g$ and to define the scoring function $s(w, c)$ between w and an instance c in its context, with vectorial representation $\mathbf{v}_c$, as[3]:

$$s(w, c) = \sum_{g \in \mathcal{G}} \mathbf{z}_g^T \mathbf{v}_c$$

This scoring function is exploited to build the conditional probability presented in 2.1, from a softmax distribution

$$p(w|c) = \frac{e^{s(w,c)}}{\sum_{j=1}^{|V|} e^{s(w,j)}}$$

Via morphological analysis of seen instances, FastText models can provide a reliable representation for rare or unseen words.

After setting a general framework for textual data embedding techniques, in the next section we present how to exploit them for a series of tasks, such as machine translation, question answering, part-of-speech-tagging, language generation and text classification. Among them, we are interested in the last one, with a particular focus on the multilingual case.

2.3 Deep Learning for Text Data

2.3.1 Feed-forward Neural Networks

Feed-forward neural networks count among the simplest algorithms for textual data handling. In particular, their application is straightforward for the task of text classification. After being fed with the vector sum or average of word vectors obtained via one of the techniques presented in the previous paragraphs, it propagates the input through a series of d feed-forward hidden layers, whose neurons may be activated depending on the trainable weights W of their inputs values and on the chosen activation function σ. The last layer exploits the same computation schema to retrieve the predicted label.

$$\begin{aligned} x &= \frac{\sum_i emb(w_i)}{N} \\ h_1 &= \sigma_1(W_1 x + b_1) \\ h_j &= \sigma_j(W_j h_{j-1} + b_j) \\ \hat{y} &= \sigma_o(W_{d+1} h_d + b_{d+1}) \end{aligned}$$

in which:

- $emb(w_k) \in \mathbb{R}^n$ is the embedded vector of the k_{th} word out of N in the textual frame
- $x \in \mathbb{R}^n$ is the input layer
- $h_j, j = 1, \ldots, d$ is the j_{th} hidden layer, connected with the $(j-1)_{th}$ via the weights matrix W_j and biased by b_j
- σ_j, σ_o are respectively the element-wise activations functions of the j_{th} hidden layer and of the output layer

- $\hat{y}$ is the predicted output label

2.3.2 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are directed acyclic graphs that are able to capture the sequential structure of textual data, by exploiting its locally correlated nature. The input is now treated like a time series instead of a static vector. Previous time step information is retained by introducing a context layer U_h, that enriches the current input instance, e.g. a word in a sentence, with the foregoing hidden vector. They can be described as feed-forward neural networks thanks to their acyclic nature:

$$
\begin{aligned}
h_t &= \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \\
y_t &= \sigma_y(W_y h_t + b_y)
\end{aligned}
$$

RNNs field of application is wide, including machine translation, speech recognition, text summarization and classification. Their main drawback is that their performance degrades with long sequences due to the gradient vanishing problem they suffer from. In order to catch long-term dependencies, a more sophisticated structure is needed. In Long-Short Term Memory (LSTM) networks, the issue is faced by building the network with cells provided with a set of gated units.

2.3.3 Long-short term memory

Long-Short Term Memory networks (LSTMs) can detain long-term dependencies thanks to their gated recurrent units structure that can discriminate how much past information should be retained and how far it should be propagated.

The core element of LSTMs is the cell state C_t, which allow keeping information barely unchanged through the chain of units, by displaying only minor linear interactions with the gates[30]. Let us inspect how past and current information is handled by the repeating unit, displayed in 2.2.

The set of gates counts an input gate i_t, a forget gate f_t and an output gate o_t. Each of them can alter the cell state through a sigmoid neural net layer and multiplicative connections. The forget gate controls which pieces of past information are to be thrown away via a sigmoid layer having trainable weights W_f, b_f and taking as input the previous hidden state h_{t-1} and the current input x_t:

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

The input gate controls which values are going to be updated with new information flow, via a sigmoid layer, and set the upgrade magnitude $\tilde{C}_t$ with a hyperbolic tangent layer:

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

The cell state can then be notified as:

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

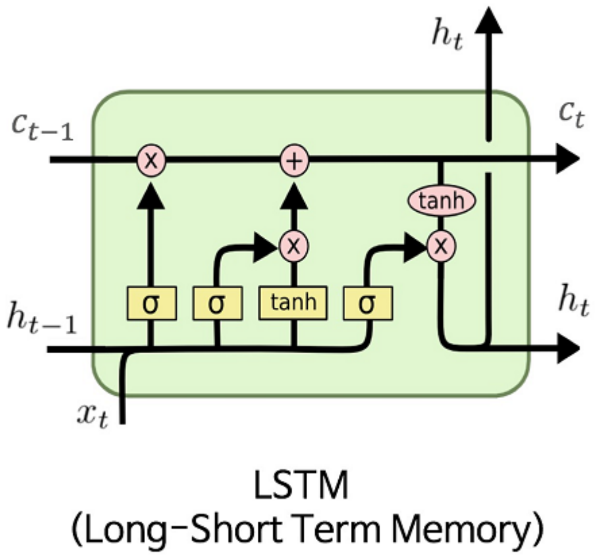


Figure 2.2: Long-Short Term Memory unit cell

Finally, a new hidden state is created, which will be the input of the next cell together with the next time step of the sequence. The cell state C_t is masked by the sigmoid layer of the output gate and normalized by an hyperbolic tangent layer.

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

LSTMs thanks to their capabilities to infer time series correlation structures are applied in a series of NLP tasks, among which is text classification.

2.3.4 Bidirectional Encoder Representation from Transformers (BERT) based architectures

Bidirectional Encoder Representation from Transformers (BERT)[6] is a pre-trained language model, meaning that, after being trained on a general purpose large corpus, it can be exploited as a front-end structure for various NLP tasks, by interfacing it with the proper back-end. For the task of text classification that we are interested in, the back-end includes at least one dense layer with neurons one-hot encoding each one of the possible categories.

As already mentioned, the main difference between BERT and the other language vectorial representation techniques presented in 2.2 is that BERT provides contextual embedding for words, i.e. it maps homonyms to different regions of the embedding space, depending on their semantic field determined by their context. Due to the fact that the embeddings are not determined a-priori for a given word, we can not just produce the vectors for each word in the dictionary and store them in a database, neglecting the model that produced them, but we need to include the pre-trained model for any new piece of text we want to operate on.

After giving a qualitative insight of how BERT works, in the next section we get a quantitative explanation of how it is able to provide bidirectional context to generate embeddings.

2.3.4.1 BERT

BERT architecture is derived from Transformers, which rely on the Attention mechanism to draw global dependencies between input and output. The main structural difference is that BERT exploits only the encoder side of Transformers, that is what is actually needed to build a general language representation. It is designed to pre-train deep bidirectional representations from unlabeled textual data by joint inference from left and right context. The unidirectionality limitation is overcome by using a 'masked language model' (MLM), inspired by the Cloze task technique[34]. During the training procedure the 15% of the input sentences is masked via substitution with the [MASK] token. The input text is prepared using WordPiece embeddings with a vocabulary of 30000 tokens. Thanks to the built-in contextual framework, the number of differences to introduce in the architecture to perform several downstream tasks, such as text classification, is minimal, with respect to the plain embedder: as a baseline, it is enough to add Dense layers to the BERT front-end.

2.3.4.2 Transformers

As introduced before, BERT is built as a multi-layer bidirectional transformer which relies entirely on an attention mechanism to draw global dependencies between input and output. The transformer allows for significantly more parallelization and has reached state-of-the-art performance on many NLP tasks. It consists of an encoder and a decoder part. Here, the encoder maps an input sequence of symbol representations $(x_1, ..., x_n)$ to a sequence of continuous representations $z = (z_1, ..., z_n)$. Given z, the decoder then generates an output sequence $(y_1, ..., y_m)$ of symbols one element at a time. At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next[36]. The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of figure 2.3 respectively.

As we can see the attention mechanism is a very important feature of the transformer architecture. In simple terms, the attention mechanism is the part of a neural architecture that enables the dynamically highlighting of relevant features of the input data. 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, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key. The original work relies on the Scaled Dot-Product Attention. The input consists of queries and keys of dimension d_k, and values of dimension d_v. The attention function is computed on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V. The Attention function returns the matrix of outputs as:

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

The transformer architecture uses a specific type of attention called multi-head attention. The idea behind it is instead of performing a single attention function, to run multi-

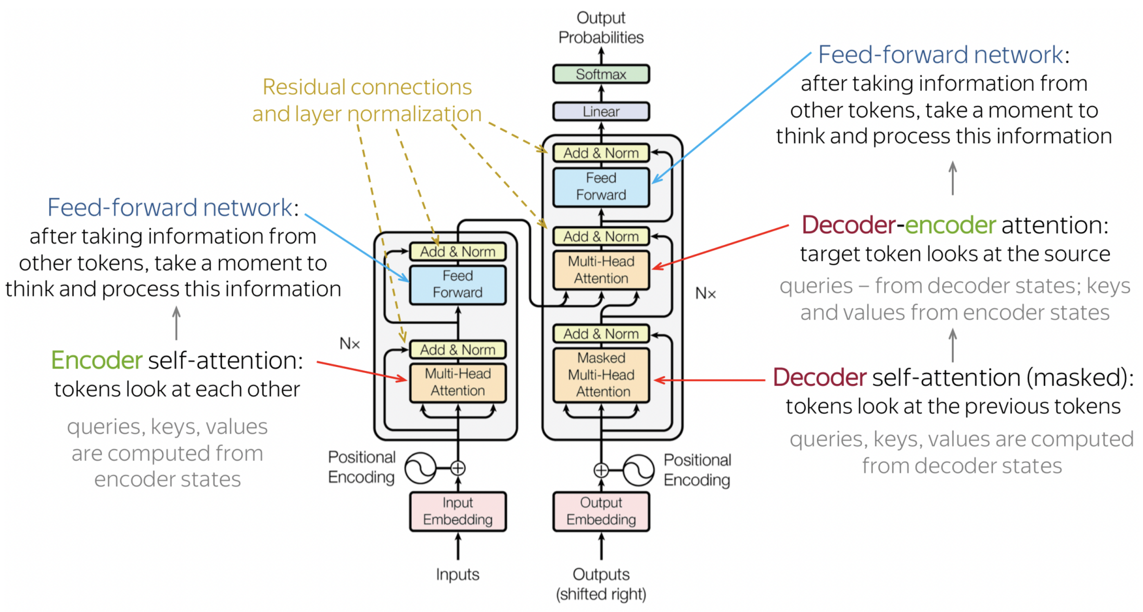

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Positional Encoding
Output Embedding
Outputs (shifted right)
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Input Embedding
Inputs
Residual connections and layer normalization
Feed-forward network: after taking information from other tokens, take a moment to think and process this information
Encoder self-attention: tokens look at each other
queries, keys, values are computed from encoder states
Feed-forward network: after taking information from other tokens, take a moment to think and process this information
Decoder-encoder attention: target token looks at the source
queries – from decoder states; keys and values from encoder states
Decoder self-attention (masked): tokens look at the previous tokens
queries, keys, values are computed from decoder states

Figure 2.3: Transformer Architecture

ple lower-dimensional functions in parallel. The independent attention outputs are then concatenated and linearly transformed into the expected dimension.

2.3.4.3 Multilingual Bert

Multilingual contextual embedders can be obtained by training a BERT architecture with textual data in a set of languages. Tokenization is performed via a shared WordPiece vocabulary. The first multilingual-BERT (m-BERT) model has been implemented by Google and it has been trained on a dataset obtained from the concatenation of monolingual Wikipedia corpora from 104 languages. A competitive alternative has been provided by Facebook with Cross-Lingual-Model Roberta (XLM-Roberta), which has been pre-trained with CommonCrawl data in 100 languages.

As the need to have general pre-trained language models grows, it would be not feasible to train and maintain dozens of single-language embedders. Moreover, the huge variance displayed in the amount of text available in different languages would be reflected in lower performances for mono-lingual models for less represented languages.

This issue is overcome by the m-BERT model with the so-called cross-lingual transfer learning, in which knowledge is observed to be transferred from one natural language to another. The cross-lingual generalization ability of m-BERT is based on a combination of factors: shared vocabulary items that act as anchor points, joint training across multiple languages that spreads this effect, which ultimately yields deep cross-lingual representations that generalize across languages and tasks[2].

In figure 2.4, we can see the 2D t-SNE (see 8 for more details) projection of the embedded vectors produced from our Italian and English text samples by the fine-tuned *multi-qa-MiniLM-L6-cos-v1* embedder.

It is interesting to inspect the spatial distribution of data belonging to the same class.

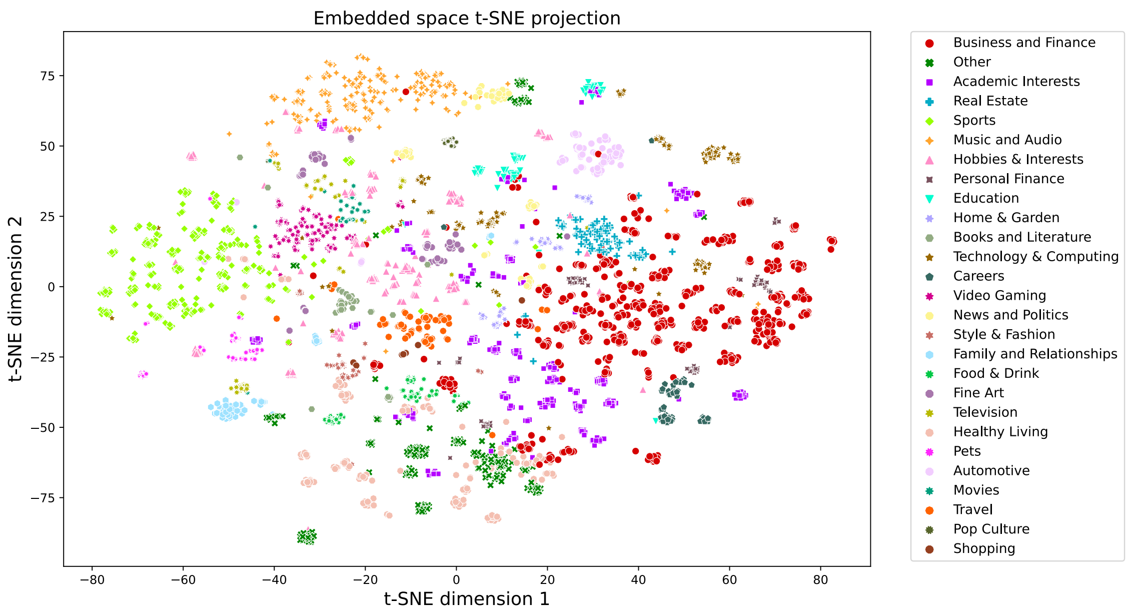


Figure 2.4: Embedded space 2D projection via t-SNE algorithm

We can spot two different behaviours. For some classes, the embedded data items form a proper cluster, being concentrated in a specific region of the 2D space. This is the case for example of *Automotive*, *Music and Audio*, *Real Estate* and *Video Gaming*, that are categories related to a specific topic each. Other classes are spread over wide regions of the t-SNE latent space: this happens for *Academic Interests*, *Hobbies & Interests* and *Other*, whose articles spread over various semantic fields.

Chapter 3

Post-processing and evaluation

3.1 Metrics

In this section both general and taxonomy-specific metrics will be introduced, in order to set a quantitative comparison between models.

3.1.1 General metrics

To evaluate models, several well-known supervised learning metrics have been exploited, such as:

- **Accuracy**: accuracy represents the ratio of correct predictions out of the total number of samples
$$ACC(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\left(\tilde{\mathbf{y}}_i = \mathbf{y}_i\right)$$
- **Binary classification metrics**
  - True Positives (TP): number of samples that belong to the positive class and are correctly classified as positive
  - True Negatives (TN): number of samples that belong to the negative class and are correctly classified as negative
  - False Positive (FP): number of samples that belong to the negative class and are incorrectly classified as positive
  - False Negatives (FN): number of samples that belong to the positive class and are incorrectly classified as negative
- **Precision**: precision is the ratio of instances that are correctly classified as belonging to the positive class out of all the ones that are predicted to be positive
$$PR = \frac{TP}{TP + FP}$$

- **Recall**: recall is the ratio of instances that are correctly classified as belonging to the positive class out of all the actually positive ones.

$$REC = \frac{TP}{TP + FN}$$

- **Mean Average Precision**: gives the average of numerical approximation of the area under the precision-recall curve

$$MAP = \frac{1}{n}\sum_{i=1}^{n} AP(k)$$

  having defined $AP(k) = (REC_k - REC_{k-1})\, PR_k$

As the considered models are multi-output due to the hierarchical nature of the data, these metrics would be evaluated for each tier. Then, these metrics, despite being able to give a general evaluation of the network performance, are not able to encode fundamental properties of a multi-tier classifier, such as hierarchical consistency and full-branch accuracy.

3.1.2 Taxonomy-specific metrics

- **Average First Diversion Ratio**: First Diversion Ratio (FDR) is the number of correctly classified tiers in a multi-output hierarchical prediction, normalized over the taxonomy depth. Then, given an L-layers taxonomy, considering a piece of text classified as $\tilde{\mathbf{y}} = (\tilde{\mathbf{y}}_1, \dots, \tilde{\mathbf{y}}_L)$ while having $\mathbf{y} = (\mathbf{y}_1, \dots, \mathbf{y}_L)$ as true label, FDR is computed as

$$FDR(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{\underset{i}{argmin}\,(\mathbf{y}_i \neq \tilde{\mathbf{y}}_i)}{L}$$

  Its averaged version is obtained by considering the mean over a n-sized set

$$AFDR = \frac{1}{n}\sum_{j=1}^{n} FDR(\mathbf{y}_j, \tilde{\mathbf{y}}_j)$$

- **Average Full Branch Accuracy**: Average Full Branch Accuracy indicates the ratio of samples that are fully correctly classified, from the first to the last tier.

$$AFBA = \frac{1}{n}\sum_{j=1}^{n} FBA(\mathbf{y}_j, \tilde{\mathbf{y}}_j)$$

  having defined

$$FBA(\mathbf{y}_i, \tilde{\mathbf{y}}_i) = \begin{cases} 1 & \text{if } \mathbf{y}_i = \tilde{\mathbf{y}}_i \ \forall i \in 1, \dots, L \\ 0 & \text{otherwise} \end{cases}$$

- **Average Full Tier Accuracy**: Average Full Tier Accuracy (AFTA) indicates the ratio of samples being correctly classified up to, at least, a certain tier. As an example, considering tier T, with $T \leq L$, its AFTA is obtained by the ratio of samples for which the produced labels, for tiers going from the 1-st to the T-th, correspond to the true ones

$$AFTA_T = \frac{1}{n}\sum_{j=1}^{n} FTA_T(\mathbf{y}_j, \tilde{\mathbf{y}}_j),$$

$$FTA_T(\mathbf{y}_i, \tilde{\mathbf{y}}_i) = \begin{cases} 1 & \text{if } \mathbf{y}_i = \tilde{\mathbf{y}}_i \ \forall i \in 1, \dots, T \\ 0 & \text{otherwise} \end{cases}$$

Clearly, for the last tier of the taxonomy, i.e. $T = L$, this metric just reduces to AFBA.

All these metrics are then computed also in the top-K version, meaning that, e.g. when considering top-K AFBA, the prediction is marked as correct when the true label y coincides with one among the first (ordered by joint probability) K full branch prediction $\tilde{y}$.

3.2 Probabilistic Correction

When dealing with taxonomic classification, it has always to be kept in mind that data are labelled following a strict hierarchical structure, meaning that, for a given sample, each single-tier prediction must be consistent with the subsequent ones. Our network, despite being able to learn inter-tier correlations, has no strict coherence constraint. This means that it could assign a tier 2 category that is not compatible with the predicted tier 1, e.g. the network, after being fed with an article, could have marked it as 'Sport' for the first level and as 'Language Learning' for the second one. Clearly, this kind of output is completely nonsense. Therefore, a reasonable way to produce consistent labelling out of the network predictions must be found. It has to be independent of the ground-truth labels since they are not available when facing real-life scenarios.

A first approach could be the one based on majority voting: having multiple hierarchical outputs for the same article, we could exploit just plain statistics and use compatibility between the majority of tiers to individuate a coherent path. The problem with this approach is that it shows useful in a negligible number of cases: as the precision of 53% of items in the test set is up to tier 2, we dispose only of tier 1 and tier 2 to determine the right labelling, that are clearly not enough for majority voting. Another approach could arise from the following observation: as a strong correlation is observed between the network returning a low softmax probability even for the most probable prediction and it being wrong, we could set a probability threshold under which we assume that for that tier the network is mistaken and we rely on the subsequent tier for re-labelling it. Again, this could work fine if the one we assume to be correct is the category assigned to tier 2, as it would be straightforward to identify the unique corresponding tier 1 label. On the other hand, it is not possible to correct the 2nd level by exploiting only tier 1 prediction, as many tier 2 labels correspond to a single tier 1 category. Moreover, this method introduces an empiric threshold, that may strongly depend on the dataset.

All this considered, the chosen approach is based on the joint probability evaluation for all the possible paths in the taxonomy. The framework is the following: given an input sample $\mathbf{x}$, our network D returns as output $D(\mathbf{x}) = (\tilde{\mathbf{y}}, \tilde{\mathbf{p}})$, in which $\tilde{\mathbf{y}} = (\tilde{\mathbf{y}}_1, \dots, \tilde{\mathbf{y}}_L)$,

$\tilde{\mathbf{p}} = (\tilde{\mathbf{p}}_1, \ldots, \tilde{\mathbf{p}}_L)$, having $\tilde{\mathbf{y}}_i = (\tilde{\mathbf{y}}_1, \ldots \tilde{\mathbf{y}}_{n_i})$ as the vector one-hot encoding the n_i categories for the i_{th} tier and $\tilde{\mathbf{p}}_i = (\tilde{\mathbf{p}}_i(\tilde{\mathbf{y}}_1|\mathbf{x}), \ldots, \tilde{\mathbf{p}}_i(\tilde{\mathbf{y}}_{n_i}|\mathbf{x}))$ as the vector of their softmax-produced probabilities.

Given a taxonomy admitted path (T1, T2, T3, T4), it is then possible to compute the joint probability of this set as

$$\tilde{p}(\text{T1, T2, T3, T4}|\mathbf{x}) = \tilde{p}_1(\text{T1}|\mathbf{x})\tilde{p}_2(\text{T2}|\mathbf{x})\tilde{p}_3(\text{T3}|\mathbf{x})\tilde{p}_4(\text{T4}|\mathbf{x}),$$

under the strong assumption of independence of each tier prediction. For each test sample $\mathbf{x}$, it is then possible to sort all the branches based on this joint probability, obtaining the top-K predictions as the K paths associated with the highest joint probabilities.

This algorithm, beyond solving the inconsistency problem between different tiers predictions, is able to find the overall most likely prediction, resulting in an observed increase of both the general and taxonomic metrics.

Chapter 4

Experiments

In this chapter, we expose the experiments that lead to our baseline multilingual classifier. All the models presented in this work have been trained on an NVIDIA®T4 Tensor Core Graphics Processing Unit provided by the CloudVeneto platform. We expose the result obtained from a single run of each experiment due to hardware limitations. However, we are conscious that since the training of a neural network is a stochastic process, it should be repeated to accumulate statistics.

The text classification task consists of two crucial components: features extraction and model selection. At first, we deal with the task of features extraction, that for our case coincide with the choice of the embedder.

4.1 Word embeddings comparison

This first set of experiments has the goal to select the multilingual embedding layer that will constitute the front-end of our classifier. The pre-trained models we compare are the following:

- multi-qa-MiniLM-L6-cos-v1, with 6 layers that maps sentences to 384 dimensional dense vectors,
- paraphrase-multilingual-MiniLM-L12-v2, having 12 layers producing 384-dimensional embeddings,
- universal-sentence-encoder-multilingual, mapping text to a 512D vector space.

The first two trained models are made available by the HuggingFace library and are BERT-based, whereas the second is offered by TensorflowHub and has a convolutional structure.

The overall network for each tested model is constituted by the embedding layer followed by the classification back-end, that for this set of experiments is made up of a Batch Normalization layer followed by 4 Dense layers, whose return the predicted category for each tier.

The training process is the following: at first, we freeze the embedding layer weights and we train the classification back-end layers only. The Cross-Entropy loss is minimized with the ADAMW[19] optimizer with an initial learning rate set to 10^{-3}. The number of epochs is set to 25 with EarlyStopping with patience 1. The batch size is set to 32. We then proceed with the next training step involving the whole architecture, in which we fine-tune the pre-trained embedder for our downstream task. In this case, the initial learning rate for ADAMW is set to 10^5. All other training hyper-parameters are the same as the previous step.

For all the experiments in this section, we feed the network with the title of a given article, as the encoders we test are supposed to work with sentences rather than with long pieces of text. In the following section, we will study how to slice an article to retain the most information from it.

In figures 4.1, 4.2, 4.3, 4.4 we report the training curves displaying the validation accuracy for the 4 tiers over epochs for the three embedders.

The 4 graphs show that the 'paraphrase-multilingual-MiniLM-L12-v2' embedder gains the best performance over all 4 taxonomic layers. However, it has to be considered that the fine-tuning of the embedding layers will be involved in all the experiments conducted in this thesis. Then, due to the limited hardware resources at our disposal, we have to consider the trade-off between accuracy and complexity, i.e. the total number of trainable parameters, for each of the considered embedders. The accuracy-complexity trade-off can be visually inspected in figure 4.5. The metric we consider to determine the performance of an embedder is the AFBA, as it encodes accuracy over the full branch of the taxonomy. From this graph, the limitations of 'paraphrase-multilingual-MiniLM-L12-v2' emerge: it grants the best performance at the cost of high complexity. For this reason, our choice falls on the 'multi-qa-MiniLM-L6-cos-v1' pre-trained model, as it allows reasonable metrics at the lowest complexity value.

4.2 Encoded pieces of text comparison

We now study which slicing procedure retains the most information from an article.

We do so by comparing the performance of the architecture proposed in the previous section, with 'multi-qa-MiniLM-L6-cos-v1' as embedder, over different pieces of text extracted from an article, in particular:

- its title,
- its title followed by the first sentence,
- its title followed by a random sentence from the article,
- just its first sentence
- just the random sentence.

This particular slicing has been chosen for the following reason: it is a widespread pick to make the title contain the keywords related to the article topic, so it is the natural choice when looking for a meaningful piece of an article. However, sometimes the title may be deceiving as it is planned to be catchy. This is the reason why we include also the first sentence, as the first paragraph should offer a resume of the full article content. As this

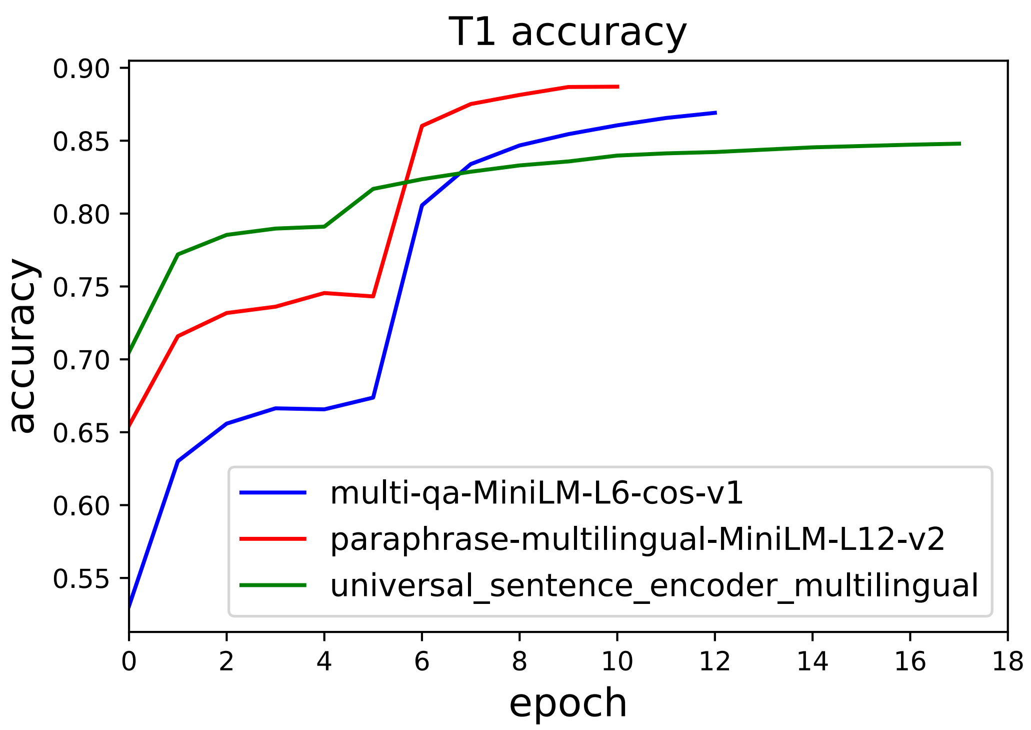


Figure 4.1: Learning curves for the validation accuracy over Tier 1 for experiments on the embedder type

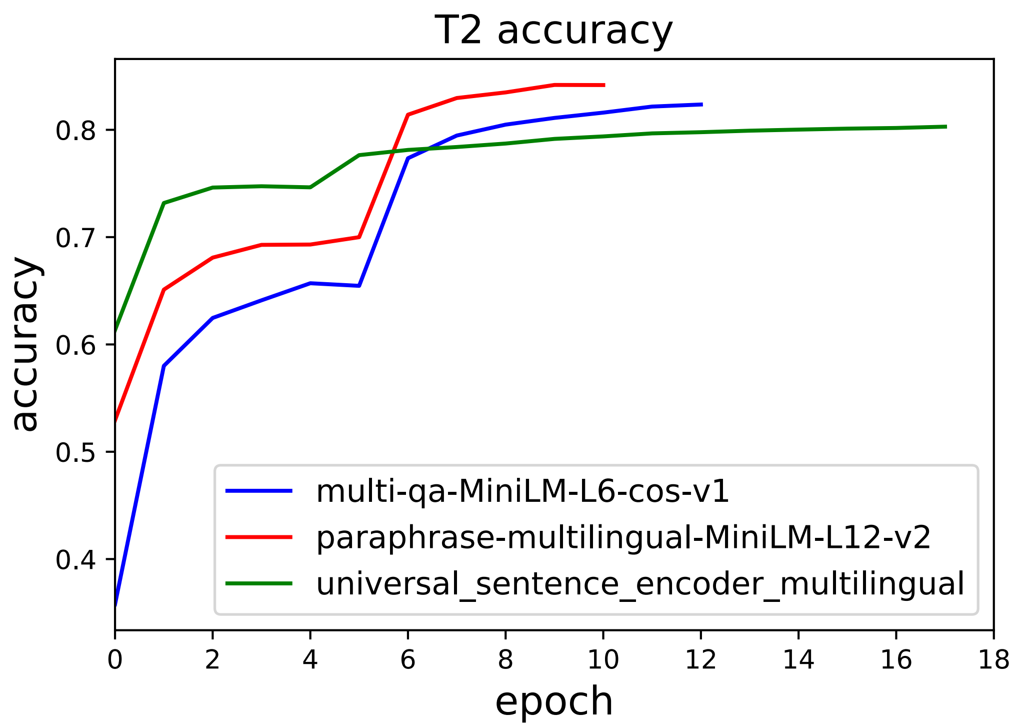


Figure 4.2: Learning curves for the validation accuracy over Tier 2 for experiments on the embedder type

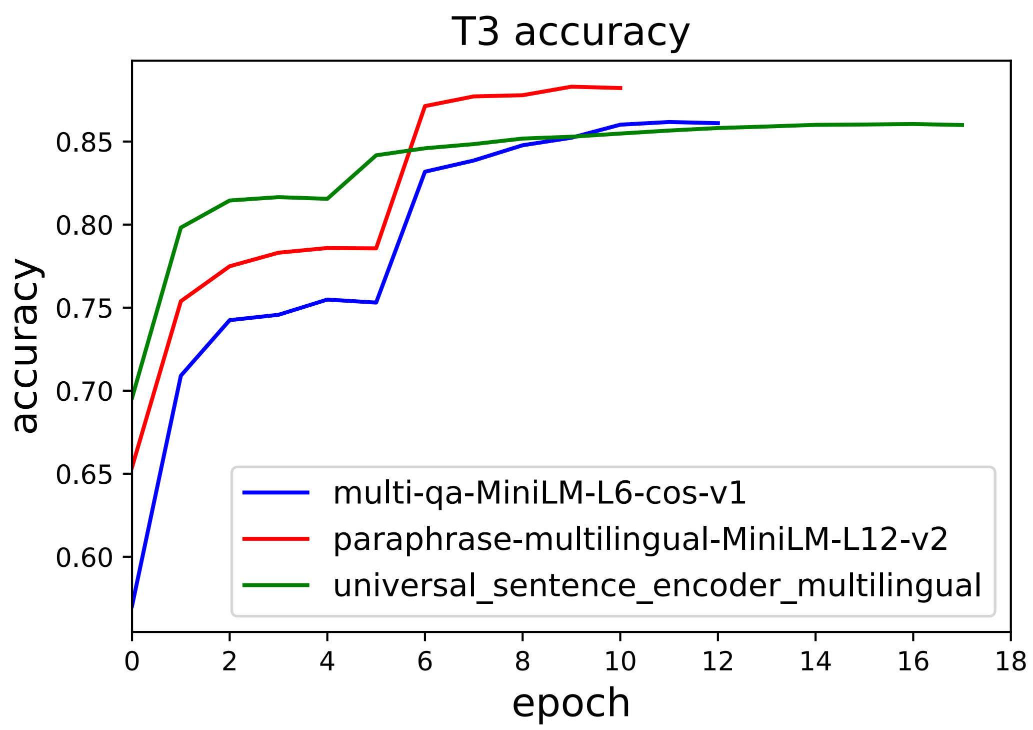


Figure 4.3: Learning curves for the validation accuracy over Tier 3 for experiments on the embedder type

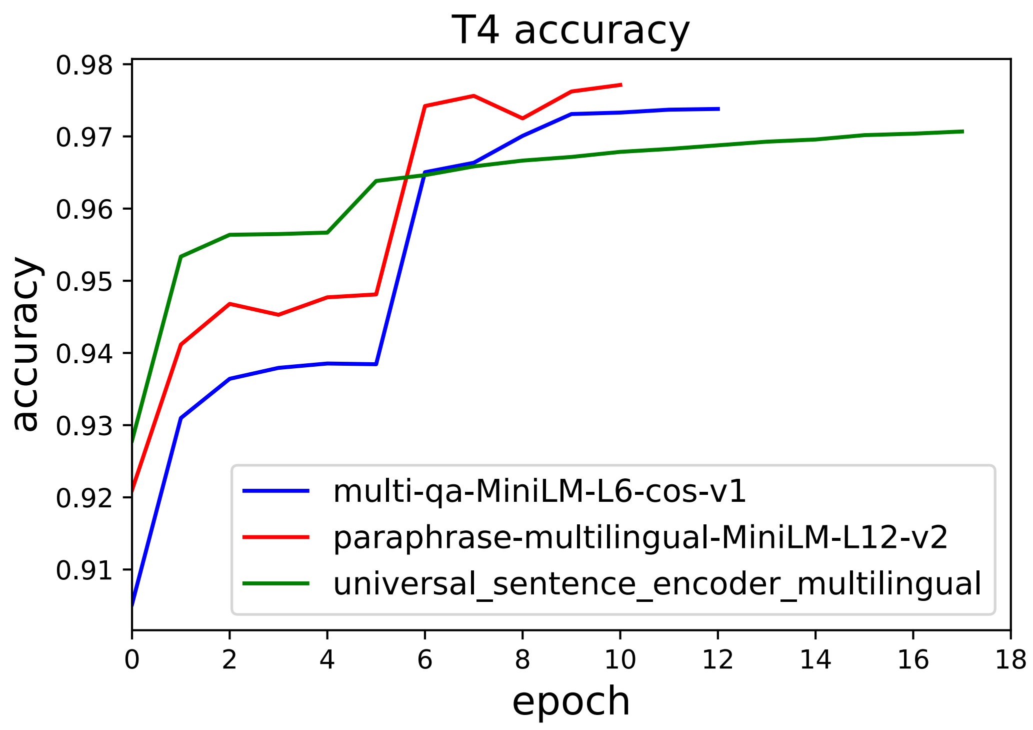


Figure 4.4: Learning curves for the validation accuracy over Tier 4 for experiments on the embedder type

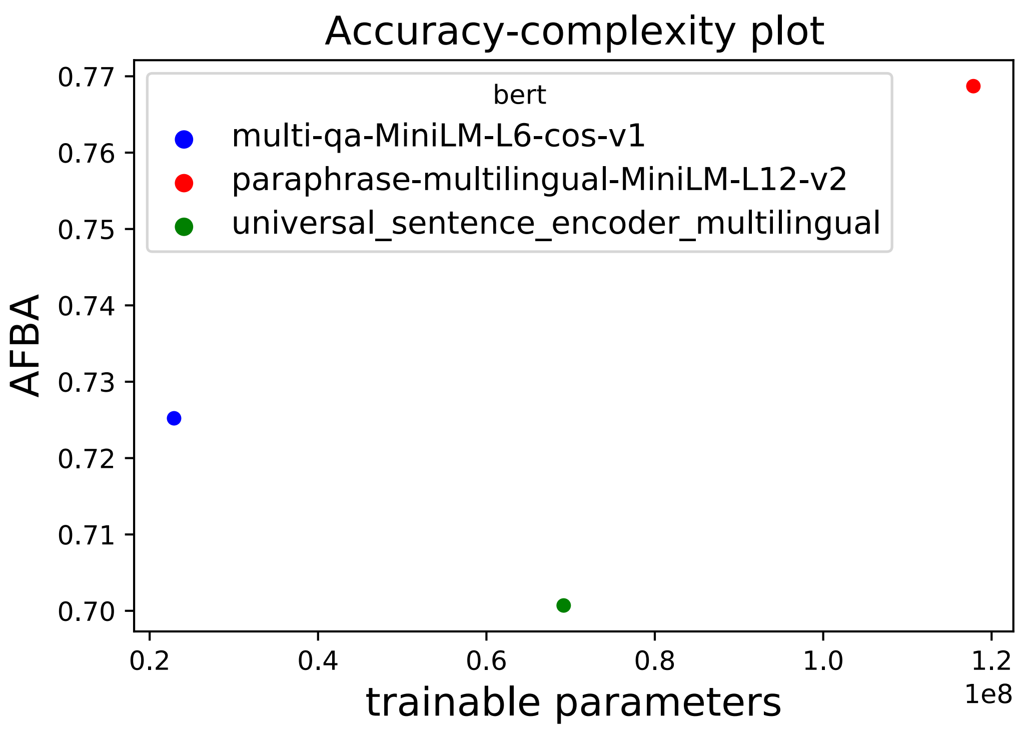


Figure 4.5: Accuracy-complexity plot

might not be true for all articles and our choice would be gratuitous, we perform the same test but adding now a random sentence from the article.

From the learning curves in figures 4.6, 4.7, 4.8, 4.9 we can see how the model fed with the title only and the model fed with the title stacked with the first sentence are pretty close in performance but the absolute best accuracy is always reached by the second one. When adding a random sentence to the title, we face the opposite behaviour: performances get slightly worse. This could be linked to the fact that not all the sentences in an article, when taken by themselves, are easily linked to the main topic and that may deceive the classifier. In fact, when comparing the results obtained with only the first or a random sentence, we see a huge gap in accuracy, with the first sentence-fed classifier dominating.

All this considered, we select the title followed by the first sentence as the input feature for our classifier.

Now, we can build our multilingual classifier, which will constitute the baseline for the Continual Learning techniques implemented in chapter 6.

4.3 Our baseline: Italian and English specific classifier

At this point, we introduce our multilingual baseline model.

Its overall structure can be inspected in figure 4.10. The model takes in input the tokens and attention masks produced by the Bert Tokenizer provided by the HuggingFace library, that will be fed to the embedding layer (tf_bert_model in 4.10). The embedder is followed

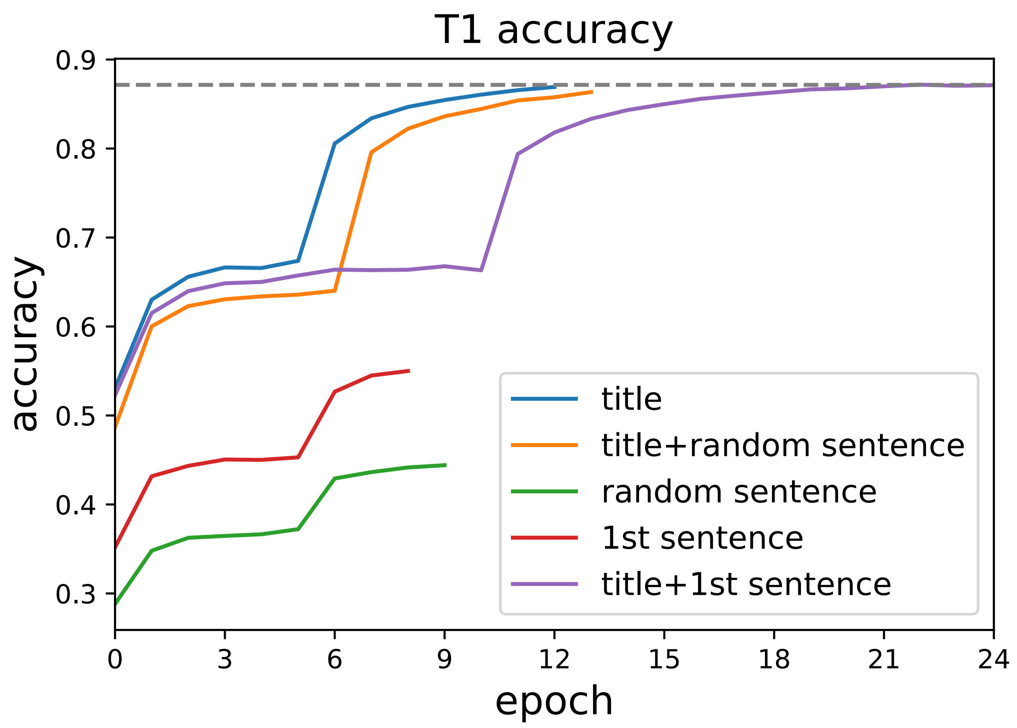


Figure 4.6: Learning curves for the validation accuracy over Tier 1 for experiments on articles slices

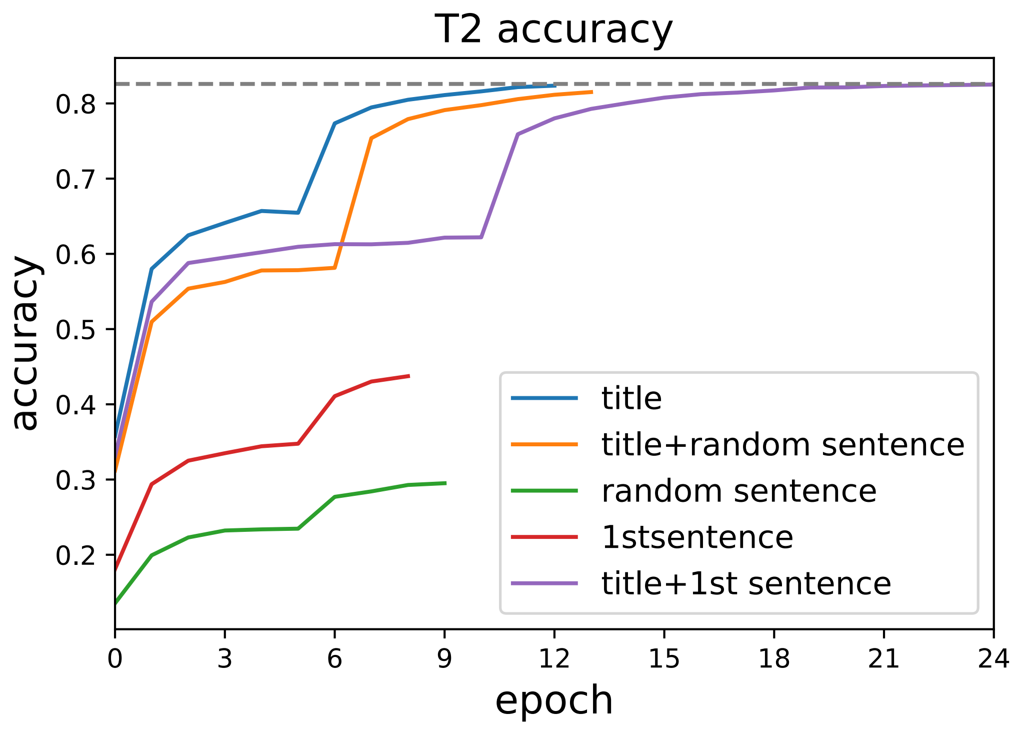


Figure 4.7: Learning curves for the validation accuracy over Tier 2 for experiments on articles slices

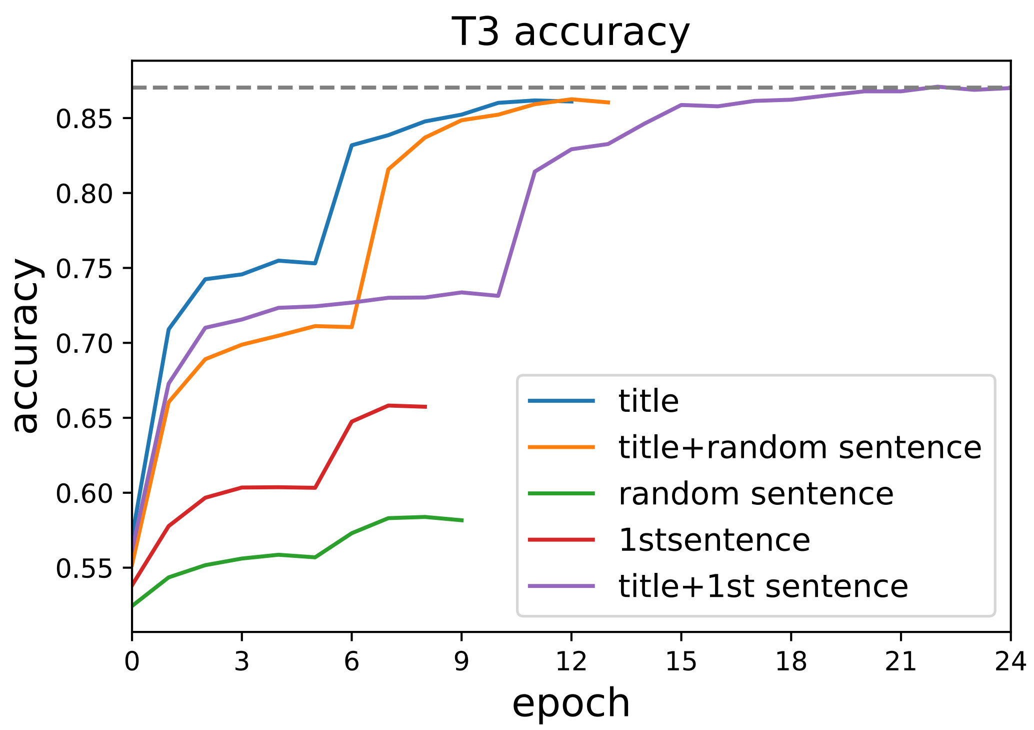


Figure 4.8: Learning curves for the validation accuracy over Tier 3 for experiments on articles slices

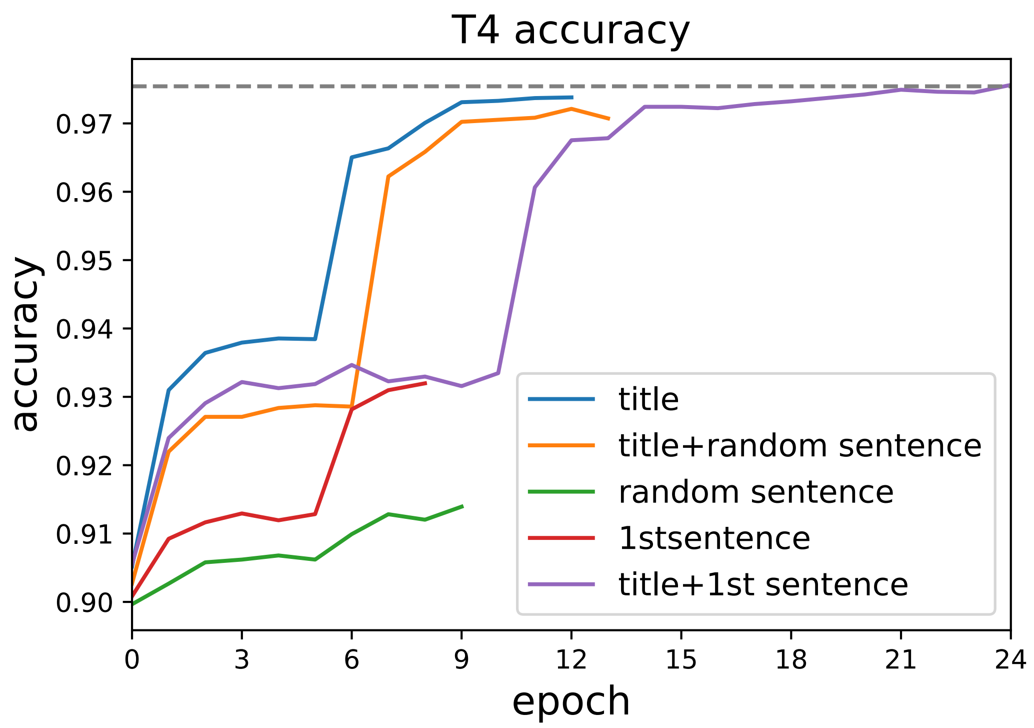


Figure 4.9: Learning curves for the validation accuracy over Tier 4 for experiments on articles slices

by the actual classification layers, in particular, we have 3 hidden stacks, each one made of a Batch Normalization layer followed by a Dense layer with 477 units and Leaky ReLU activation function (see equation 4.1), with $a = 0.3$.

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{if } \leq 0 \end{cases} \tag{4.1}$$

Finally, we have 4 output layers, with respectively 27, 261, 171 and 18 neurons, respectively encoding a category in each layer of the taxonomy. Their activation function is linear, but the most probable predicted category for each tier is retrieved via softmax mapping.

4.3.1 The training phase

The model is trained on Italian and English labelled data samples. The training process involves two phases: at first, we freeze the pre-trained embedding layer and we train the task-specific back-end and afterwards we fine-tune the whole architecture. In both cases, the loss function is the Cross Entropy loss computed between the network predictions and the ground-truth labels.

In the first step, the optimization algorithm is Adam, with an adaptive learning rate depending on the epoch number $lr(epoch) = 10^{-3}/(1 + 0.001 * epoch)$. The maximum number of epochs is set to 25, with Early Stopping option controlling the validation accuracy with the patience parameter set to 5. The batch size is set to 16 and we have $870,339$ trainable parameters. The training process reaches convergence after 22 epochs.

We now proceed with the fine-tuning phase, which includes $23,583,555$ trainable parameters. The optimizer is again Adam, with a much lower initial learning rate $lr(epoch) = 10^{-5}/(1 + 0.05 * epoch)$ Now the maximum number of epochs is set to 30 and convergence in accuracy is reached in 19 epochs. The batch size is set to 16.

The learning curves for the loss and the accuracy over the 4 Tiers are displayed respectively in figures 4.11, 4.12, 4.13, 4.14, 4.15. We can observe a steep change in all of them in correspondence to the $22nd$ epoch: this is due to the increase in the number of trainable parameters, that allows the model to reach a lower minimum in the loss manifold.

4.3.2 Evaluation on the full test dataset

In this section, we report both plain and taxonomic metrics for our model on the full test dataset, containing pieces of text in both Italian and English. To have its first evaluation, we compute them before the probability-based sorting phase. Then, in order to be consistent with the taxonomic structure, we repeat the calculations after the probability-based sorting phase.

4.3.2.1 Metrics before probability-based sorting

At first, as the most straightforward standard approach to benchmark a classifier, we evaluate the accuracy. As mentioned, it is simply computed as the true positive per cent ratio for each tier softmax logits. As it has no particular meaning for a taxonomic classifier, we report it for the sake of completeness, but future evaluations will be based on taxonomic metrics only.

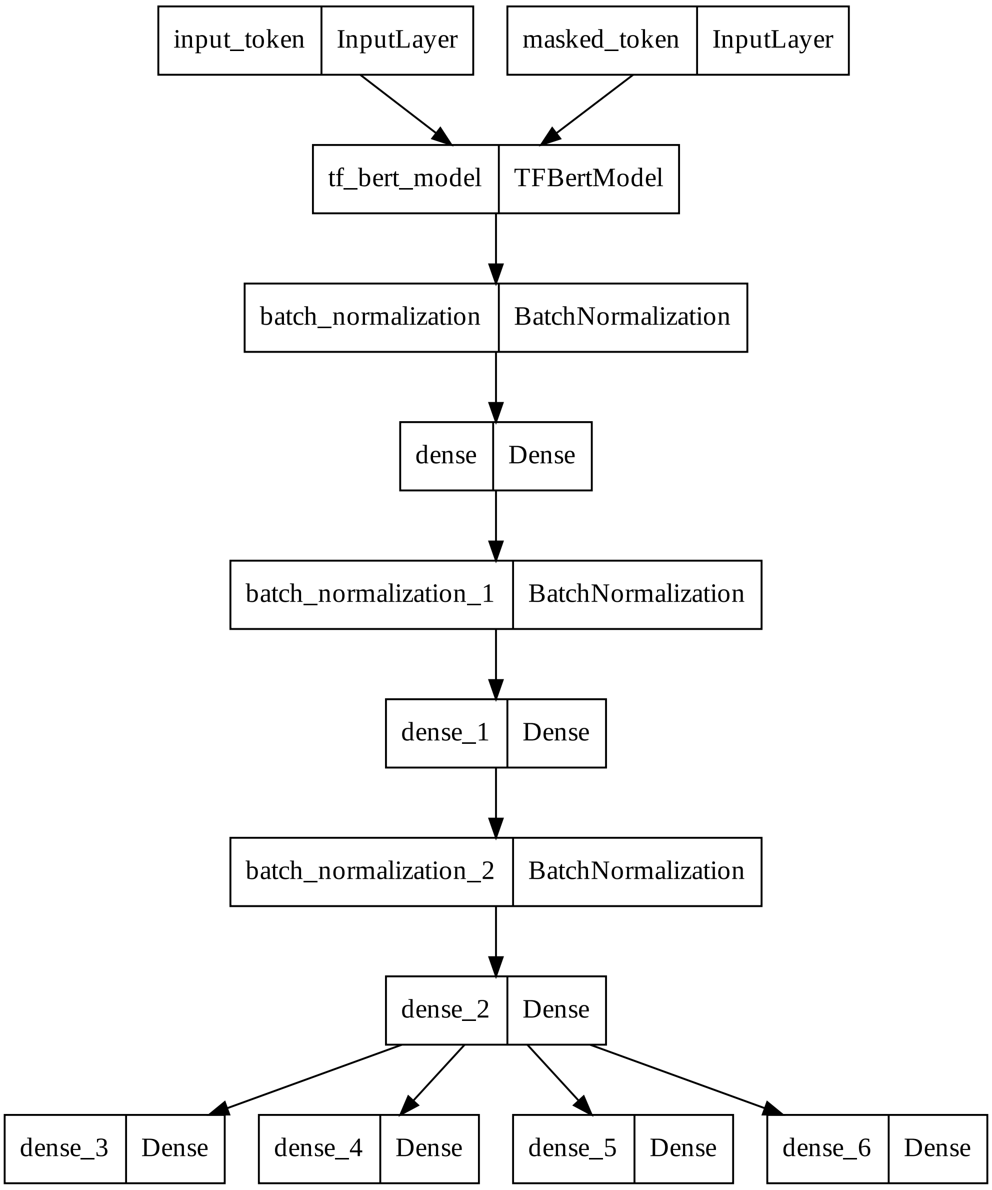

input_token | InputLayer
masked_token | InputLayer
tf_bert_model | TFBertModel
batch_normalization | BatchNormalization
dense | Dense
batch_normalization_1 | BatchNormalization
dense_1 | Dense
batch_normalization_2 | BatchNormalization
dense_2 | Dense
dense_3 | Dense
dense_4 | Dense
dense_5 | Dense
dense_6 | Dense

Figure 4.10: Baseline model structure

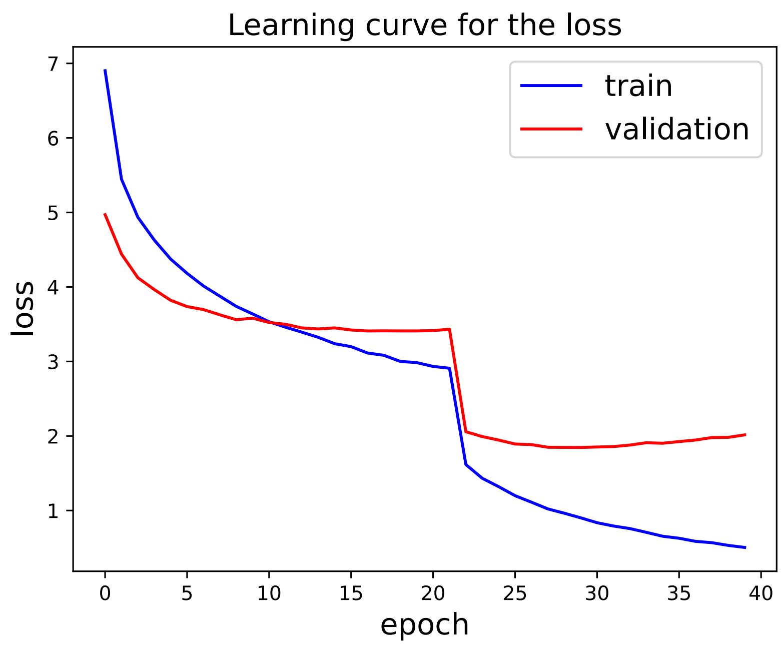


Figure 4.11: Learning curve for train and validation loss

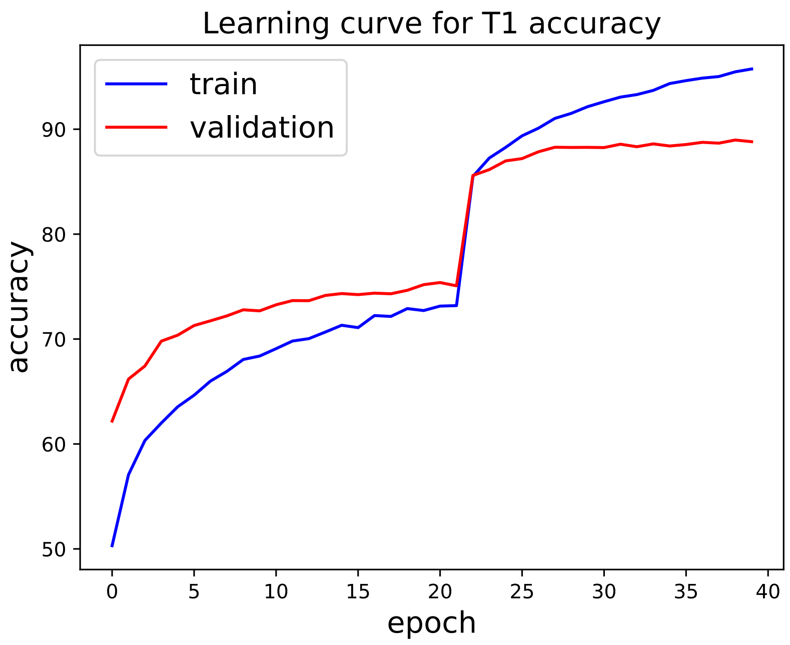


Figure 4.12: Learning curve for train and validation accuracy on Tier 1

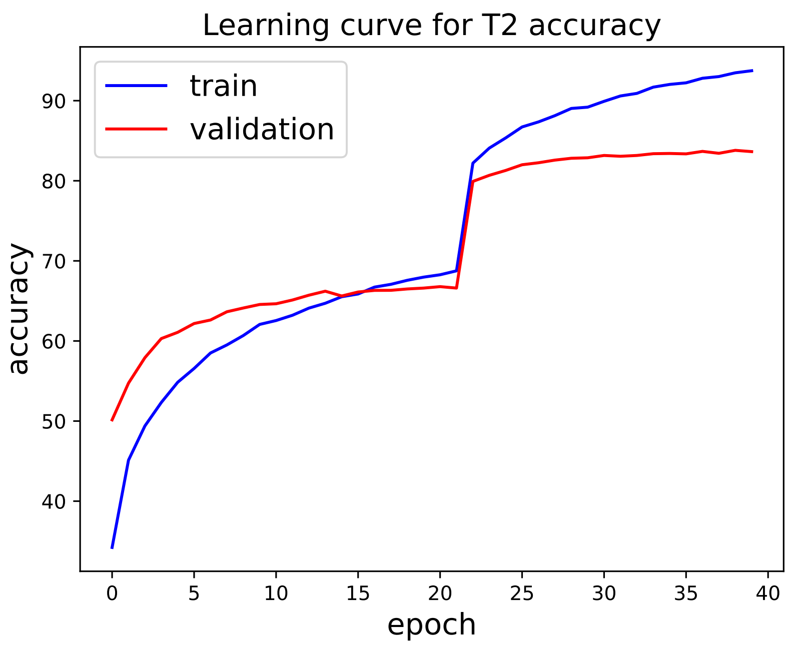


Figure 4.13: Learning curve for train and validation accuracy on Tier 2

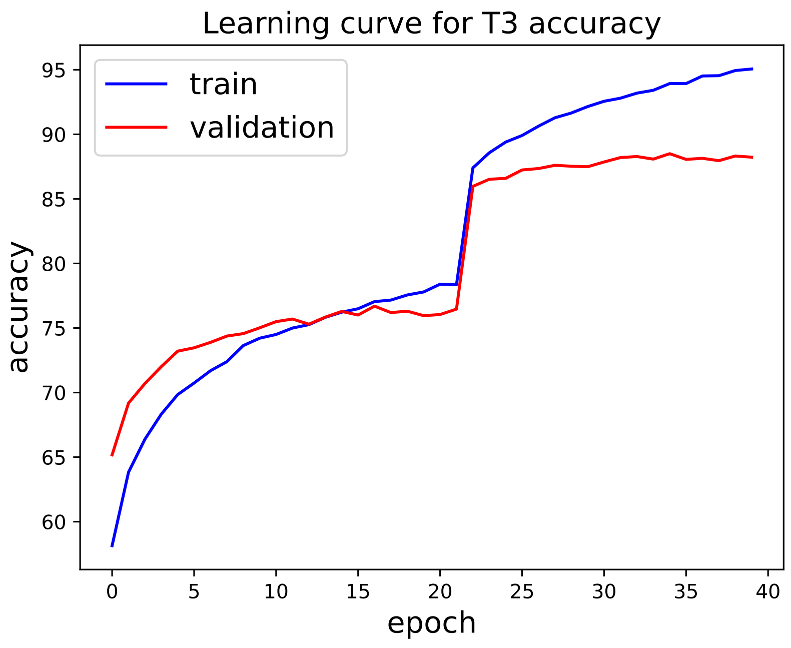


Figure 4.14: Learning curve for train and validation accuracy on Tier 3

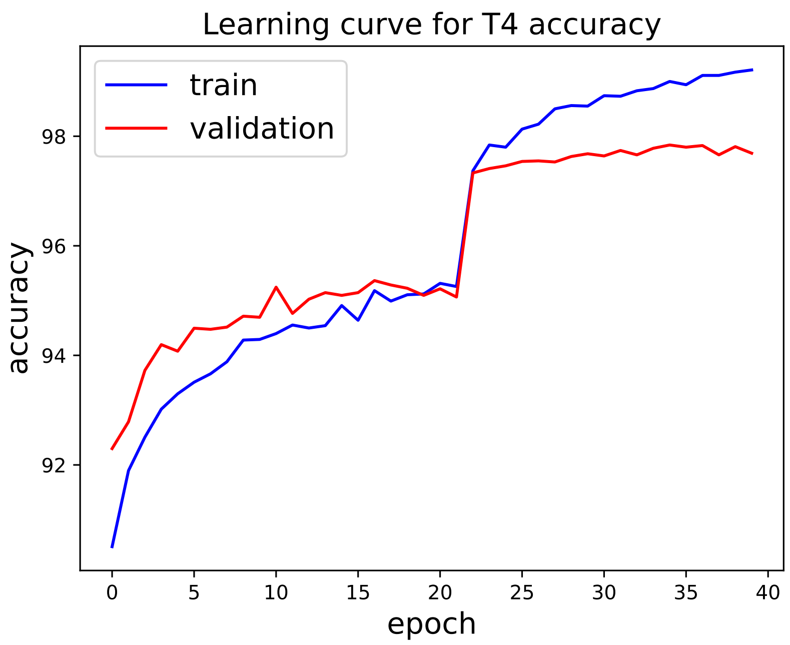


Figure 4.15: Learning curve for train and validation accuracy on Tier 4

The results are reported in table 4.1 It shows that the model has its worst performance on Tier 2 predictions and its best one on Tier 4 predictions: both results can be justified by the distribution of articles among the categories in these two layers. In fact, Tier 2 is the taxonomic level with the highest number of distinct categories, whereas in Tier 4 90% of articles belong to the *Nan* class.

Tier-by-tier accuracy			
T1	T2	T3	T4
89.0	83.5	88.0	97.6

Table 4.1: Tier-by-tier accuracy before probability-based sorting on full test dataset

A deeper perspective can be offered by specific taxonomic metrics, explained in table 4.2. Here, we can infer that most mistakes happen at Tier 2 level by looking at the accuracy gap in AFTA between subsequent levels: we have the biggest performance drop when going from T1 to T2, whereas it is basically negligible when proceeding from T3 to T4.

AFTA			
T1	T2	T3	T4
89.0	81.1	77.2	77.0

AFDR	AFBA
81.1	77.0

Table 4.2: Taxonomic metrics before probability-based sorting on full test dataset

4.3.2.2 Metrics after probability-based sorting

We now apply the probability-based sorting algorithm to retrieve the most overall likely prediction and to impose consistency between Tiers.

The new taxonomic metrics after the post-processing phase are reported in table 4.3. What emerges is that after the taxonomy alignment of results all the metrics have improved.

The joint-probability-based ordering algorithm allows to easily get the top-K taxonomic metrics, that we report for $K = 3$ in table 4.4: after post-processing, we get the full-depth correct result among the most probable 3 predictions almost 90% of the time.

AFTA			
T1	T2	T3	T4
89.3	83.2	79.6	79.3

AFDR	AFBA
82.9	79.3

Table 4.3: Taxonomic metrics after probability-based sorting on full test dataset

Top-3 AFTA			
T1	T2	T3	T4
93.8	91.3	89.1	88.7

Top-3 AFDR	Top-3 AFBA
90.7	88.7

Table 4.4: Top-3 taxonomic metrics after probability-based sorting on full test dataset

4.3.2.3 Confusion matrices

To inspect how the classifier performs over a specific class, we take advantage of the confusion matrix visualization tool.

For categories in Tier 1, its application is straightforward (see figure 4.16), having the rows representing the actual labels and the columns representing the predicted labels for each class. When considering the hierarchical structure, i.e. tiers that are subsequent to the first, the representation becomes more tricky due to the high number of categories present in Tier 2 (figure 4.17) and in Tier 3 (figure 4.18. What we represent for these two levels is, then, a clustered confusion matrix, in which we collapse sub-hierarchies into their root node as proposed in [9]. For example, for Tier 2, we mark as True Positives all the samples for which the T2 predictions correspond to a category that belongs to the same parent node of their true label in T1.

Overall, we observe a good rate of True Positives for all the layers of the taxonomy. There is a particularly high rate of False Negatives for the T1 *Education* cluster that are predicted as *Academic Interests*. This is also related to an ill-definition in the taxonomy: the Tier 2 category *Language Learning* is present as a children node for both these two categories. The symmetric error is not observed as, in the training set, the articles labelled as *Academic Interests* are 8 times more present than the ones belonging to the *Education* class.

Another inter-cluster error that has an occurrence rate of more than 10% regard *Personal Finance* articles that are wrongly classified as *Business and Finance*. The mistake is

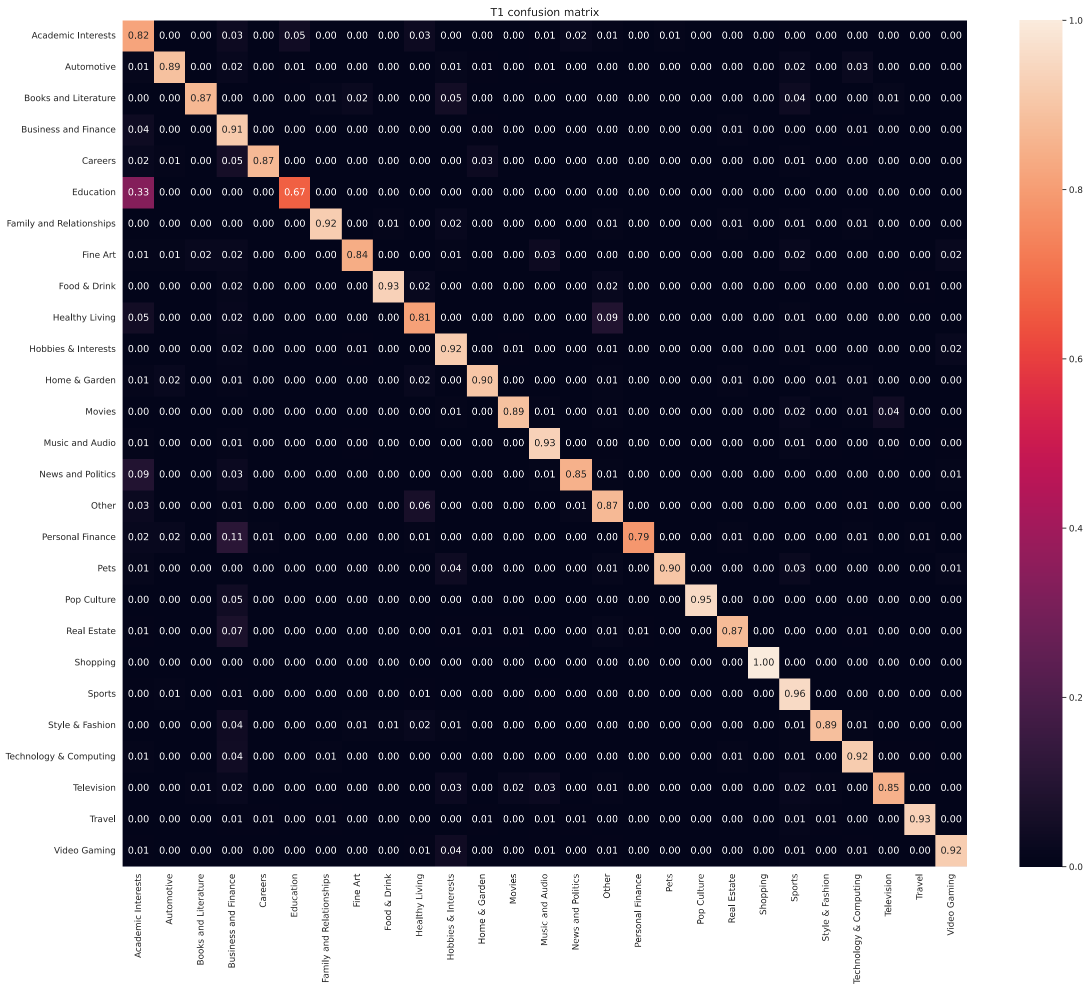


Figure 4.16: Accuracy matrix for Tier 1 predictions after probability-based sorting

made reasonable by the fact that the two topics are pretty similar and share many of their keywords. Again, the symmetric error is not observed for a simple statistical reason: *Business and Finance* articles are 10 times more present than *Personal Finance* one, so the network is keener on classifying into the first cluster.

As far as the confusion matrix for the fourth tier is concerned, there is no need to perform any hierarchical clustering since it includes only 18 possible categories (see figure 4.19). Its classes belong either to the Financial cluster or to the *Healthy Living* topic and what we can notice is that there are no inter-clusters mistakes but only among topics related to the same root node. We can observe a high number of False Positives mapped to the *Nan* class, which is again imputable to class imbalance in the training set: over 90% of its articles have *Nan* T4 ground-truth, pushing the network to classify mostly as that, at the cost of producing False Positives.

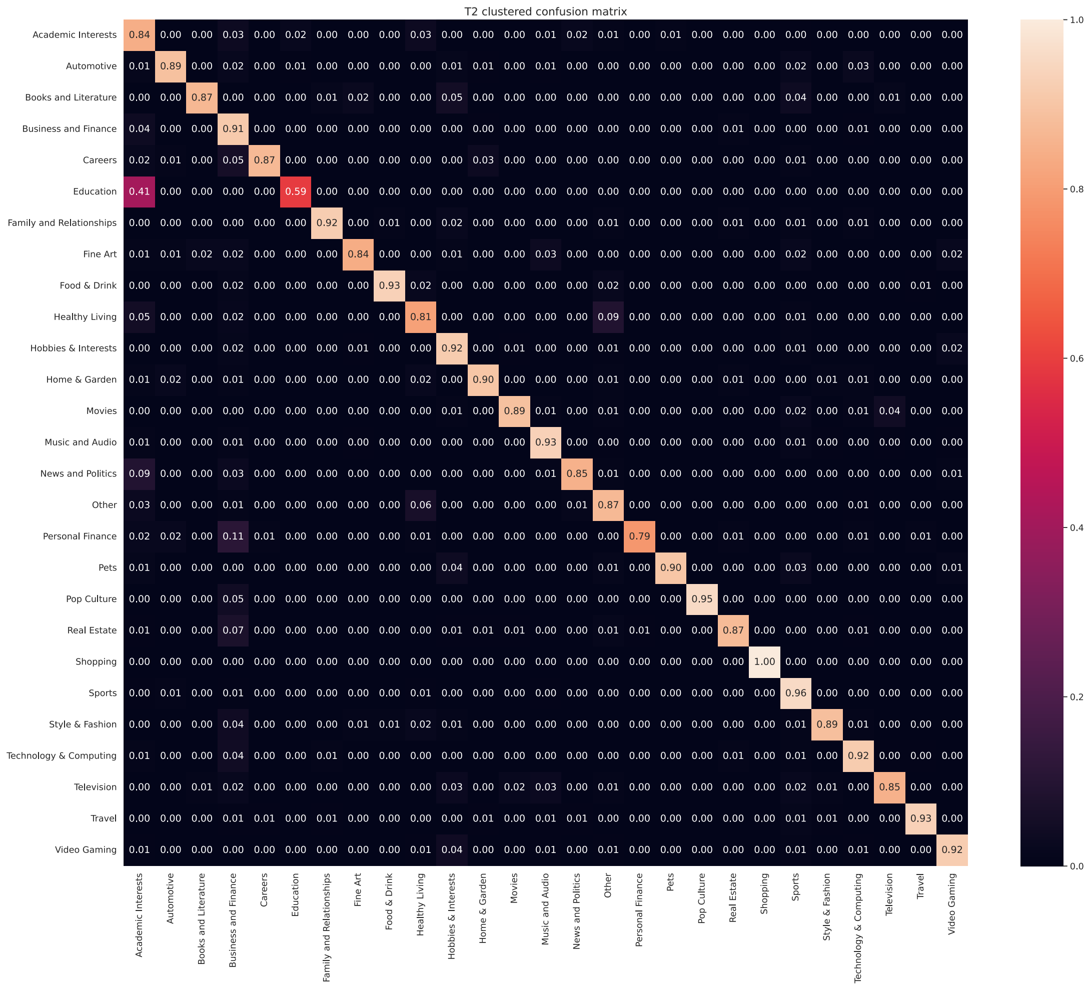
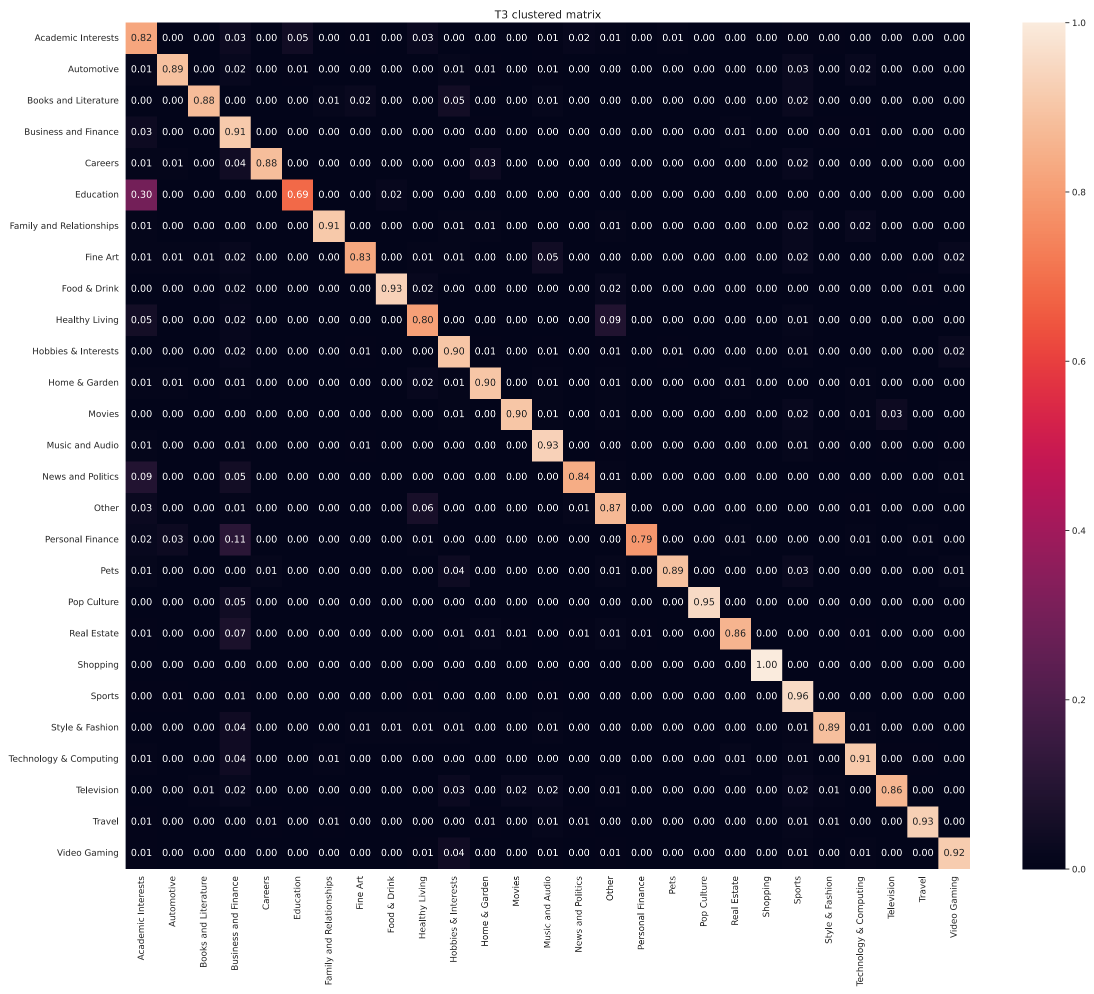


Figure 4.17: Clustered accuracy matrix for Tier 2 predictions after probability-based sorting



	Academic Interests	Automotive	Books and Literature	Business and Finance	Careers	Education	Family and Relationships	Fine Art	Food & Drink	Healthy Living	Hobbies & Interests	Home & Garden	Movies	Music and Audio	News and Politics	Other	Personal Finance	Pets	Pop Culture	Real Estate	Shopping	Sports	Style & Fashion	Technology & Computing	Television	Travel	Video Gaming
Academic Interests	0.82	0.00	0.00	0.03	0.00	0.05	0.00	0.01	0.00	0.03	0.00	0.00	0.00	0.01	0.02	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Automotive	0.01	0.89	0.00	0.02	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.02	0.00	0.00	0.00
Books and Literature	0.00	0.00	0.88	0.00	0.00	0.00	0.01	0.02	0.00	0.00	0.05	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00
Business and Finance	0.03	0.00	0.00	0.91	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00
Careers	0.01	0.01	0.00	0.04	0.88	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00
Education	0.30	0.00	0.00	0.00	0.00	0.69	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Family and Relationships	0.01	0.00	0.00	0.00	0.00	0.00	0.91	0.00	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.02	0.00	0.00	0.00
Fine Art	0.01	0.01	0.01	0.02	0.00	0.00	0.00	0.83	0.00	0.01	0.01	0.00	0.00	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.02
Food & Drink	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.93	0.02	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00
Healthy Living	0.05	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.80	0.00	0.00	0.00	0.00	0.00	0.09	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00
Hobbies & Interests	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.01	0.00	0.00	0.90	0.01	0.00	0.01	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.02
Home & Garden	0.01	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.02	0.01	0.90	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00
Movies	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.90	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.01	0.03	0.00	0.00
Music and Audio	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.93	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00
News and Politics	0.09	0.00	0.00	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.84	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01
Other	0.03	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.00	0.01	0.87	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00
Personal Finance	0.02	0.03	0.00	0.11	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.79	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.01	0.00
Pets	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.01	0.00	0.89	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.01
Pop Culture	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.95	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Real Estate	0.01	0.00	0.00	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.00	0.01	0.01	0.01	0.00	0.00	0.86	0.00	0.00	0.00	0.01	0.00	0.00	0.00
Shopping	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
Sports	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.96	0.00	0.00	0.00	0.00	0.00
Style & Fashion	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.01	0.01	0.01	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.89	0.01	0.00	0.00	0.00
Technology & Computing	0.01	0.00	0.00	0.04	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.01	0.00	0.91	0.00	0.00	0.00
Television	0.00	0.00	0.01	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.02	0.02	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.02	0.01	0.00	0.86	0.00	0.00
Travel	0.01	0.00	0.00	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.00	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.00	0.93	0.00
Video Gaming	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.04	0.00	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.01	0.00	0.00	0.92

Figure 4.18: Clustered accuracy matrix for Tier 3 predictions after probability-based sorting

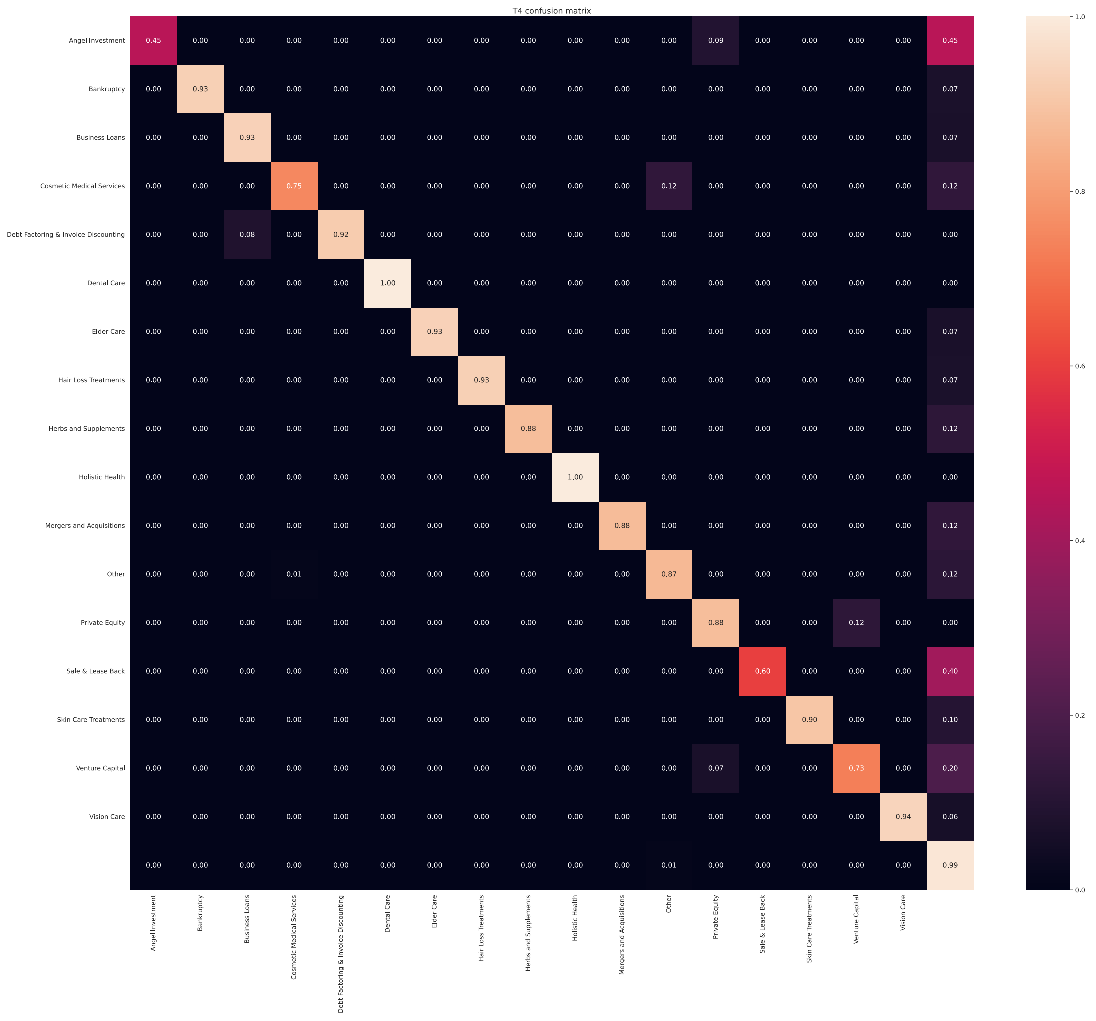


Figure 4.19: Accuracy matrix for Tier 4 predictions after probability-based sorting

After having a full inspection of the model performance on the full test dataset, we study its behaviour on Italian and English separately.

4.3.3 Evaluation on the Italian only test dataset

We start our analysis on the Italian-only test dataset, by computing the raw taxonomic metrics on it. What we notice from 4.5 is that on one hand, the overall trend is the same, observing the biggest *AFTA* drop in correspondence of T2, whereas, on the other hand, the actual performance is lower than on the complete dataset. We will dive deeper into this phenomenon in 4.3.3.1.

Again, after the probability-based post-processing, all the computed metrics increase, as shown in 4.6.

4.3.3.1 Metrics before probability-based sorting

AFTA			
T1	T2	T3	T4
86.6	77.8	73.6	73.3

AFDR	AFBA
77.8	73.3

Table 4.5: Taxonomic metrics before probability-based sorting on Italian test dataset

4.3.3.2 Metrics after probability-based sorting

AFTA			
T1	T2	T3	T4
87.1	80.3	76.5	76.3

AFDR	AFBA
80.0	76.3

Table 4.6: Taxonomic metrics after probability-based sorting on Italian test dataset

4.3.4 Evaluation on the English-only test dataset

4.3.4.1 Metrics before probability-based sorting

The focus is now on metrics on the English-only sub-dataset. As it could be expected from its behaviour on Italian data, we see an overall better performance than the average in the two languages. This difference observed between Italian and English articles is probably imputable to the embedder pre-training.

AFTA			
T1	T2	T3	T4
91.9	85.3	82.0	81.8

AFDR	AFBA
85.2	81.8

Table 4.7: Taxonomic metrics before probability-based sorting on English test dataset

4.3.4.2 Metrics after probability-based sorting

The sorting phase turns out to bring, also in this case, both the advantage of producing reasonable taxonomic predictions and enhancing accuracy: as a reference, for the $AFBA$ metric it is observed a rise of 1.5. We can conclude this evaluation phase by asserting that

AFTA			
T1	T2	T3	T4
92.2	86.9	83.6	83.3

AFDR	AFBA
86.5	83.3

Table 4.8: Taxonomic metrics after probability-based sorting on English test dataset

we have observed a pretty accurate model, that also when making wrong predictions, still keeps them reasonable in most cases.

In the next chapter, we deal with the common problem of accuracy-confidence inconsistency in modern deep learning models and how to deal with that.

Chapter 5

Network calibration

Modern Deep Neural Networks (DNNs) can reach astonishing accuracy over several tasks, sometimes out-performing human capability[11]. However, in real-life scenarios, like autonomous driving or medical health applications, accuracy estimation should be coupled with an unbiased confidence measure, revealing at which level of trust we should rely on the network prediction. Usually, the available confidence measure is the output of a softmax layer, which is interpreted as the probability of the output label being correct. Shallow Neural Networks are usually well-calibrated, meaning that the softmax probability associated with the network-produced label reflects the ground-truth correctness likelihood. Instead, this is not true for most DNNs. It is observed that an increase in network complexity is highly linked with getting over-confident[10]. In the case of a calibrated network, the prediction-linked confidence value is close to the expected accuracy. When dealing with an over-confident network, the likelihood indicator overcomes, on average, the network accuracy. This accuracy-confidence imbalance can often be linked to a Negative Likelihood Loss over-fitting trend, which allows to decrease the 0/1 loss, which corresponds to a lower classification error, at the price of increasing NLL loss, which encodes the probabilistic error. In this framework, network calibration methods arise as a technique that aims at reducing the miscalibration error by decreasing the confidence of misclassified samples while leaving barely untouched the confidence of correct predictions[28]. The two main categories of calibration methods are probabilistic approaches and measure-based approaches. The second ones are usually the go-to choice since they include various post-processing algorithms which prevent re-training the network. Among them, the Temperature Scaling (TS) algorithm can be found, representing the state-of-the-art measure-based approach. It offers both an extremely low computational complexity, having a single optimization parameter T, called temperature, together with appreciable effectiveness, out-performing several other methods on a set of tasks[10].

A quantitative set-up, including definitions, calibration measures and algorithm, is built in the following sections.

5.1 Definitions

In the framework of supervised multi-class classification, the training set is constituted of couples (X, Y), with $X \in \mathcal{X}$, being the input and $Y \in \mathcal{Y} = \{1, \ldots, K\}$ being the label.

Both of them are random variables following a joint distribution $\pi(X,Y) = \pi(Y|X)\,\pi(X)$. Given a classifier D, in our case a neural network, the returned output, when fed with X, is $D(X) = (\hat{Y}, \hat{P})$, being $\hat{Y}$ the network-predicted label and $\hat{P}$ its associated probability. As mentioned before, most often $\hat{P} = S_{Y=\hat{Y}}(X)$ is the softmax probability computed from the logit layer $[h_1, \dots, h_K]$ as $S_Y = \exp(h_Y) / \sum_{i=1}^{K} \exp(h_j)$. The model is calibrated if the confidence $\hat{P}$ can be interpreted as a probability in a frequentist approach, meaning that it gives the likelihood that the sample has been correctly classified. This last definition resembles the one of accuracy, with it being the ratio between correctly classified samples and observed ones. Then, what we are asking for is, in the case of a set of samples having $\hat{P} = 0.95$, its accuracy should be 95%.
Formally, calibration is defined in [10] as

$$\mathbb{P}(\hat{Y} = Y|\hat{P} = p) = p, \quad \forall p \in [0,1]$$

where p is the probability over $\pi(X,Y)$. Clearly, perfect calibration can not be achieved in real-life settings, since $\hat{P}$ is a continuous random variable, that can not be computed by a finite number of samples. Then, empirical measures of calibration that allow to inspecting of practical settings are defined in the following section.

5.2 Measures

The two most common tools used to inspect network calibration are NLL and Expected Calibration Error (ECE) measures, along with reliability diagrams for visualization.

5.2.1 Negative log likelihood

Negative log-likelihood, also referred to as cross entropy loss in Deep Learning, is a standard measure of similarity between probability distributions. In particular, NLL in supervised DL is minimized when the classifier-produced distribution $\hat{\pi}(Y|X) = S_Y(X)$ is exactly the ground truth distribution $\pi(Y|X)$, as it should be when the network is perfectly calibrated. Then, having n samples $(x_i, y_i) \sim \pi(Y|X)$, NLL definition is:

$$NLL = -\sum_{i=1}^{n} \log\left(S_{y_i}(x_i)\right)$$

From this definition, it is possible to explain why NLL overfitting still benefits accuracy. In fact, in several deep architectures, it can be observed that the network starts to overfit NLL loss while it keeps decreasing 0/1 loss, i.e. to increase accuracy. The same behaviour is shown in our architecture, as displayed in 5.1. That means that the network, that is able to decrease training NLL by increasing output probability, is learning better classification sacrificing a well-modelled probability distribution[10].

5.2.2 Expected Calibration Error (ECE)

As mentioned before, miscalibration is the inconsistency between confidence and accuracy. Then, it can be simply expressed by the expectation of the absolute difference between these two quantities:

$$\mathop{\mathbb{E}}_{\hat{P}}\left[\left|\mathbb{P}\left(\hat{Y} = Y|\hat{P} = p\right) - p\right|\right] \tag{5.1}$$

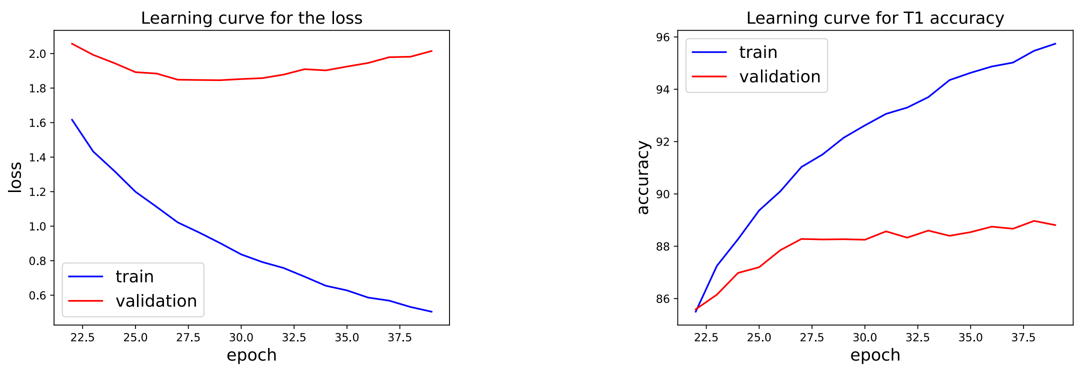


Figure 5.1: Learning curves for the fine-tuning phase for loss and accuracy on Tier 1

To obtain the corresponding empirical quantity, an approximate distribution of miscalibration is needed. This is obtained by binning samples, depending on their confidence, into M equally-spaced bins. Then, the m-th bin contains the set B_m of samples having confidence in the range $R_m = \left(\frac{m-1}{M}, \frac{m}{M}\right]$. Its average confidence is

$$conf(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \hat{p}_i,$$

where $\hat{p}_i$ denotes the confidence assigned to the i_{th} instance in B_m. The accuracy for the set B_m is simply defined as the percentage of its correctly classified elements

$$acc(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \mathbb{1}\left(\hat{y}_i = y_i\right),$$

having set $\hat{y}_i$ as the predicted class label for sample i. Then, considering 5.1, the corresponding empirical quantity is the Empirical Classification Error, as defined in [29]:

$$ECE = \sum_{m_1}^{M} \frac{|B_m|}{n} \left| acc\left(B_m\right) - conf\left(B_m\right) \right|$$

This summary statistic is strictly linked to reliability diagrams as a visualisation tool, that allows the inspection of bin-by-bin calibration measure.

5.2.3 Reliability diagrams

Reliability diagrams allow us to visually inspect the calibration status of a model by plotting the average bin accuracy as a function of confidence in a bar plot. The binning process is the same as the one illustrated in the previous section. Therefore, since for a perfectly calibrated model we would expect $acc(B_m) = conf(B_m)$, i.e. the graph should display the identity function, the miscalibration magnitude can be inferred by looking at the calibration gap, given by the difference between average accuracy and confidence of a certain bin.

5.3 Temperature scaling

The state-of-the-art approach among measure-based calibration methods is temperature scaling, a post-processing parametric algorithm relying on a single scalar parameter $T > 0$, called temperature. It acts by scaling the logit layer of the network, with the result of softening the output of the softmax layer:

$$S_{y=y_i}(x_i, T) = \exp\left(\frac{h_i^{y_i}(x)}{T}\right) / \sum_{j=1}^{K} \exp\left(\frac{h_i^j(x)}{T}\right)$$

This operation does not affect the argmax of the softmax distribution, leaving each predicted label, and then the accuracy, unchanged. The optimal value T^* is computed via minimization of NLL loss on a hold-out validation set, that in practice will be the same used for hyper-parameters selection.

$$T^* = \underset{T}{argmin}\left(-\sum_{i=1}^{n} \log\left(S_{y=y_i}(x_i, T)\right)\right)$$

Looking for zeros of the derivative w.r.t T of this scaled NLL loss function results in:

$$\sum_{i=1}^{n} h_i^{y_i} = \sum_{i=1}^{n}\sum_{k=1}^{K} h_i^k S_{y=k}(x_i, T^*)$$

from which it arises that the optimal T^* value is obtained by the trade-off between the low temperature required by correctly classified samples, for which $k = y_i$, and the trend of $T \to \infty$ imposed by mistaken samples ($k \neq y_i$). This means that confidence will be increased for correct predictions and lowered for the wrong ones so that accuracy-confidence consistency is boosted.

5.3.1 Single temperature parameter

We now apply the temperature scaling algorithm to our multi-tier classifier. As a first lowest-complexity approach, we admit a single temperature parameter T, which will be the same among the 4 outputs.

The calibration metrics we are considering take the following values before calibration: ECE= 0.23, NLL= 4.30

The optimal parameter T^* is found minimizing the total NLL loss on the validation set via the L-BFGS optimization algorithm with a learning rate magnitude of 10^{-4} over 10^5 iterations.

The optimal value for T^* is 1.7 and the calibration metrics report $ECE = 0.089$, NLL= 2.24 after scaling. We report in table 5.1 the total and tier-by-tier NLL and ECE loss before and after temperature scaling.

Since the temperature scaling algorithm does not affect the likelihood ordering of output nodes for a single tier and since the scaling parameter is shared among tiers, all the metrics are left untouched by this calibration procedure, for whose we remand at table 4.3

ECE	T1	T2	T3	T4	TOT
before calibration	0.05	0.11	0.05	0.01	0.23
after calibration	0.02	0.03	0.03	0.01	0.09

NLL	T1	T2	T3	T4	TOT
before calibration	0.47	1.87	0.54	1.42	4.31
after calibration	0.40	1.28	0.47	0.08	2.24

Table 5.1: ECE and NLL metrics before and after temperature scaling algorithm with single temperature parameter

ECE	T1	T2	T3	T4	TOT
before calibration	0.05	0.11	0.05	0.01	0.23
after calibration	0.01	0.01	0.01	0.003	0.04

NLL	T1	T2	T3	T4	TOT
before calibration	0.47	1.87	0.54	1.42	4.31
after calibration	0.40	1.23	0.46	0.08	2.17

Table 5.2: ECE and NLL metrics before and after temperature scaling algorithm with multiple temperature parameter

5.3.2 Tier-specific temperature parameter

We now increase the complexity of the algorithm by allowing each tier to have its own independent temperature parameter. The notation used implies that T_{t_i} is the temperature parameter for the i_{th} tier and $T^*_{t_i}$ is its optimal value.

The optimal parameters $T^*_{t_1}$, $T^*_{t_2}$, $T^*_{t_3}$, $T^*_{t_4}$ are found minimizing the total NLL loss on the validation set via the L-BFGS optimization algorithm with learning rate magnitude of 10^{-4} over 10^5 iterations.

The optimal values are $T^*_{t_1} = 1.53$, $T^*_{t_2} = 1.90$, $T^*_{t_3} = 1.52$, $T^*_{t_4} = 1.55$ and the calibration metrics report $ECE = 0.038$, NLL= 2.24 after scaling. We report in table 5.2 the total and tier-by-tier NLL and ECE loss before and after temperature scaling.

We resume here the total NLL and ECE metrics before and after scaling, with both the shared and the tier-specific temperature parameters, in order to compare the two scaling methods.

	ECE	NLL
before calibration	0.23	4.31
single T calibration	0.09	2.24
multiple T calibration	0.04	2.17

Table 5.3: Shared vs tier-specific temperature parameter

It is straightforward to notice that the 4 degrees of freedom algorithm performs better than its single parameter counterpart.

From now on, then, the tier-specific temperature calibrated model will be our reference classifier for Italian and English text and it will be used as a starting point for continual

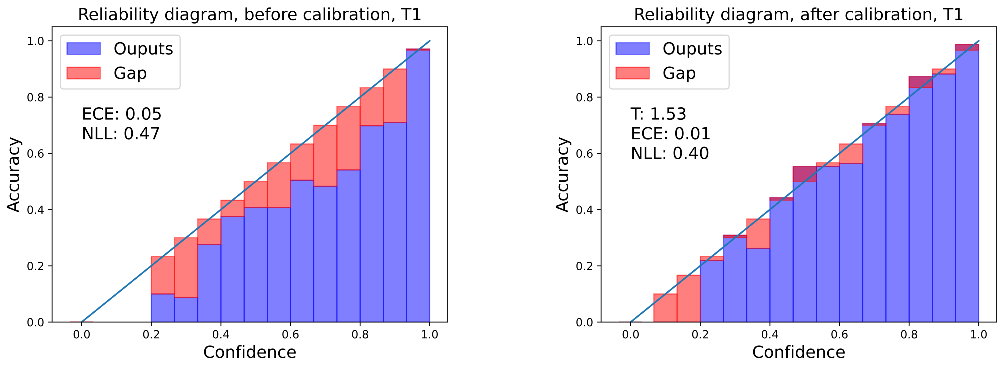


Figure 5.2: Reliability diagram for Tier 1, before (left) and after (right) temperature scaling calibration

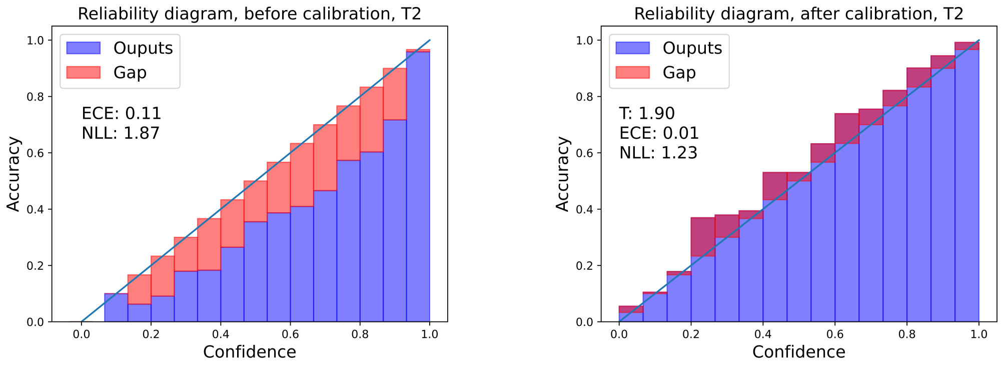


Figure 5.3: Reliability diagram for Tier 2, before (left) and after (right) temperature scaling calibration

learning techniques for incrementing the number of languages.

Finally, in Figures 5.2, 5.3, 5.4, 5.5 we report the reliability diagrams for, respectively, T1, T2, T3 and T4. For each tier, the diagrams before any calibration step and after the tier-specific temperature scaling procedure are shown: the blue bars report the average accuracy for prediction in a certain confidence interval, whereas the red bars indicate the accuracy-confidence gap within a specific bin softmax outputs. We observe an overall significant gaps reduction after calibration, as a reflection of the ECE loss reduction via temperature scaling.

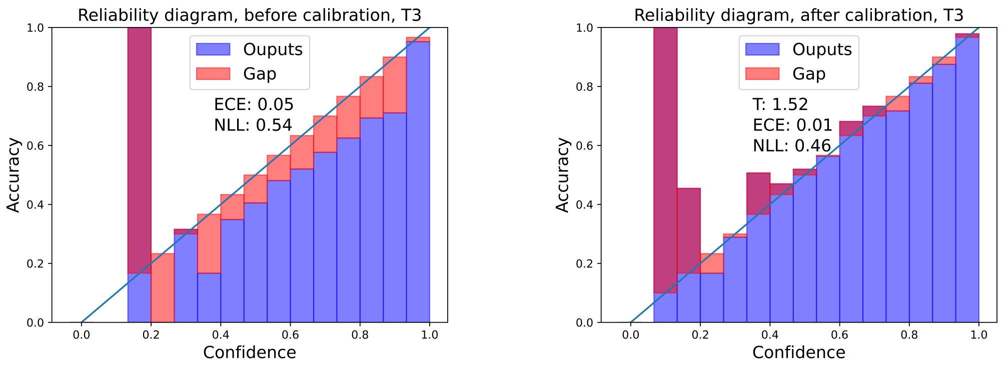


Figure 5.4: Reliability diagram for Tier 3, before (left) and after (right) temperature scaling calibration

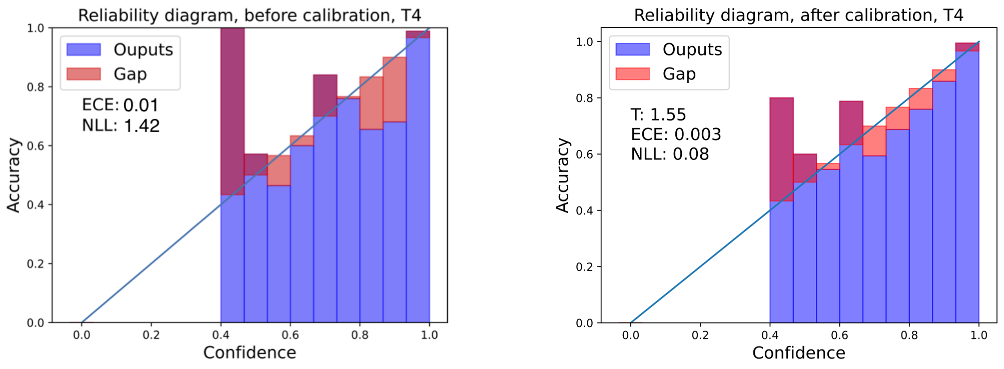


Figure 5.5: Reliability diagram for Tier 4, before (left) and after (right) temperature scaling calibration

Chapter 6

Extending multilingual models to an incremental number of languages

In real-life scenarios, it is common that a Multilingual classifier, trained on a set of languages, is required to be extended to new languages over time. Whereas the most straightforward approaches would be plain transfer learning (straightforward application of the old model to the new task) or re-training from scratch with the full extended dataset, both strategies present non-negligible drawbacks in their application to concrete problems. This need to allow Neural Networks to learn tasks in a sequential way is satisfied by Continual Learning (CL), of which we are giving a general setting in the following section. Later on, we deal with its application to our specific problem of sequential support of new languages for a m-BERT-based classifier.

6.1 Continual Learning

Although in recent years, machine learning models have been able to outperform humans on individual tasks, these impressive results have been obtained with static models, that need to reface the entire training process from scratch anytime new data comes out. In real-world set-ups, this practice is most often not exploitable, due to the streaming nature of data, storage limitations or privacy issues. Therefore, we would like Neural models to be able to learn incrementally, that is, to learn to solve a certain number (ideally infinite) of tasks by being exposed sequentially to each one of them. However, this strategy is observed to suffer from Catastrophic Forgetting (CF): the performance on previously acquainted domains usually significantly degrades when observing new data. This issue is named the stability-plasticity dilemma, which encodes the trade-off observed in the ability to integrate the latest observed distribution and the capability of retaining the previously observed ones[5].

Then, CL aims at restricting CF while having limited or restricted access to previous task data. The setting in which it operates is this: for each task t, data $(\mathcal{X}^{(t)}, \mathcal{Y}^{(t)})$ are randomly drawn from a distribution $D^{(t)}$, in which $\mathcal{X}^{(t)}$ are the instances and $\mathcal{Y}^{(t)}$ the corresponding ground-truth labels. The goal is then to control the statistical risk of all seen tasks up to the current one $\mathcal{T}$

$$\sum_{t=1}^{\tau} \mathbb{E}_{(\mathcal{X}^{(t)},\mathcal{Y}^{(t)})} \left[l(f_t(\mathcal{X}^{(t)}, \theta), \mathcal{Y}^{(t)}) \right]$$

having l as loss function, τ as the number of tasks seen until now, parameters θ and f_t as the network function for the t_{th} task. For $\mathcal{T}$, the statistical risk can be approximated by the empirical risk

$$\frac{1}{N_{\mathcal{T}}} \sum_{i=1}^{N_{\mathcal{T}}} l \left(f(x_i^{\mathcal{T}}; \theta), y_i^{\mathcal{T}} \right),$$

where $\{(x_i^{\mathcal{T}}, y_i^{\mathcal{T}}), i = 1, \ldots, N_{\mathcal{T}}\}$ is the set of $N_{\mathcal{T}}$ examples at our disposal for $\mathcal{T}$.

In our case, each task in the sequential schema is the ability to perform classification on an additional language, while retaining the most possible taxonomic accuracy on the previously seen ones.

In the next section, we introduce, from a theoretical point of view, a series of approaches that can be applied to tackle this problem. For each of them, the results of the related experiments are presented in 6.4

6.2 Extending a m-BERT-based classifier

Our setting consists of a multilingual BERT-based model $\mathcal{M}_{L_A}$, that, at each step, after being already finetuned for the downstream task of classification on a set of languages L_A, included in the set of languages L_P on which the m-BERT embedder has already been trained on, is being exploited for the same task on a new language $l_{new} \in L_P$. The approaches that we deal with are plain transfer learning, new-language-only fine-tuning, a Teacher-Student framework and retraining on a dataset containing annotated examples for all the languages. These methods are listed in increasing computational time and resource usage.

6.2.1 Naïve transfer learning

The Naïve transfer learning setting operates in the most straightforward way: it exploits the model that has been trained on L_A without any fine-tuning on l_{new} data. This option is suggested by the fact that $l_{new} \in L_P$, so the embedding model has already faced data in the l_{new} language during the pre-training stage. However, the performances in this framework are usually too poor for real-world applications and any form of fine-tuning on the new language is required.

6.2.2 New-language-only fine-tuning

The most naive approach to CL consists of fine-tuning $\mathcal{M}_{L_A}$ on supervised data $S_{l_{new}}$ in the new language only. Despite this method being cheap and able to achieve a good performance on the last step task l_{new}, it is the most subject to CF. In fact, since the network is exposed to the new distribution only, it tends to adapt its weights to fit it, at the cost of degrading its accuracy on the previously seen languages.

We need then to introduce some more solid CL techniques that, while still staying competitive at the computational level with respect to retraining from scratch with the full dataset, allow to retain previously acquired knowledge. A greedy solution is proposed by [32] as self-training: the model itself $\mathcal{M}_{L_A}$ is used to annotate some samples in the pre-learnt languages set L_A, so that this pseudo-labelled dataset $\tilde{S}_{L_A}$ is used jointly with $S_{l_{new}}$ in the l_{new} fine-tuning step. Despite this method's capability to reduce CF, it has been shown that it may also reinforce the errors of $\mathcal{M}_{L_A}$[13]. This is solved by the development of some more refined approaches in the context of CL, such as Elastic Weight Consolidation (EWC) and Knowledge Distillation (KD).

6.2.3 Elastic Weight Consolidation

When dealing with the most competitive methods in CL, Elastic Weight Consolidation (EWC) is sure worth mentioning[16]. CF is controlled by considering in the loss function a regularization term that penalizes variations on the model's weights that are decisive for the tasks learnt so far.

This algorithm takes inspiration from how knowledge is retained in the mammalian neocortex, via the so-called task-specific synaptic consolidation: knowledge about how to perform a previously acquired task is durably encoded in a subset of synapses that are made less plastic over time. The equivalent for an artificial neural network is to regulate learning sharpness based on weights' importance to previously seen tasks, so that determinant parameters stay close to their old values.

We can intuitively expect to find a solution for task B in an optimal region for A since when training with interleaving data the parameters of the networks are jointly optimized for both tasks.

From a theoretical point of view, we can rely on the result that many configurations of the parameters θ return the same architecture performance [35], [12]. Over-parametrization makes it likely that the optimal solution for B, θ^*_B, will be close to the optimal solution for A, θ^*_A. Then, the optimization problem is constrained via a quadratic penalty, that acting as a spring anchored to θ^*_A, gives the name elastic to the algorithm. The stiffness of this spring should be proportional to the contribution to the performance on task A, which is easily inferred in a probabilistic perspective

In fact, optimizing the parameters with respect to the statistical risk is equivalent to finding their most probable estimation given some data $\mathcal{D}$, so the maximum of $\log p(\theta|\mathcal{D})$, than can be obtained by using Bayes' theorem

$$\log p(\theta|\mathcal{D}) = logp(\mathcal{D}|\theta) + \log p(\theta) - \log p(\mathcal{D}), \tag{6.1}$$

in which $\log p(\mathcal{D}|\theta$ is just the negative of the loss function $-\mathcal{L}(\theta)$. When splitting our dataset in two slices, $\mathcal{D}_A$, $\mathcal{D}_B$, we obtain a reformulation of 6.1 that depends only on the loss on task B:

$$\log p(\theta|\mathcal{D}) = \log p(\mathcal{D}|\theta) = \log p(\mathcal{D}_B|\theta) + \log p(\theta|\mathcal{D}_A) - \log p(\mathcal{D}_B)$$

Then, all the information encoded by the parameters about task A must be retained in the posterior distribution $p(\theta|\mathcal{D}_A)$. Due to the intractability of the true posterior, it can be approximated as a Gaussian distribution with mean θ^*_A and diagonal covariance given by

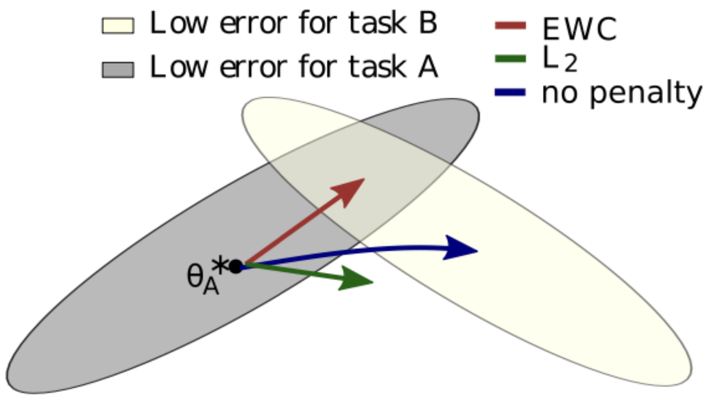


Figure 6.1: Training trajectories, with starting point θ_A^*, determined by the minimization of the loss on task B in a schematic parameter space

the Fisher information matrix F. The most important property of F is that it is equivalent to the second derivative of the loss near a minimum. Then, the optimization problem can be reframed as the minimization w.r.t. θ of

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \frac{\lambda}{2}\sum_i F_i(\theta_i - \theta_{A,i}^*)^2, \tag{6.2}$$

in which λ sets the trade-off between loss on the old and new task and i labels the network parameters.

6.2.3.1 Computing the Fisher Information matrix for a taxonomic classifier

One of the properties of F is the capability of approximating the Hessian while still being able to be computed from first-order derivatives only. In particular, it can be computed as:

$$F(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log p(x_i|\theta) \nabla_\theta \log p(x_i|\theta)^T \tag{6.3}$$

In the EWC setting, F is evaluated at θ_A^*, in order to constrain the most meaningful parameters for the previous task A.

In the taxonomic classification framework, $p(x_i, \theta)$ represents the joint distribution over the 4 tiers $p(x_i|\theta) = p(x_i, (T1, T2, T3, T4)|\theta)$.

As done throughout all this thesis, we make the (rather strong) assumption that the predictions of our network over the 4 tiers are independent, so that

$$p(x_i, (T1, T2, T3, T4)|\theta) = p(x_i, T1|\theta) \cdot p(x_i, T2|\theta) \cdot p(x_i, T3|\theta) \cdot p(x_i, T4|\theta)$$

This permits to express the gradient of the joint probability distribution in 6.3 as the sum of the gradients over the weights of the single-tier probabilities.

6.2.4 Teacher-Student framework

Another competitive field is represented by the Knowledge Distillation[13] framework, in which our teacher-student model arises. KD application for CF preventing is mainly exploited in Computer Vision, whereas in NLP it is more popular for model compression, having the DistilBert model as an example[33]. However, recently KD has been applied also to NLP tasks, at first for CL in Named Entity Recognition[27], then also for the task of mitigating CF in multilingual transformer-based models for semantic processing tasks [4]. This last exposed paper sets the basis for our study.

In this framework, the existing model, $\mathcal{M}_{L_A}$, acts as a teacher, imparting its knowledge about the languages it knows to a student model, that it is in the meanwhile trained on the new language. In our case, the student model is a clone of the teacher $\mathcal{M}_{L_A}$ and it is fine-tuned using a multi-loss function

$$\mathcal{L}_{CL} = \alpha \cdot \mathcal{L}_{\mathcal{T}} + (1 - \alpha) \cdot \mathcal{L}_{KD} \tag{6.4}$$

The first contribution to the loss $\mathcal{L}_{\mathcal{T}}$ is the task-specific loss, that we probably used also to train $\mathcal{M}_{L_A}$, as in our case, in which this term is given by the Cross-Entropy Loss function, computed on labelled $S_{l_{new}}$ samples. The key point of this technique is the presence of the distillation loss $\mathcal{L}_{KD}$, computed on the prediction of, respectively, the teacher and the student, on a set of unlabeled examples from the previous languages. So while $\mathcal{L}_{\mathcal{T}}$ gets the student to learn how to solve the text classification task on l_{new}, $\mathcal{L}_{KD}$ helps the student maintaining a consistent performance on L_A, via teacher imitation. The trade-off between the two contributions is controlled by the α parameter. We exploit the $\mathcal{L}_{KD}$ version proposed in [13]. At first, we need to define $z_i(x)$ as the output logits of the model's last layer when fed with the sample x. Then, the temperature-softmax converts the logits into a class-probability distribution:

$$y_i(x) = \frac{\exp\left(z_i/T\right)}{\sum_j \exp\left(z_j/T\right)}.$$

T is the temperature hyper-parameter, that controls how much the distribution is softened w.r.t. plain softmax. Finally, $\mathcal{L}_{KD}$ is computed as the cross-entropy loss between the teacher and student smoothed distributions $y_i^t(x)$, $y_i^s(x)$:

$$\mathcal{L}_{KD} = -T^2 \sum_i y_i^t(x) \log y_i^s(x),$$

where the T^2 is made necessary to compensate the $1/T^2$ factor provided by the smoothed softmax derivation. The key difference between self-training and this Student-Teacher model resides in the Temperature smoothing, which allows preserving of uncertainty on the teacher predictions. In fact, there is no control over the correctness of the teacher's prediction and this shows in the so-called genetic errors, i.e. errors that show in both the teacher and the student that learnt from it.

6.3 Training from scratch

Finally, we proceed to train our model with the complete dataset, containing annotated samples in all languages of interest. In this method, being just the traditional offline machine learning procedure, the loss minimization over the whole dataset pushes the weights of the network in a joint optimal region for all the languages considered.

	$AFTA$			
	T1	T2	T3	T4
Spanish	58.0	45.2	34.9	34.3
Swedish	40.6	26.6	20.9	20.8

	$AFDR$	$AFBA$
Spanish	43.1	34.3
Swedish	27.2	20.8

Table 6.1: Taxonomic metrics for naïve transfer learning approach after probability-based sorting on Spanish and Swedish test dataset

6.4 Experimental results

6.4.1 Naïve transfer learning

As a first approach, we evaluate our calibrated baseline on Spanish and Swedish reserved test datasets. The model has not been trained with samples on any of the two languages, then this approach reveals how keen it is on transferring knowledge to unseen data. Taxonomic metrics obtained from the evaluation on Spanish and Swedish articles are shown in table 6.1. We start our inspection with results in Spanish. Taking $AFBA$ as a reference, we see that it degrades from 79.3, when computed for the languages it has been trained on, to 34.3 for Spanish.

The performance loss for Swedish is even higher, with $AFBA = 20.8$. The performance gap displayed between Spanish and Swedish evaluations can be easily justified by the strong syntactic and semantic similarity of the Spanish language with the Italian one, both belonging to the family of romance languages. Despite Swedish and English both belonging to the Germanic linguistic group, their difference is due to the influence that Latin has had on English only. This makes Swedish an outlier in our set of languages.

6.4.2 New-language-only fine-tuning

In this section, we display the results of the experiments drawn with the fine-tuning technique. As already explained, it enlarges the set of languages known by a model by performing an additional training phase with data in l_{new} language only. Since we want to study how each of the following techniques performs when exposed to an incremental number of languages, the experiments will be performed sequentially. This means, in the fine-tuning case, that we have two distinguished models, both derived from the calibrated baseline. The first will be fine-tuned on Spanish data first, and once this language is acquired, Swedish will be added via fine-tuning too. The second will face the same process but with new languages learning in reversed order: the fine-tuning phase is performed on Swedish first and on Spanish afterwards. Then, to assess how each CL technique performs, we will consider its performance as the average of the metrics on these two models: this will remove any eventual dependency on language order in the sequential learning approach. All the training phases in this set of experiments are performed minimizing the Cross Entropy loss using Adam as an optimizer, with adaptive learning rate depending on *epoch* as $lr = 10^{-5}/(1+0.05*epoch)$. The maximum number of training epochs is set to 30, with Early Stopping monitoring the validation accuracy on Tier 3, with the patience parameter set to 5. The batch size is set to 16.

	k=1				k=2				k=3			
	$AFTA$				$AFTA$				$AFTA$			
	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4
Italian+English	89.3	83.2	79.6	79.4	84.8	76.8	72.1	71.8	77.6	66.2	59.5	58.9
Spanish	58.0	45.2	34.9	34.3	90.0	85.6	82.0	81.7	78.5	67.7	60.6	60.0
Swedish	40.6	26.6	20.9	20.8	39.1	26.0	20.6	20.4	84.0	76.5	71.9	71.8

	k=1		k=2		k=3	
	$AFDR$	$AFBA$	$AFDR$	$AFBA$	$AFDR$	$AFBA$
Italian+English	82.9	79.4	76.3	71.8	65.6	58.9
Spanish	43.1	34.3	84.8	81.7	66.7	60.0
Swedish	27.2	20.8	26.5	20.4	76.0	71.8

Table 6.2: Taxonomic metrics on the model obtained fine-tuning the baseline on Spanish first, and on Swedish at the following step

6.4.2.1 Fine-tuning on Spanish first

Here we expose the results of the first of the two subsets of experiments for the fine-tuning technique. As mentioned just above, we integrate Spanish at first, and Swedish sequentially. The results are exposed as follows: in order to study the capability of the algorithm for an incremental number of language extension, we individuate the following training phases, marked with an index k. For $k = 1$, we are just evaluating the baseline trained on Italian and English data only. When $k = 2$, we are evaluating the model after acquiring Spanish, that becomes our starting point for incorporating also Swedish knowledge, at the $k = 3$ step. Results are shown in table 6.2. For $k = 1$, we simply retrieve the already known metrics for the baseline, which we now compare with their equivalent at $k = 2$. At this second step, the effect of fine-tuning on Spanish is evident: $AFBA$ evaluated on Spanish escalates from 34.3 to 81.7 and we observe an overall important raise in all the considered metrics. On the other hand, we observed a decrease in performance in the previously acquired languages, i.e. Italian and English. This behaviour reveals that, as expected, fine-tuning forces the distribution of the weights towards the one that has been seen as last, at the cost of deforming the previous one. The metrics on Swedish face a slight reduction, which may be due to the network reinforcement in spotting typical features of romance languages. Finally, for $k = 3$, we obtain a Swedish specialized model: it outperforms all other languages, despite the huge gap in metrics at previous steps. Clearly, this comes at the cost of a further downgrading in Italian and English samples and of partial forgetting of Spanish too.

6.4.2.2 Fine-tuning on Swedish first

In this section, we deal with the same framework exposed in the previous paragraph but with fine-tuning steps facing Swedish at first and Spanish afterwards. For $k = 2$, all the metrics reveal the network adjustment to Swedish data distribution: we observe a jump in $AFBA$ from a value of 20.8 to 72.3, with a consequent drop in performance in Spanish, Italian and English. An interesting fact can be noticed by considering the effect of adding Spanish or Swedish first on the other: when adding Spanish, $AFBA$ on Swedish worse by just 2%, whereas when adding Swedish first, $AFBA$ on Spanish drops by 22%. The second case is also characterized by a more severe fall in Italian and English metrics with

	k=1				k=2				k=3			
	$AFTA$				$AFTA$				$AFTA$			
	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4
Italian+English	89.3	83.2	79.6	79.4	79.6	69.2	63.3	62.7	83.2	74.4	69.7	69.5
Swedish	40.6	26.6	20.9	20.8	84.2	76.7	72.7	72.3	76.9	68.4	63.2	63.1
Spanish	58.0	45.2	34.9	34.3	51.1	36.4	27.2	26.3	89.0	82.5	78.6	78.2

	k=1		k=2		k=3	
	$AFDR$	$AFBA$	$AFDR$	$AFBA$	$AFDR$	$AFBA$
Italian+English	82.9	79.4	68.7	62.7	74.2	69.5
Swedish	27.2	20.8	76.5	72.3	67.9	63.1
Spanish	43.1	34.3	35.3	26.3	82.1	78.2

Table 6.3: Taxonomic metrics on the model obtained fine-tuning the baseline on Swedish first, and on Spanish at the following step

respect to the first, revealing how the network needs to face stronger reshaping to adapt to Swedish. Moving to $k = 3$, the model faces fine-tuning on Spanish. What is definitely worth to be mentioned is that this step reveals to favour Italian and English too. In fact, all taxonomic metrics face a non-negligible rise when the model learns Spanish after specializing on Swedish, ending in higher metrics at step $k = 3$ for the reverse ordering on training languages in the previous section. This highlights again the similarity between Spanish, Italian at first and English, which is displayed in a correlation in performance trends. However, this adaptation hinders, reasonably, the performance on Swedish.

Giving an overall picture of the sequential fine-tuning method, it shows promising in the language learnt at current step, at the cost of forgetting previously acquired information.

6.4.3 EWC

In this section, we display the results of the experiments drawn in the context of EWC, which dampens CF by including a regularization term in the loss that freezes the most significant weights for previous tasks. Again, to have a fair sequential learning process evaluation, we perform two lines of experiments, distinguished by the order of new languages integration. Each of the two sub-processes will face two training phases: the first is exposed to Spanish initially and to Swedish later on, and the second absorbs Swedish first. All the training phases in this set of experiments are performed minimizing the loss function proposed in 6.1, with the Cross Entropy as the loss-specific task, using Adam as an optimizer, with adaptive learning rate depending on *epoch* as $lr = 10^{-5}/(1+0.05*epoch)$. The maximum number of training epochs is set to 25, with Early Stopping monitoring the validation accuracy on Tier 3, with the patience parameter set to 5. The batch size is set to 16. The regularization parameter λ is set to 0.5 as we assign the same importance to old and new tasks.

6.4.3.1 Adding Spanish first

In this approach, we want our model to learn Spanish at first while retaining the most knowledge possible on previously acquired languages. In this case, then, the approximated

	k=1				k=2				k=3			
	$AFTA$				$AFTA$				$AFTA$			
	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4
Italian+English	89.3	83.2	79.6	79.4	87.8	81.0	77.4	77.1	85.3	77.4	72.5	72.1
Spanish	58.0	45.2	34.9	34.3	88.6	82.5	78.5	78.1	83.4	73.4	67.5	67.0
Swedish	40.6	26.6	20.9	20.8	41.3	28.2	22.6	22.5	82.1	72.9	67.4	67.1

	k=1		k=2		k=3	
	$AFDR$	$AFBA$	$AFDR$	$AFBA$	$AFDR$	$AFBA$
Italian+English	82.9	79.4	80.8	77.1	76.8	72.1
Spanish	43.1	34.3	81.9	78.1	72.8	67.0
Swedish	27.2	20.8	28.7	22.5	72.4	67.1

Table 6.4: Taxonomic metrics on the model obtained with EWC continual learning algorithm adding Spanish first, and on Swedish at the following step

Fisher matrix is computed with 5000 samples from the Italian and English validation dataset. Moving to the next step, marked with $k = 3$ in table 6.4, we now have a model that knows Italian, English and Spanish and is being trained on Swedish. The approximated Fisher matrix is now obtained via processing 5000 examples sampled from the Italian, English and Spanish validation dataset. We now deal with the results of this subset of experiments. Considering the behaviour on Italian and English dataset, we observe that, despite the performance loss still being present, its magnitude is smaller with respect to the fine-tuning framework, revealing the effectiveness of the implemented constraint. The same behaviour is shown by Spanish metrics when adding Swedish at $k = 3$, which again outperforms the fine-tuned model at same step and for same procedure, as it can be seen in the last column of table 6.2. The metrics on the last added language, however, are slightly lower than the corresponding ones for the fine-tuned model. For example, in the EWC approach, $AFBA$ for Spanish at $k = 2$ reaches 71.8, whereas in the fine-tuning experiment it touches 81.7, at the cost of stronger CF.

6.4.3.2 Adding Swedish first

In this setting, the technique is the same as the one discussed in the previous paragraph, but with reversed languages exposition. At $k = 2$, still, the approximated Fisher information matrix is computed with 5000 validation samples from Italian and English articles, whereas at step $k = 3$, the 5000 examples include also items in Swedish. Outcomes can be inspected from table 6.5 Again, we overall observe a better retaining of past information than plain fine-tuning, at the price of slightly more restrained performances on last step language.

6.4.4 Teacher-Student framework

The teacher-student setting exploits a Knowledge Distillation setting for CL. When the learning is performed sequentially, the trained student at step k becomes the teacher at step $k + 1$. In our case, at step $k = 2$, the teacher is the calibrated baseline and the student acquires either Spanish or Swedish from the corresponding training data. At $k = 3$, the 3 language-speaking student will now be the teacher to its clone, which

	k=1				k=2				k=3			
	$AFTA$				$AFTA$				$AFTA$			
	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4
Italian+English	89.3	83.2	79.6	79.4	86.9	79.2	75.0	74.7	86.2	78.9	75.0	74.6
Swedish	40.6	26.6	20.9	20.8	82.7	73.8	69.1	68.8	78.1	68.2	63.1	62.7
Spanish	58.0	45.2	34.9	34.3	55.7	41.9	32.9	32.0	88.0	80.9	76.3	75.9

	k=1		k=2		k=3	
	$AFDR$	$AFBA$	$AFDR$	$AFBA$	$AFDR$	$AFBA$
Italian+English	82.9	79.4	78.9	74.7	78.7	74.6
Swedish	27.2	20.8	73.6	68.8	68.0	62.7
Spanish	43.1	34.3	40.6	32.0	80.2	75.9

Table 6.5: Taxonomic metrics on the model obtained with EWC continual learning algorithm adding Swedish first, and on Swedish at the following step

will be fed with either Swedish or Spanish training samples. All the training phases in this set of experiments are performed by minimizing the multi-loss function proposed in 6.4, with both the task-specific loss and the distillation loss functions being the Cross Entropy. The chosen optimizer is Adam, with adaptive learning rate depending on *epoch* as $lr = 10^{-5}/(1+0.05*epoch)$. The maximum number of training epochs is set to 25, with Early Stopping monitoring the validation loss of the student network, with the patience parameter set to 5. The batch size is set to 16. The trade-off parameter α is set to 0.5 At each training step, a batch of training items from the old languages is sampled from a uniform distribution on the corresponding dataset. Their ground-truth label can be neglected, if available, as the student learns from the teacher-produced logits.

6.4.4.1 Learning Spanish first with baseline as teacher

In this paragraph, we inspect the results of this CL method when exposed to Spanish first. For $k = 2$, the student has learnt Spanish, while being reminded of Italian and English distributions by the calibrated baseline, acting as its teacher. This first step is performed both for $T = 1$, in which the logits are exploited as they are, and for $T = 3$, in which the reduced logits result in a smoother softmax output. When looking at table 6.6, we can notice how this method, along with granting an optimal performance on the student's freshly learnt language, is able to strengthen the knowledge of the past observed ones. This phenomenon is known as cross-language transfer and can be spotted by observing that all metrics in Italian and English increase from $k = 1$ to $k = 2$ and then from $k = 2$ to $k = 3$. It is interesting to notice that integrating Spanish turns out to favour also Swedish, raising $AFBA$ from 20.8 to 23.5. A major increase is then noticed when Swedish data actually faces training, with $AFBA$ reaching 67.6. A slight increase is also faced by Spanish-related metrics and, as already mentioned, by Italian and English ones.

We now study the effect of raising the temperature parameter in the distillation loss, setting it to $T = 3$. However, this model under-performs the non-smoothed one: at $k = 2$, $AFBA = 75.1$, while reaching 79.7 for $T = 1$. The reinforcement effect is still observed both in the previously seen and in the not seen languages.

T=1	k=1				k=2				k=3			
	$AFTA$				$AFTA$				$AFTA$			
	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4
Italian+English	89.3	83.2	79.6	79.4	89.6	83.5	80.2	79.8	89.7	83.6	80.3	80.0
Spanish	58.0	45.2	34.9	34.3	89.4	83.7	80.1	79.7	89.6	83.6	80.1	79.8
Swedish	40.6	26.6	20.9	20.8	43.1	29.4	23.7	23.5	81.7	73.0	67.8	67.6

	k=1		k=2		k=3	
	$AFDR$	$AFBA$	$AFDR$	$AFBA$	$AFDR$	$AFBA$
Italian+English	82.9	79.4	83.3	79.8	83.4	80.0
Spanish	43.1	34.3	83.2	79.7	83.3	79.8
Swedish	27.2	20.8	29.9	23.5	72.5	67.6

Table 6.6: Taxonomic metrics on student, learning at first Spanish, and Swedish at the following step, for T=1

T=3	k=1				k=2				k=3			
	$AFTA$				$AFTA$				$AFTA$			
	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4
Italian+English	89.3	83.2	79.6	79.4	89.5	83.4	80.0	79.7	89.5	83.5	80.0	79.7
Spanish	58.0	45.2	34.9	34.3	87.5	80.4	75.5	75.1	88.0	80.6	75.5	74.9
Swedish	40.6	26.6	20.9	20.8	43.8	30.5	24.4	24.0	80.3	70.6	65.0	64.9

	k=1		k=2		k=3	
	$AFDR$	$AFBA$	$AFDR$	$AFBA$	$AFDR$	AFBA
Italian+English	82.9	79.4	83.2	79.7	83.2	79.7
Spanish	43.1	34.3	79.6	75.1	79.8	74.9
Swedish	27.2	20.8	30.7	24.0	70.2	64.9

Table 6.7: Taxonomic metrics on student, learning Spanish from the baseline, for T=3

T=1	k=1				k=2				k=3			
	$AFTA$				$AFTA$				$AFTA$			
	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4
Italian+English	89.3	83.2	79.6	79.4	89.6	83.6	80.4	80.1	89.6	83.7	80.5	80.2
Swedish	40.6	26.6	20.9	20.8	83.0	74.5	69.9	69.7	82.7	74.5	70.1	69.8
Spanish	58.0	45.2	34.9	34.3	58.0	45.2	36.3	35.7	87.6	80.1	75.0	74.5

	k=1		k=2		k=3	
	$AFDR$	$AFBA$	$AFDR$	$AFBA$	$AFDR$	$AFBA$
Italian+English	82.9	79.4	83.4	80.1	83.4	80.2
Swedish	27.2	20.8	74.2	69.7	74.3	69.8
Spanish	43.1	34.3	43.8	35.7	79.3	74.5

Table 6.8: Taxonomic metrics on student, learning at first Swedish, and Spanish at the following step, for T=1

T=3	k=1				k=2				k=3			
	$AFTA$				$AFTA$				$AFTA$			
	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4
Italian+English	89.3	83.2	79.6	79.4	89.5	83.2	79.9	79.7	89.5	83.6	80.2	79.9
Swedish	40.6	26.6	20.9	20.8	80.2	70.5	65.5	65.0	80.5	70.5	65.7	65.1
Spanish	58.0	45.2	34.9	34.3	59.0	46.9	38.2	37.5	85.3	76.1	69.8	69.3

	k=1		k=2		k=3	
	$AFDR$	$AFBA$	$AFDR$	$AFBA$	$AFDR$	AFBA
Italian+English	82.9	79.4	83.1	79.7	83.3	79.9
Swedish	27.2	20.8	70.3	65.0	70.5	65.1
Spanish	43.1	34.3	45.4	37.5	75.1	69.3

Table 6.9: Taxonomic metrics on student, learning at first Swedish, and Spanish at the following step, for T=3

6.4.4.2 Learning Swedish first with baseline as teacher

The process now still starts with the calibrated baseline acting as the teacher, but now the student is fed with the Swedish training articles. At first, we set $T = 1$, letting the student learn from the plain teacher logits. The behaviour is the same as in the reversed order experiment, exposed in the previous paragraph: adding a new language strengthens, or at worse, leave untouched, the network capability on the already acquired ones.

Now, we raise the temperature in our experiment in order to evaluate the effect of smoothed predictions. Again, while it still allows retaining the knowledge on the languages seen up to the point, it displays lower performances on the language to be learnt.

6.4.5 Re-training from scratch

Finally, we report the results for a non-CL strategy, in which the model is trained from scratch on all the languages of interest, just as plain offline machine learning. This approach represents a sort of upper bound to CL methods, as it allows the optimizer to look for the optimal parameters configuration on the whole dataset. We make our experiments

	k=1				k=2				k=3			
	$AFTA$				$AFTA$				$AFTA$			
	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4
Italian+English	89.3	83.2	79.6	79.4	90.0	84.0	80.6	80.3	89.9	83.9	80.6	80.3
Spanish	58.0	45.2	34.9	34.3	90.1	84.3	80.9	80.6	90.0	84.8	81.3	81.1
Swedish	40.6	26.6	20.9	20.8	40.5	27.1	21.8	21.7	83.3	75.5	71.3	71.2

	k=1		k=2		k=3	
	$AFDR$	$AFBA$	$AFDR$	$AFBA$	$AFDR$	$AFBA$
Italian+English	82.9	79.4	83.7	80.3	83.7	80.3
Spanish	43.1	34.3	84.8	80.6	84.3	81.1
Swedish	27.2	20.8	27.7	21.7	75.3	71.2

Table 6.10: Taxonomic metrics on the model trained from scratch adding Spanish first to the baseline, and Swedish too at the following step

	k=1				k=2				k=3			
	$AFTA$				$AFTA$				$AFTA$			
	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4
Italian+English	89.3	83.2	79.6	79.4	89.5	83.9	80.6	80.3	89.9	83.9	80.6	80.3
Swedish	40.6	26.6	20.9	20.8	84.2	76.6	72.6	72.3	83.3	75.5	71.3	71.2
Spanish	58.0	45.2	34.9	34.3	59.8	45.6	37.1	36.4	90.0	84.8	81.3	81.1

	k=1		k=2		k=3	
	$AFDR$	$AFBA$	$AFDR$	$AFBA$	$AFDR$	$AFBA$
Italian+English	82.9	79.4	83.6	80.3	83.7	80.3
Swedish	27.2	20.8	76.4	72.3	75.3	71.2
Spanish	43.1	34.3	44.7	36.4	84.3	81.1

Table 6.11: Taxonomic metrics on the model trained from scratch adding Swedish first to the baseline, and Spanish too at the following step

ordering-proof by performing two parallel training processes.

6.4.5.1 Adding Spanish First

The first training process includes Italian, English and Spanish for $k = 2$, granting good results in all the languages exploited during training. The last step is interested by a training phase including all 4 languages and it is effectively the best performing model among the ones proposed.

6.4.5.2 Adding Swedish First

In this case for $k = 2$, the training dataset is built with the articles in Italian, English and Swedish. It is worth noticing that with respect to the model at step 3, which has learnt also Spanish, it reaches higher performance in Swedish: this is probably due to the fact that the absence of Spanish data allows the network to shape its weights following better Swedish distribution.

	k=1				k=2				k=3			
	$AFTA$				$AFTA$				$AFTA$			
	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4
Fine-tuning	89.3	83.2	79.6	79.4	87.1	81.2	77.4	77.0	86.5	79.5	75.3	75.0
EWC	89.3	83.2	79.6	79.4	85.7	78.2	73.8	73.5	85.1	76.9	71.9	71.5
Teacher-student	89.3	83.2	79.6	79.4	86.3	79.1	75.0	74.7	84.7	76.6	71.4	71.1
Full training	89.3	83.2	79.6	79.4	86.4	79.1	75.0	74.7	85.7	80.2	76.3	76.2

	k=1		k=2		k=3	
	$AFDR$	$AFBA$	$AFDR$	$AFBA$	$AFDR$	$AFBA$
Fine-tuning	82.9	79.4	75.3	77.0	79.1	75.0
EWC	82.9	79.4	77.8	73.5	76.3	71.5
Teacher-student	82.9	79.4	78.7	74.7	75.9	71.1
Full training	82.9	79.4	78.8	74.7	79.8	76.2

Table 6.12: Taxonomic metrics for the languages learnt at the current step.

6.4.6 Methods Comparison

In order to have a fair comparison between the various methods exposed in this chapter, we now analyze their performances on the languages they have been trained on at the current step, on the languages it has faced in previous steps, on the ones seen up to the current step, and finally on the languages it has not been trained on yet. For each of the listed cases above, we obtain a general performance indicator by averaging the metrics obtained for the two independent experimental paths, that have been trained on data, respectively, at first in Spanish and in Swedish later on, and in Swedish first and in Spanish afterwards.

6.4.6.1 Average performance for the languages learnt at the current step

We now compare the average performance obtained for the languages on whose the model has been trained at the current step.

For both models, the observed languages at $k = 1$ are English and Italian. For $k = 2$, the first model is trained on Spanish and the second on Swedish. For $k = 3$, the second model is trained on Swedish and the second on Spanish.

Average results are reported in table 6.12 and can be visualized in figure 6.2 We observe an overall decreasing trend at step $k = 2$. This is mainly due to the inclusion of Swedish in the training data, on which metrics are always worse than on the other languages at our disposal. This effect is probably justified by both the fact that Swedish lacks the Latin influence that other languages share and by structural limitations of the embedder derived from an uneven distribution among languages of the data it has been trained on. At step $k = 3$, the best performance is achieved by the full training from scratch procedure, as we could expect, followed by the fine-tuning technique. This result is straightforward to be understood since during fine-tuning the network weights are shaped on the new task distribution. On the other hand, proper CL methods, i.e. EWC and the Teacher-student framework, turn out to be integrating the least amount of knowledge from the last observed dataset.

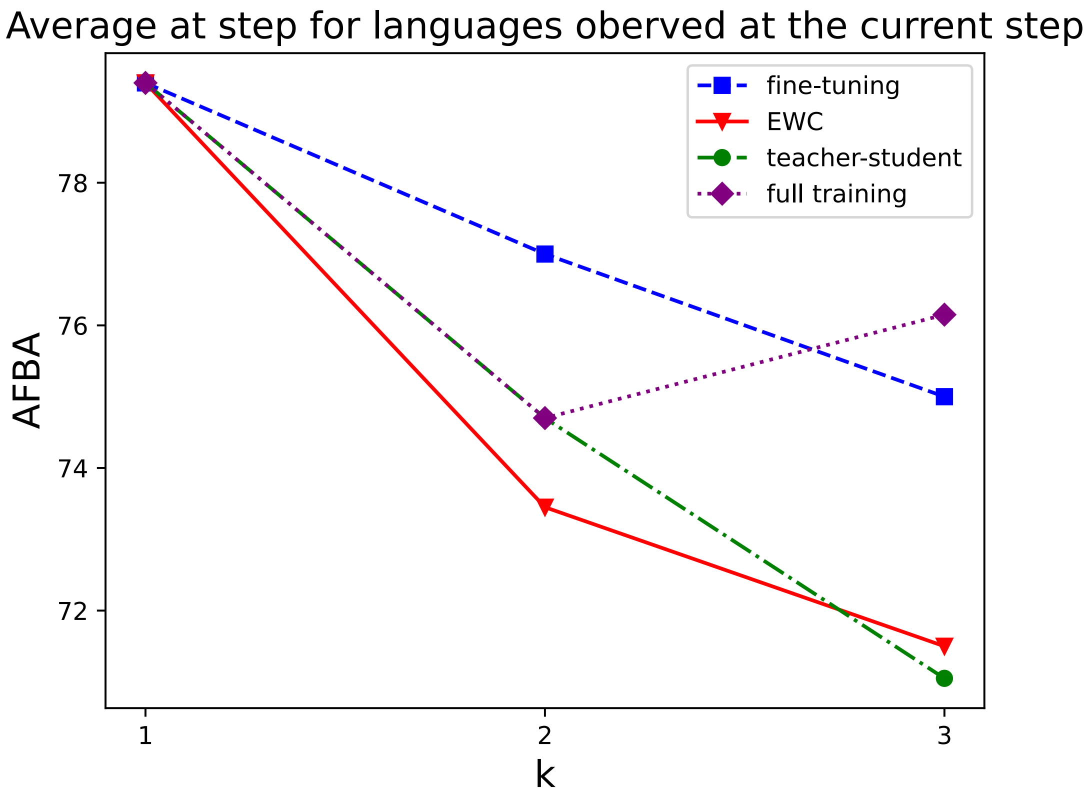


Figure 6.2: Average performances on the languages learnt at the current step.

	k=1				k=2				k=3			
	$AFTA$				$AFTA$				$AFTA$			
	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4
Fine-tuning	-	-	-	-	82.2	73.0	67.7	67.3	79.5	69.5	63.7	63.3
EWC	-	-	-	-	87.4	80.1	76.2	75.9	84.1	75.6	70.8	70.5
Teacher-student	-	-	-	-	89.6	83.6	80.3	80.0	88.4	82.0	78.5	78.3
Full training	-	-	-	-	88.5	81.6	77.8	77.5	88.8	82.6	79.2	78.9

	k=1		k=2		k=3	
	$AFDR$	$AFBA$	$AFDR$	$AFBA$	$AFDR$	$AFBA$
Fine-tuning	-	-	76.8	67.3	69.0	63.3
EWC	-	-	79.9	75.9	75.3	70.5
Teacher-student	-	-	83.4	80.0	81.8	78.3
Full training	-	-	81.3	77.5	82.4	78.9

Table 6.13: Taxonomic metrics for the languages learnt at previous steps.

6.4.6.2 Average performance for the languages learnt at previous steps

Now we focus our inspection on the performance of the proposed algorithms on previously learnt languages.

Clearly, this evaluation can not be performed for the first step. For $k = 2$, metrics are evaluated on Italian and English, whereas for $k = 3$ they are evaluated, respectively, on Swedish and on Spanish. Finally, the average on sub-experiments, distinguished by training order, is performed. Again, the upper-bound $AFBA$ is gained by the non-CL approach, in which all the languages seen up to step k are included in the training. However, the KD-based approach, i.e. the teacher-student algorithm, displays a negative gap in $AFBA$ of just 0.6, whereas this gap is of 8.4 and 15.6 for EWC and fine-tuning, respectively. Quantitative results can be found in table 6.13, whereas figure 6.3 offers a visual inspection of $AFBA$. The behaviour we observe is exactly the expected one: the non-CL approach of fine-tuning tends to modify its weights to adapt to the last-seen distribution, while sequentially losing previously acquired skills. On the other hand, EWC offers a better retention of past information, but it can not compete with the teacher-student framework. The difference between these two CL approaches can be justified by the distinct routines used by the two to encode past information: while EWC just retains the previous distribution in the approximated Fisher Information matrix, the Teacher-Student algorithm keeps being trained on batches of unlabelled data belonging to the previous task. The second approach clearly allows a much higher complexity for the information retain goal.

6.4.6.3 Average performance for the languages learnt up to the current step

Now, we deal with the comparison of average performances on all the languages that have been included in training up to now. These indicators reveal which approach dominates for the totality of tasks that we are interested in, up to a certain time.

Full taxonomic metrics are shown in table 6.14, whereas $AFBA$ trend over step k is exposed in figure 6.4 to offer an immediate understanding. Again, the time-consuming full training proved to be successful, with $AFBA = 78.1$ for $k = 3$. However, the Student

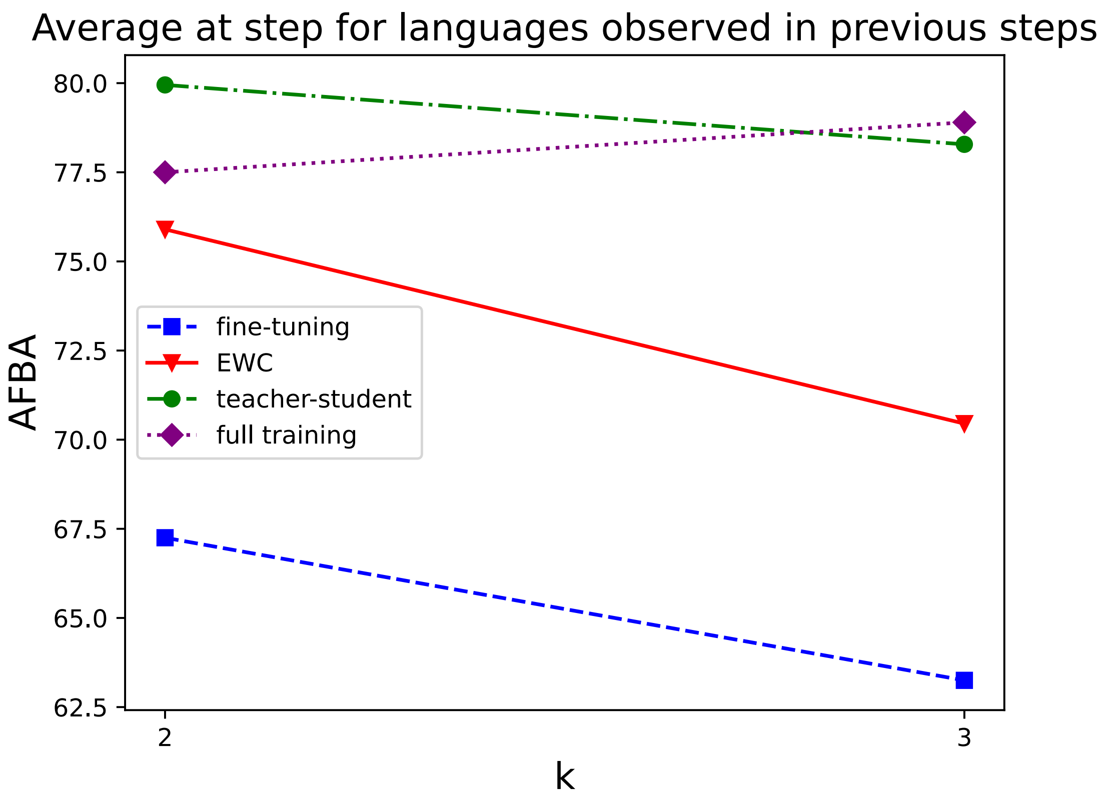


Figure 6.3: Average performances on the languages learnt at previous steps.

	k=1				k=2				k=3			
	$AFTA$				$AFTA$				$AFTA$			
	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4
Fine-tuning	89.3	83.2	79.6	79.4	83.9	75.8	71.0	70.6	84.3	75.4	70.0	69.4
EWC	89.3	83.2	79.6	79.4	86.8	79.3	75.4	75.0	84.3	75.9	71.0	70.7
Teacher-student	89.3	83.2	79.6	79.4	88.4	82.0	78.4	78.1	87.4	80.6	76.7	76.4
Full training	89.3	83.2	79.6	79.4	87.7	80.6	76.8	76.5	88.2	82.0	78.4	78.1

	k=1		k=2		k=3	
	$AFDR$	$AFBA$	$AFDR$	$AFBA$	$AFDR$	$AFBA$
Fine-tuning	82.9	79.4	75.3	70.6	74.8	69.4
EWC	82.9	79.4	79.1	75.0	75.5	70.7
Teacher-student	82.9	79.4	81.7	78.1	80.3	76.4
Full training	82.9	79.4	80.4	76.5	81.6	78.1

Table 6.14: Taxonomic metrics for the languages learnt up to the current step.

network in the KD approach competes with $AFBA = 76.4$, which comes at a much lower computational cost. EWC and fine-tuning retrieve less interesting results, with $AFBA = 70.7$ and $AFBA = 69.4$

6.4.6.4 Average performance for the languages that have not been learnt yet

Finally, we study how sensitive each approach is to the phenomenon of cross-lingual transfer by inspecting its performance on the languages that it has never been trained on.

We can inspect results from table 6.15 in figure 6.5. What we see is that the Teacher-student method offers a much higher performance on the languages it has not faced yet, with $AFBA$ raising to 29.6 at step $k = 2$, from a starting point of 27.3. EWC and the full training procedure are keener to keep it stable, with respectively $AFBA = 27.3$ and 26.9. Finally, fine-tuning displays the lowest observed cross-lingual transfer, as it shows $AFBA = 23.4$.

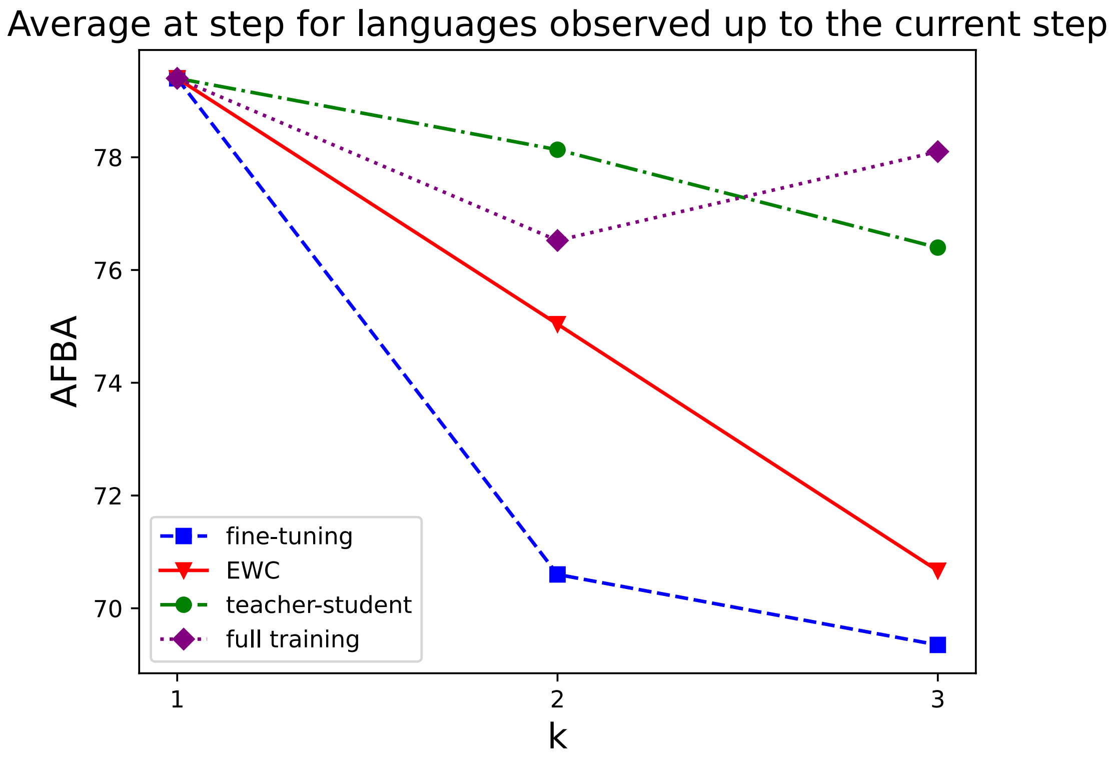


Figure 6.4: Average performances on the languages learnt up to the current step.

	k=1				k=2				k=3			
	$AFTA$				$AFTA$				$AFTA$			
	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4
Fine-tuning	49.0	35.6	27.7	27.3	45.1	31.2	23.9	23.4	-	-	-	-
EWC	49.0	35.6	27.7	27.3	48.5	35.1	27.8	27.3	-	-	-	-
Teacher-student	49.0	35.6	27.7	27.3	50.6	37.3	30.0	29.6	-	-	-	-
Full training	49.0	35.6	27.7	27.3	48.1	34.5	27.4	26.9	-	-	-	-

	k=1		k=2		k=3	
	$AFDR$	$AFBA$	$AFDR$	$AFBA$	$AFDR$	$AFBA$
Fine-tuning	34.9	27.3	30.9	23.4	-	-
EWC	34.9	27.3	34.7	27.3	-	-
Teacher-student	34.9	27.3	36.9	29.6	-	-
Full training	34.9	27.3	34.2	26.9	-	-

Table 6.15: Taxonomic metrics for the languages that have not been learnt yet.

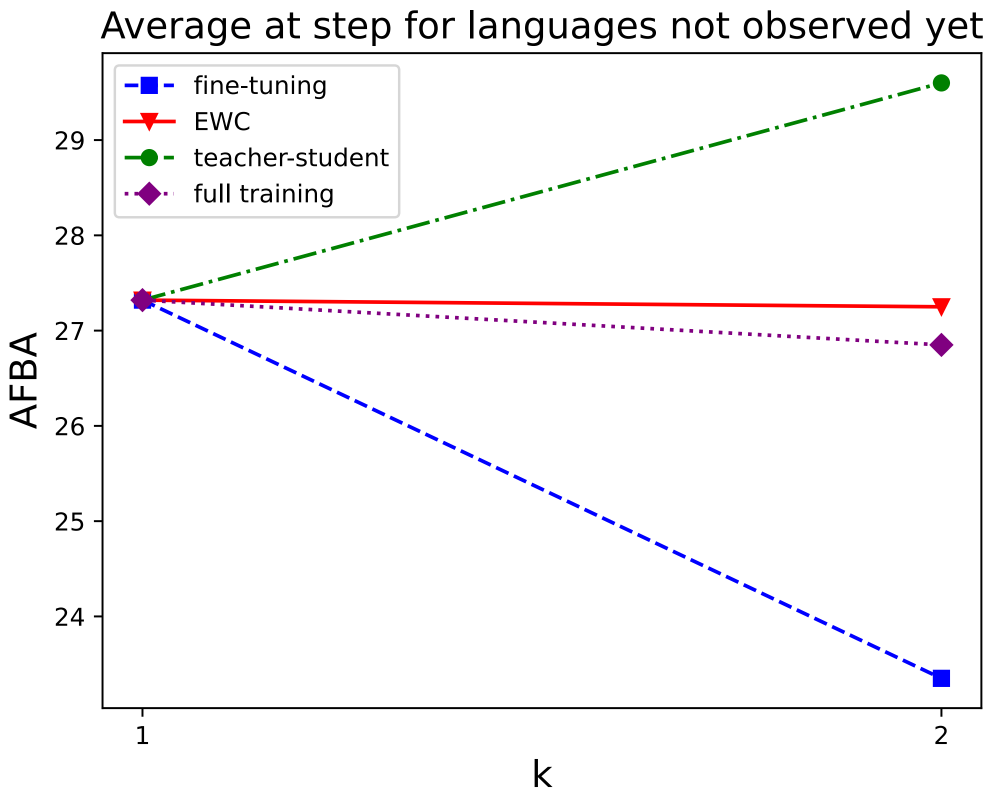


Figure 6.5: Average performances on the languages that have not been learnt yet.

Chapter 7

Conclusion

In this thesis, we have been dealing with the implementation of a taxonomic multilingual text classifier and with its progressive extension to an incremental number of languages.

We use a multilingual deep BERT embedder for the features extraction task and we interface it with a stack of dense layers for the downstream task of classification.

Our model does not impose any a priori taxonomic consistency between predicted tiers, which is retrieved during the post-processing phase via a joint probability-based sorting algorithm.

After obtaining our multilingual baseline, which has been trained on Italian and English article slices obtained from HTML files, we correct its tendency of being over-confident via the temperature scaling calibration algorithm. The extension of this algorithm to the multi-output case is one of the main contributions of this thesis.

The calibrated baseline has been the starting point for the comparison of various methods for obtaining a model that can solve the taxonomic classification task in an incremental number of languages. The naïve application of this model to unseen languages turns out to be too inaccurate for real-life applications. A possible approach would be the offline machine learning technique of training the model from scratch on the full dataset containing all the languages of interest. Despite representing the upper-bound method, this procedure comes at the highest computational cost and it may not be applicable, as it relies on the full availability of labelled training data.

A cheaper approach for sequential learning could be the fine-tuning of the baseline on new language data only. This method reaches optimal performance on the last acquired task but it is particularly subject to catastrophic forgetting.

We then move to the Continual Learning framework, in which we test two algorithms: the Knowledge-Distillation inspired Teacher-Student setting and the Elastic Weight Consolidation method. The extension of the EWC algorithm to a multivariate output distribution counts among the main contributions of this work. Both these CL approaches achieve better performance with respect to the plain fine-tuning on previously seen languages while being much computationally cheaper than training from scratch. Among the two, the Teacher-Student framework offers an overall better performance on tasks seen up to the current one and it represents a competitive alternative to traditional offline machine learning.

This work represents a starting point for further research in taxonomic multilingual text classification. In particular, a deeper inspection of the sequential learning behaviour for a higher number of languages should be carried out for the different algorithms. Moreover, another improvement could be obtained with the compensation of the high class imbalance, either via an up-sampling phase or by the usage of specific loss functions, such as the focal loss[18].

Chapter 8

Appendix

The t-SNE algorithm belongs to the class of *Stochastic Neighbour Embedding* (SNE) methods. It operates via associating a Gaussian probability distribution p_{ij} to $\boldsymbol{x}_j \in \mathbb{R}^p$ belonging to the neighbourhood of $\boldsymbol{x}_i$, based on their Euclidean distance:

$$p_{ij} = \frac{\exp(-\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|^2)/2\sigma_i^2}{\sum_{k \neq i} \exp(-\left\|\boldsymbol{x}_i - \boldsymbol{x}_k\right\|^2)/2\sigma_i^2}$$

The tuning of the σ_i parameter is achieved by fixing the local entropy $H(p_i)$:

$$H(p_i) \equiv -\sum_j p_{ji} \log_2(p_{ji})$$

to the same constant across all data points $X = \{\boldsymbol{x}_i\}$ via user-constraining the perplexity $\Sigma = 2^{H(p_i)}$. The perplexity can be interpreted as a measure of the effective number of neighbours and its typical value ranges between 5 and 50 [20], [22]. This strategy identifies $\sigma_i \; \forall i$, assigning a lower value in higher-density regions.
T-SNE carries out the dimensionality reduction task through the construction of a neighbourhood probability distribution q_{ij} for the latent coordinates $Y = \{\boldsymbol{y}_i\}$ where $\boldsymbol{y}_i \in \mathbb{R}^{p'}$, with $p' < p$

$$q_{ij} = \frac{(1 + \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2)^{-1}}{\sum_{k \neq i}(1 + \|\boldsymbol{y}_i - \boldsymbol{y}_k\|^2)^{-1}}$$

The q_{ij} distribution is chosen to be long-tailed to represent the similarity between closer data while solving the problem of exponentially vanishing contributions for outliers [23]. The latent coordinates are found through the minimisation of the Kullback-Leibler divergence of q_{ij} and p_{ij}

$$D_{KL}(p||q) = \sum_{ij} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right),$$

that measures the dissimilarity between the two distributions. This course of action preserves the distance-related information: close points in the original space are mapped to close-by points in the latent space, even though the actual distance is not preserved and the scale is deformed.

Bibliography

[1] Felipe Almeida and Geraldo Xexéo. "Word Embeddings: A Survey". In: *CoRR* abs/1901.09069 (2019). arXiv: 1901.09069. URL: http://arxiv.org/abs/1901.09069.

[2] Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. "On the Cross-lingual Transferability of Monolingual Representations". In: *CoRR* abs/1910.11856 (2019). arXiv: 1910.11856. URL: http://arxiv.org/abs/1910.11856.

[3] Piotr Bojanowski et al. *Enriching Word Vectors with Subword Information*. 2016. DOI: 10.48550/ARXIV.1607.04606. URL: https://arxiv.org/abs/1607.04606.

[4] Giuseppe Castellucci et al. "Learning to Solve NLP Tasks in an Incremental Number of Languages". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 837–847. DOI: 10.18653/v1/2021.acl-short.106. URL: https://aclanthology.org/2021.acl-short.106.

[5] Matthias Delange et al. "A continual learning survey: Defying forgetting in classification tasks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. DOI: 10.1109/tpami.2021.3057446. URL: https://doi.org/10.1109%2Ftpami.2021.3057446.

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

[7] *GDPR*. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN#d1e40-1-1.

[8] *Google search api*. URL: https://serpapi.com.

[9] Jochen Görtler et al. "Neo: Generalizing Confusion Matrix Visualization to Hierarchical and Multi-Output Labels". In: 2022. URL: https://arxiv.org/abs/2110.12536.

[10] Chuan Guo et al. *On Calibration of Modern Neural Networks*. 2017. DOI: 10.48550/ARXIV.1706.04599. URL: https://arxiv.org/abs/1706.04599.

[11] Kaiming He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. DOI: 10.48550/ARXIV.1502.01852. URL: https://arxiv.org/abs/1502.01852.

[12] Hecht-Nielsen. "Theory of the backpropagation neural network". In: *International 1989 Joint Conference on Neural Networks*. 1989, 593–605 vol.1. DOI: 10.1109/IJCNN.1989.118638.

[13] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. DOI: 10.48550/ARXIV.1503.02531. URL: https://arxiv.org/abs/1503.02531.

[14] *IAB audience taxonomy*. URL: https://iabtechlab.com/standards/audience-taxonomy/.

[15] Armand Joulin et al. “Bag of Tricks for Efficient Text Classification”. In: *CoRR* abs/1607.01759 (2016). arXiv: 1607.01759. URL: http://arxiv.org/abs/1607.01759.

[16] James Kirkpatrick et al. “Overcoming catastrophic forgetting in neural networks”. In: *Proceedings of the National Academy of Sciences* 114.13 (Mar. 2017), pp. 3521–3526. DOI: 10.1073/pnas.1611835114. URL: https://doi.org/10.1073%2Fpnas.1611835114.

[17] Danail Krzhalovski. “Generalized Taxonomic Text Classification”. In: (2021).

[18] Tsung-Yi Lin et al. “Focal Loss for Dense Object Detection”. In: *CoRR* abs/1708.02002 (2017). arXiv: 1708.02002. URL: http://arxiv.org/abs/1708.02002.

[19] Ilya Loshchilov and Frank Hutter. “Fixing Weight Decay Regularization in Adam”. In: *CoRR* abs/1711.05101 (2017). arXiv: 1711.05101. URL: http://arxiv.org/abs/1711.05101.

[20] Laurens van der Maaten and Geoffrey Hinton. “Visualizing Data using t-SNE”. In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[21] Amit Mandelbaum and Adi Shalev. “Word Embeddings and Their Use In Sentence Classification Tasks”. In: *ArXiv* abs/1610.08229 (2016).

[22] Alessandro Marcomini, Giulia Campesan, and Tommaso Faorlin. “t-SNE and DBSCAN for clustering and visualisation of high dimensional datasets”. In: (2021).

[23] Pankaj Mehta et al. “A high-bias, low-variance introduction to Machine Learning for physicists”. In: *Physics Reports* 810 (May 2019), pp. 1–124. DOI: 10.1016/j.physrep.2019.03.001. URL: https://doi.org/10.1016%2Fj.physrep.2019.03.001.

[24] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. “Linguistic Regularities in Continuous Space Word Representations”. In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia: Association for Computational Linguistics, June 2013, pp. 746–751. URL: https://aclanthology.org/N13-1090.

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

[26] Tomás Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *CoRR* abs/1310.4546 (2013). arXiv: 1310.4546. URL: http://arxiv.org/abs/1310.4546.

[27] Natawut Monaikul et al. “Continual Learning for Named Entity Recognition”. In: *AAAI*. 2021.

[28] Azadeh Sadat Mozafari et al. *Attended Temperature Scaling: A Practical Approach for Calibrating Deep Neural Networks*. 2018. DOI: 10.48550/ARXIV.1810.11586. URL: https://arxiv.org/abs/1810.11586.

[29] Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht. “Obtaining Well Calibrated Probabilities Using Bayesian Binning”. In: *Proceedings of the Twenty-*

Ninth AAAI Conference on Artificial Intelligence. AAAI'15. Austin, Texas: AAAI Press, 2015, pp. 2901–2907. ISBN: 0262511290.

[30] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[31] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global Vectors for Word Representation." In: *EMNLP*. Vol. 14. 2014, pp. 1532–1543.

[32] Chuck Rosenberg, Martial Hebert, and Henry Schneiderman. "Semi-Supervised Self-Training of Object Detection Models". In: *2005 Seventh IEEE Workshops on Applications of Computer Vision (WACV/MOTION'05) - Volume 1*. Vol. 1. 2005, pp. 29–36. DOI: 10.1109/ACVMOT.2005.107.

[33] Victor Sanh et al. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. 2019. DOI: 10.48550/ARXIV.1910.01108. URL: https://arxiv.org/abs/1910.01108.

[34] Wilson L Taylor. ""Cloze procedure": A new tool for measuring readability". In: *Journalism quarterly* 30.4 (1953), pp. 415–433.

[35] "Uniqueness of the weights for minimal feedforward nets with a given input-output map". English (US). In: *Neural Networks* 5.4 (1992), pp. 589–593. ISSN: 0893-6080. DOI: 10.1016/S0893-6080(05)80037-1.

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