ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Natural Language Processing

HIERARCHICAL MULTI-LABEL TEXT CLASSIFICATION IN A LOW-RESOURCE SETTING

CANDIDATE
Andrea Lavista

SUPERVISOR
Prof. Paolo Torroni

CO-SUPERVISOR
Dott. Giuseppe Savino

Academic year 2021-2022

Session 3rd

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Text classification encompasses those Natural Language Processing problems in which text is associated with a set of categories: sentiment analysis, topic classification, spam filtering, and emotion detection are some of the most famous examples. Text classification is increasingly used in real-world applications: pre-trained models (like word embeddings and contextual embeddings) have facilitated this process, making these problems addressable with fewer resources and data.

This thesis addresses a hierarchical multi-label text classification problem in a low-resource setting. The dataset used in the experiments consists of 2546 English school exercises associated with labels describing their content. This is a hierarchical classification problem because the set of classes is defined through a three-level hierarchical taxonomy; it is multi-label because more than one label can be assigned to each exercise. The goal of this thesis is to find a suitable solution to this task, including the use of pre-trained models and few-shot learning techniques for NLP.

In chapter 2 we present the theoretical background related to this work, focusing on text representation, multi-label classification, hierarchical classification, and few-shot text classification. Chapter 3 shows

the datasets used in this work, with details on the hierarchical taxonomy. Chapter 4 introduces the experiments made in this project, explaining the architectures employed and showing the metrics used for the evaluation. Chapter 5 contains the results of the experiments with our quantitative and qualitative analysis. Chapter 6 indicates possible directions for future work to improve and extend this work.

Chapter 2

Background

In this chapter we present the theoretical background relevant to this work. In particular, some of the main text representation techniques used in Natural Language Processing are presented. Subsequently, we discuss about types of classification tasks in the field of Machine Learning, focusing on the difference between single-label and multi-label problems and on hierarchical classification approaches and metrics. Eventually, we briefly introduce few-shot learning and describe SetFit, a recently proposed framework for few-shot text classification.

2.1 Text representation

In order to be processed by computers words, sentences, paragraphs, and documents have to be represented in a numerical form: this is done through text representation techniques, which transforms words and texts in multidimensional vectors within a *semantic space*. Many of these techniques are based on the distributional hypothesis for which words occurring in similar contexts tend to have similar meanings.

In this section we present some of the most important text representation techniques, highlighting the advances they have introduced to the field of NLP.

2.1.1 Tf-idf

Tf-idf (term frequency-inverse document frequency) is a weighting algorithm applied to term-document matrixes. It can be used as a feature extraction method in NLP classification problems (as in the experiment described in section 4.3) and to represent documents in Information Retrieval (IR) systems. Tf-idf is a simple but effective model and often is used as a baseline, before shifting to more complex models.

Term-document matrices are constructed upon a set of documents counting the number of occurrences of each term in each document. In other words, the cell in position i,j represents the frequency of term$_i$ in doc$_j$. The drawback of considering only frequencies is that very frequent words (which are often uninformative) have a high weight even though they are not significant for the vector representation. tor the vector representation. Tf-idf overcomes this limitation by weighting less uninformative words, introducing in the weighting function the inverse document frequency (idf). The formula of tf-idf weighting is:

$$w_{t,d} = tf_{t,d} \times idf_t$$

This formula is the product of two terms:

- $tf_{t,d}$: the frequency of the term t in the document d
- idf_t: the inverse document frequency of the term t, defined as:

$$idf_t = log\left(\frac{N}{df_t}\right)$$

where N is the total number of documents and idf_t is the number of documents in the document set that contains the term t.

2.1.2 Word embeddings

Word embeddings are representation methods that map words into short dense vectors (earlier methods such as tf-idf e PPMI represents words with sparse vectors).

Word embeddings have had a significant impact on NLP research: word embeddings (respect to sparse vectors) better preserve semantic relations as synonymy and relational similarity and "work better in every NLP task than sparse vectors"[8].

The first word embedding method was **Word2vec**, proposed in Mikolov et al. (2013) [14]. Unlike tf-idf and PPMI, word2vec is not based on frequencies and does not compute co-occurrence matrices: it consists of training a classifier in a self-supervised manner, using one of these two architectures (shown in figure 2.1):

- Continuous Bag-of-Words (CBOW): the model predicts the current word based on the context
- Skip-gram: the model predicts surrounding words given the current word

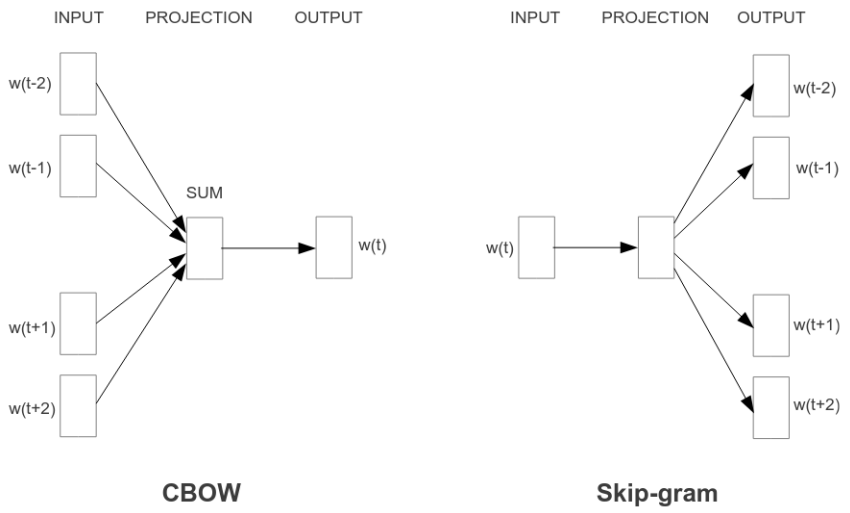

Figure 2.1: CBOW and Skip-gram architectures for word2vec training. [14, Figure 1]

Many other word embedding models have been proposed after word2vec; the most prominent are:

- **GloVe** (Pennington et al., 2014[16]), which "leverages statistical information by training only on the nonzero elements in a word-word co-occurrence matrix, rather than on the entire sparse matrix or on individual context windows in a large corpus"
- **fastText** (Bojanowski et al., 2017) [2] which extends word2vec enriching word vectors with subword information. FastText, operating at the subword level, computes embeddings for n-gram of characters: these can be used to compute embeddings for out-of-vocabulary words
- **ConceptNet Numberbatch** (Speer and Chin., 2016 [26]), that combines distributional word embeddings learned from text with ConceptNet, a knowledge graph that encodes semantic relations between words and common sense knowledge

One of the disadvantages of word embeddings is that they provide static vectors: each word is associated with a vector regardless of the context in which the word occurs. This limitation makes word embeddings unable to deal with polysemy and semantic nuances. This aspect is considered by contextual embeddings, which are models that attempt to interpret words by also considering their context: the vector representation assigned to a certain word changes in relation to its context.

2.1.3 Contextual embeddings

The first contextual embedding model was **ELMo** (Embeddings from Language Models), proposed in Peters et al. (2018) [17], based on bidirectional LSTM; then followed by **GPT** (Radford et al. 2018) [20], based on **Transformer** architecture (Vaswani et al., 2017) [29]. Transformer is

a deep learning model based on attention mechanism that does not use recurrence: this makes the model more parallelizable, thus more efficient than LSTM.

The most popular contextual embedding model is **BERT** (Bidirectional Encoder Representations from Transformers) proposed by Devlin et al. (2019) [5], also based on the Transformer model.

BERT's framework follows these two steps during training (as shown in figure 2.2):

- pre-training: the model is trained with unlabeled data, with self-supervision, solving simultaneously two tasks:
  - MLM (Masked Language Model): 15% of the input tokens are masked and the model has to predict them
  - NSP (Next Sentence Prediction): each sample is composed of two sentences and the models objective is to predict whether the second sentence follows the first one in the training corpus
- fine-tuning: the model is trained on the downstream task with labeled data

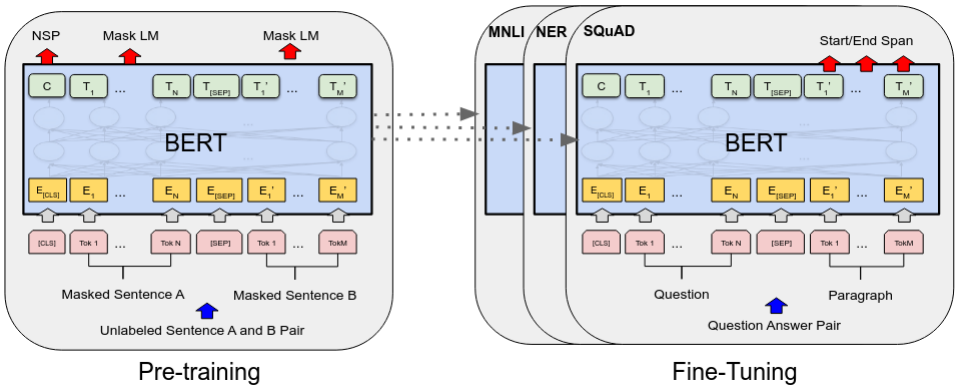

Figure 2.2: Pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architecture is used in both pre-training and fine-tuning. [5, Figure 1]

Anyway, pre-trained models based on BERT can also be used as a feature extraction method or to produce embeddings, without any fine-tuning.

Many variants of BERT have been proposed and are freely available for experiments: mBERT, the multilingual version of BERT, is used in this work for the experiments explained in section 4.5.

2.1.4 Sentence embeddings

Sentence embedding techniques aim to represent sentences by semantic vectors. A simple and strong baseline to compute the embedding of a sentence is the average of the vectors of the words that compose it. Many models have been proposed for this task: the most popular are **Doc2vec** (Le and Mikolov, 2014) [11], **InferSent** (Conneau et al., 2017) [4], **Universal Sentence Encoder** (Cer et al., 2018) [3], **LASER** (Artetxe and Schwenk, 2019) [1] and **Sentence-BERT** (Reimers and Gurevych, 2019) [21]. Below we examine LASER and Sentence-BERT which are those used in this work.

LASER (Language-Agnostic SEntence Representations) is a language-agnostic BiLSTM encoder that transforms sentences into language-independent vectors. So that "semantically similar sentences in different languages are close in the embedding space"[1].

Sentence-BERT (SBERT) indicates a family of sentence embedding models based on BERT architecture, which has been proposed because of the inability of BERT-based model to generate good sentence embeddings. SBERT fine-tune BERT in a siamese / triplet networks, depending on the available data; the training arechitectures for SBERT are:

- classification objective function (the architecture at the left in figure 2.3): this can be used with classification taks datasets, like NLI datasets

- regression objective function: (the architecture at the left in figure 2.3): this is used with sentence similarity datasets
- triplet objective function: given a sentence a, a positive sentence p, and a negative sentence n, triplet loss tunes the network such that the distance between a and p is smaller than the distance between a and n

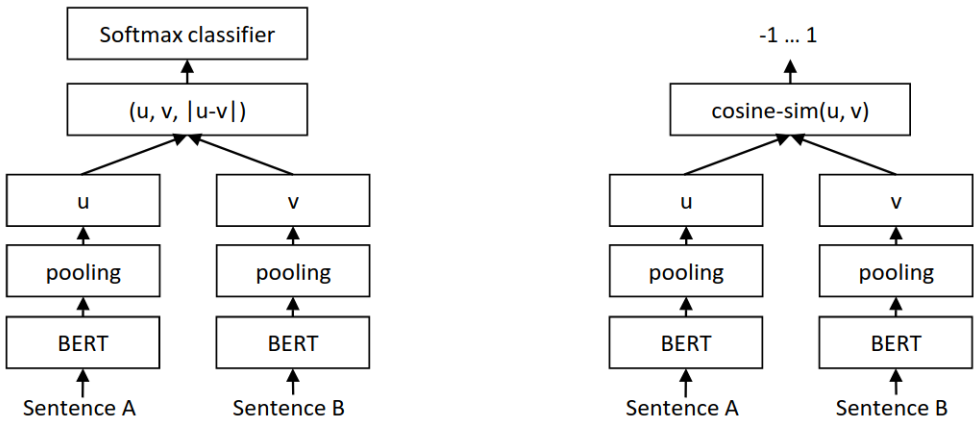

Figure 2.3: SBERT training architectures: on the left, there is the architecture with the classification objective function; on the right the one with the regression objective function. The second one can also be used during inference, to compute similarity scores between sentences. [21, Figures 1, 2]

SBERT models are also employed by SetFit, a framework for few-shot text classification used in this work. More details about how SetFit works are provided in section 2.3.1.

2.2 Classification tasks and metrics

This section delves into classification tasks and metrics. The concepts presented in this section are not related only to text classification but apply to classification tasks in general.

2.2.1 Single-label and multi-label classification

Classification tasks can be divided into two categories:

- **single-label**: each sample is assigned to exactly one label
- **multi-label**: each sample is assigned to a set of labels of any size (even to no label at all)

Multi-label classification methods can be further grouped into two categories (Tsoumakas and Katakis, 2007) [27]:

- **algorithm adaptation methods**: which handle multi-label problems with tweaks in the learning algorithm
- **problem transformation methods**: which transform multi-label problem into one or more single-label classification or regression problems

Many problem transformation methods have been proposed, the most popular are:

- **binary relevance**: transform a multi-label problem with N labels into N binary classification problems. The binary classifiers are learned independently, one for each label and the output is the union of all the binary classifiers' output
- **label powerset**: transform a multi-label problem into a multi-class problem where the labels are all unique label combinations in the training data

In this work binary relevance transformation is used with the Logistic Regression classifier in the experiment explained in section 4.6; all the other classifiers fall into the algorithm adaptation methods category.

2.2.2 Hierarchical classification

Hierarchical classification in Machine Learning is a type of classification problem where the classes are defined through a hierarchical taxonomy. Hierarchical classification can be solved with different approaches as delineated in Silla and Freitas, (2011) [25]:

- **flat classification approach**: learning a single classifier for the classes at the leaf nodes of the taxonomy, ignoring the entire hierarchical structure (see figure 2.4). This approach transforms the hierarchical classification problem into a standard multi-class (or multi-label) classification problem where the labels are the leaf nodes of the hierarchy. The main advantage is its simplicity, but the disadvantage is that the model cannot exploit hierarchical information during training and inference

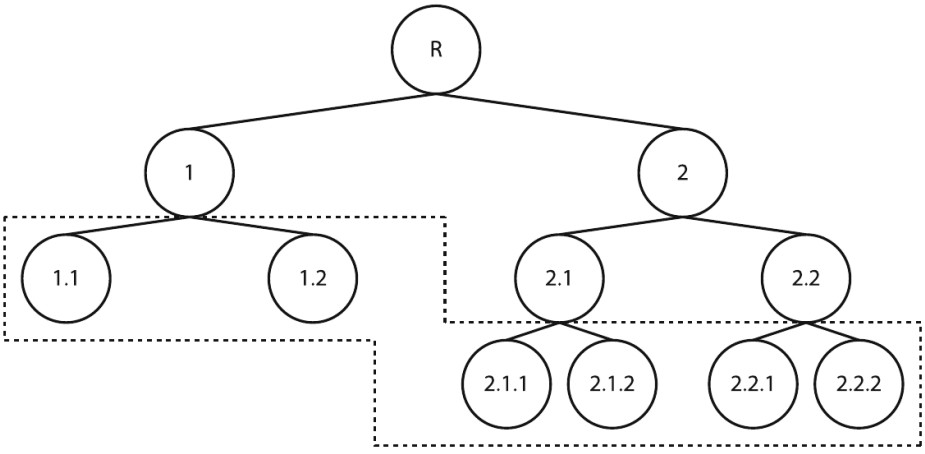

Figure 2.4: Flat classification approach. [25, Figure 3]

- **local classifier approaches**: building a set of local classifiers combined together in the inference phase. The main advantage of local classifier approaches is that they consider the class hierarchy during the creation of the training sets and during inference; on the other hand, the drawbacks are increased complexity and error

propagation during inference. There are three standard ways of implementing local classifiers:

– *local classifier per node* (LCN): training one binary classifier for each node of the class hierarchy, except the root node (figure 2.5)

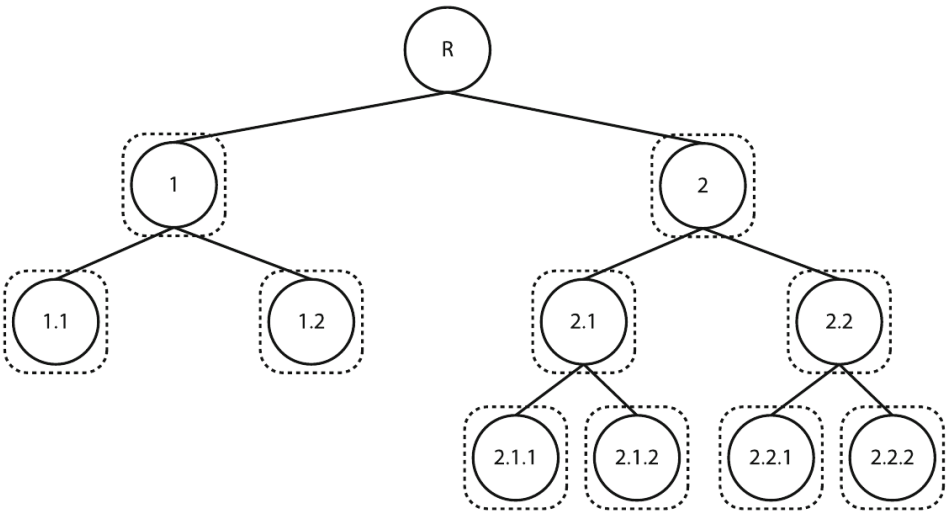

Figure 2.5: "Local classifier per node approach (circles represent classes and dashed squares with round corners represent binary classifiers). "[25, Figure 4]

– *local classifier per parent node* (LCPN): a classifier is trained for each parent node in the class hierarchy, to distinguish between its child nodes (figure 2.6)

– *local classifier per level* (LCL): training a classifier for each level of the class hierarchy (figure 2.7)

HiClass (Miranda et al., 2021 [15]) is an open-source Python library (compatible with scikit-learn) that provides the implementation of these three local classifier approaches; it also contains the implementations of hierarchical metrics (that will be explained later). A limitation of HiClass is that it does not yet support multi-label settings.

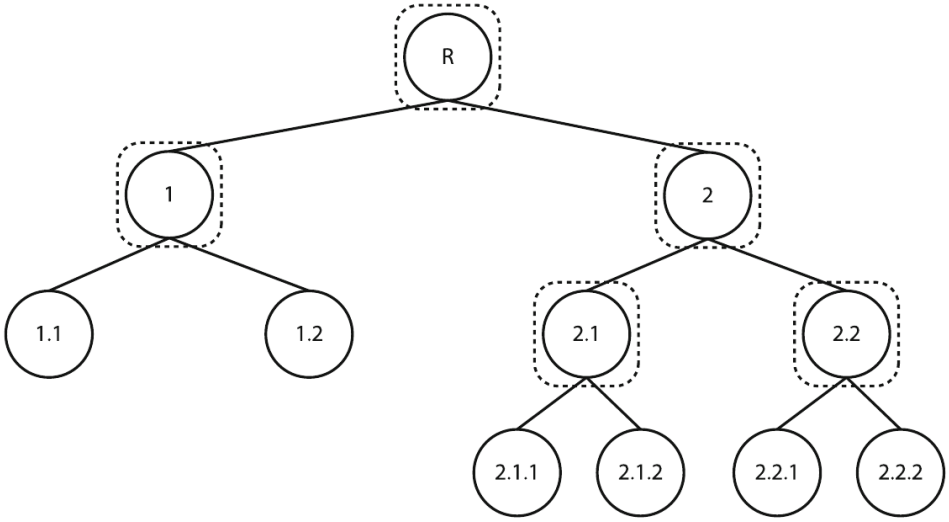

Figure 2.6: "Local classifier per parent node (circles represent classes and dashed squares with round corners in parent nodes represent multi-class classifiers predicting their child classes). "[25, Figure 5]

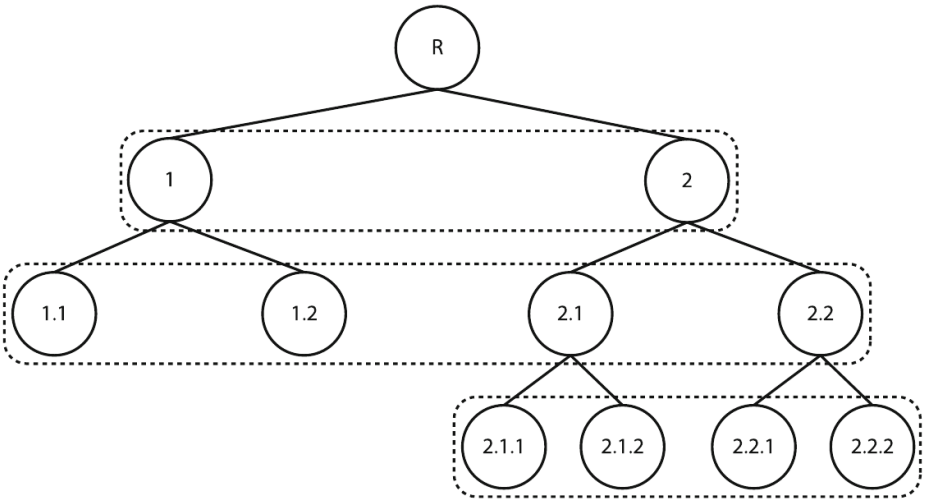

Figure 2.7: "Local classifier per level (circles represent classes and each dashed rectangle with round corners encloses the classes predicted by a multi-class/multi-label classifier)."[25, Figure 6]

- **global classifier (or big-bang) approach**: learning a single model for all the classes of the taxonomy. The advantage over local approaches is that this approach can exploit hierarchical information during training and inference with a single (although complex) model.

Metrics

All the evaluation metrics used for flat classification problems (exact match, precision, recall, etc.) can be used also for hierarchical classification. However, "these measures are not suitable for hierarchical categorization since they do not differentiate among different kinds of misclassification errors."[9].

Many metrics that consider the hierarchical structure have been proposed: in this work, we will use those proposed in Kiritcheno et al. (2005) [9]. These metrics penalize less those misclassifications where the wrong predicted class share ancestors with the correct class, and punishes errors at higher levels of a hierarchy more heavily; they are defined as follow:

- hierarchical precision (hP)

$$hP = \frac{\sum_i |\hat{C}_i \cap \hat{C}'_i|}{\sum_i |\hat{C}'_i|}$$

- hierarchical recall (hR)

$$hR = \frac{\sum_i |\hat{C}_i \cap \hat{C}'_i|}{\sum_i |\hat{C}_i|}$$

- hierarchical F-measure (hF)

$$hF_\beta = \frac{(\beta^2 + 1) \cdot hP \cdot hR}{(\beta^2 \cdot hP + hR)}, \beta \in [0, +\infty)$$

Hierarchical precision and recall extend the classical precision and recall with the following addition: "each example belongs not only to its class but also to all ancestors of the class in a hierarchical graph, except the root"[9]. Indeed, in the previous formulas, $\hat{C}_i$ indicates the set of true classes extended with all their ancestors (except the root); $\hat{C}'_i$ is the same for the predicted classes.

These metrics can also be applied in a multi-label setting; unfortunately, the only library we found that provides an implementation of these metrics is HiClass, but they do not apply to the multi-label scenario. So, we implement them from scratch.

2.3 Few-shot learning

Few-shot learning in Machine Learning refers to learning from just a few examples. Data annotation is costly and time-consuming, so these approaches have received a lot of attention. Below we present SetFit (Tunstall et al., 2022) [28], a framework for few-shot text classification based on Sentence Transformers (Reimers and Gurevych, 2019) [21].

2.3.1 SetFit

SetFit framework is composed of two components: a Sentence Transformer model that derives sentence embeddings and a text classification head (a logistic regression in the original work) that performs the classification. The learning is divided in two steps, as in figure 2.8:

- ST fine-tuning: the sentence transformer model is fine-tuned on the input data in a contrastive, Siamese manner on sentence pairs. In particular, the training set used for this fine-tuning is formed by pairs of examples where the objective is 1 if those two examples belong to the same class, and 0 otherwise.

- classification head training: the fine-tuned ST encodes the original labeled training data; these embeddings, along with their class labels, constitute the training set for the classification head

SetFit, with respect to previous approaches, employs lighter models and is faster at training and inference [28]. Moreover, SetFit has been

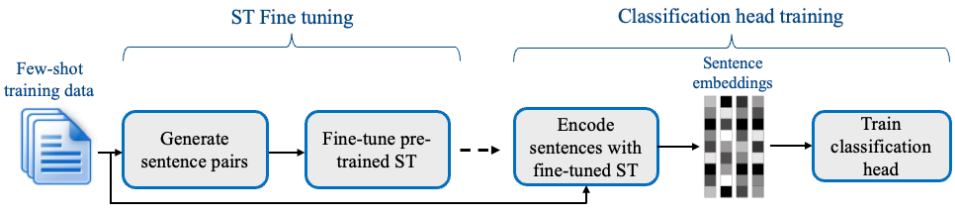

Figure 2.8: SetFit's training steps. [28, Figure 2]

chosen for this work because it can be applied to multi-label classification problems and can be used with multilingual texts (employing a multilingual sentence transformer model).

Chapter 3

Dataset

In this chapter we present the datasets employed in this work: a labeled dataset, which is used in all the experiments (presented in chapter 4) and an unlabeled dataset necessary for the experiment explained in section 4.5.1.

3.1 Labeled dataset

In this section, the main dataset used in this work is presented, providing information about the textual content and about the hierarchical taxonomy used for its classification. Moreover, we also explain the preprocessing steps performed to obtain the dataset that will be used for the experiments and the train-test split procedure.

This dataset is a hierarchical multi-label dataset composed of 2546 school exercises. This means that each exercise is tagged with classes coming from a hierarchical taxonomy and given its multi-label nature, more than one class for each level of the taxonomy can be assigned to each of them.

3.1.1 Text

The dataset contains English language exercises for Italian secondary schools; therefore, the texts of the exercises are multilingual and include two languages: Italian and English. Here are three examples of exercises taken from the dataset:

1. Scegli l'alternativa corretta.

   Jessica: I'm going to the supermarket; do you need _____ (lots / any) groceries, Bill?

   Bill: Yes, could you get me _____ (a / any) bottle of milk, please?

   Jessica: I think we've still got _____ (much / some) in the fridge.

   Bill: No, I just drank it. There isn't _____ (any / some) left.

   Jessica: Ok. I'm going to get _____ (a little / a few) apples, too.

   Bill: Get _____ (many / lots of) apples so we can make a pie!

   Jessica: That's a great idea!

2. Completa con il verbo corretto.

   _____ martial arts

   _____ cricket

   _____ gymnastics

   _____ a horse

   _____ rugby

3. Speak _____, please. slowly / slow / slowed

These texts are quite short, as shown in figure 3.1, more specifically 2499 out of 2546 exercises contain less than 300 words. This aspect allows to apply more easily BERT-based models, which receive a fixed size input (i.e., 512 tokens), as we will see in 4.5.

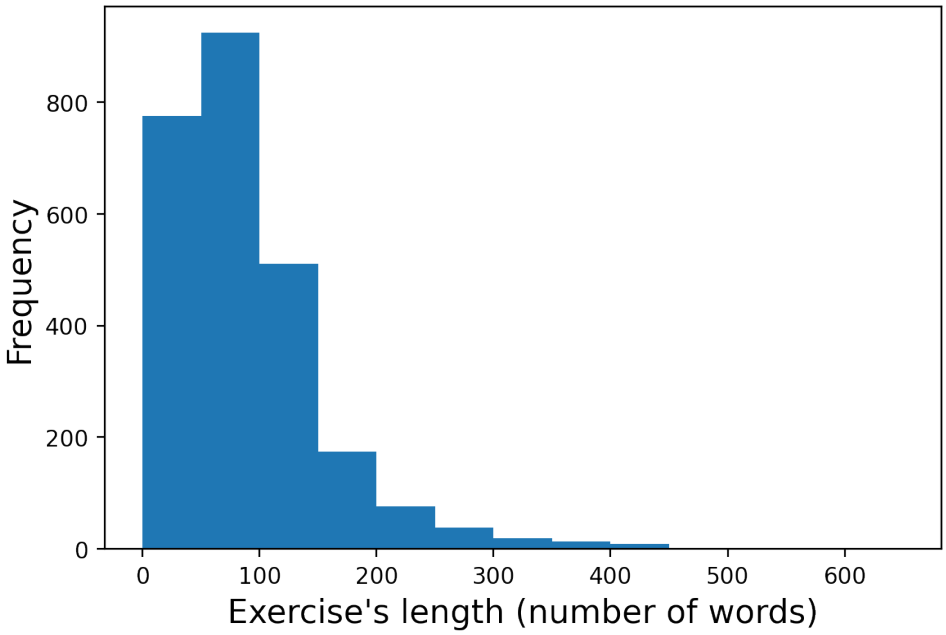

Figure 3.1: Distribution of labeled exercises' lengths, where length is defined as the number of words.

As previously stated, each exercise of the dataset is tagged with at least one label for each level of the taxonomy. The following section provides more details about the pre-processing steps performed on the labels to obtain the dataset and the hierarchical taxonomy used for the experiments.

3.1.2 Content taxonomy

The labels associated with the exercises come from two hierarchical taxonomies: one for *lower secondary education* (*scuola secondaria di primo grado* in the Italian educational system) and the other one for *upper secondary education* (*scuola secondaria di secondo grado* in the Italian educational system). Both these two taxonomies are described by a three-level hierarchy and each label is identified by an ID and comes with a name.

Even if the taxonomies are defined upon three levels, in this work

we choose to ignore the third level and focus only on the first two levels. This choice is made because of the extremely high number of classes on the third level of the taxonomies and the small number of exercises per class. Indeed, on the third level of the taxonomy, about 12% of the classes are associated with no example and only about 14% of the classes are associated with more than 20 examples. Trying to perform classification at the third level with machine learning models would be extremely challenging and goes beyond the purpose of this work. In addition, with so scarce data we cannot produce a large enough test set that we can rely on to test performance and compare different approaches.

Moreover, the two taxonomies mostly overlap, especially in the first two levels, which means that there are many classes for which there is a class in the other taxonomy that refers to the same topic. Given that the purpose is to classify the content of the exercises we decide to generate a new taxonomy produced from their unification and use this one for tagging the exercises (see figure 3.2). From now on, classes on the first level of taxonomy will be also referred to as *macro-topics* while those on the second level as *topics*.

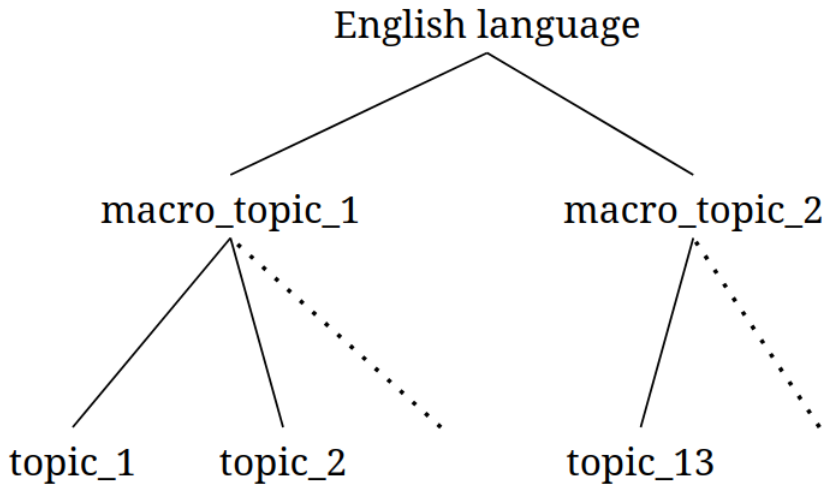

Figure 3.2: Structure of the hierarchical taxonomy used for the classification of the English language exercises.

The classes in the two taxonomies that refer to the same topic have

different IDs and sometimes also slightly different names (due to capital letters, commas, dashes, etc.). So, the unification of the taxonomies is performed through a semi-automatic procedure, in particular:

1. a python script tries to detect these pairs of classes, matching those whose edit-distance computed upon their names is below a certain threshold

2. then a manual check is performed, updating the script to:

   (a) choose the better edit-distance threshold

   (b) remove the false positive matching pairs detected by the script

   (c) add potential pairs of labels not detected by the procedure

The script compares only the names of the classes that belong to the same level of the taxonomy and the search is performed iteratively, starting from an edit-distance threshold equal to zero (i.e., searching for labels with the same name) and then the threshold is increased at the end of each iteration. Moreover, the edit-distance is computed upon the lowercase version of the classes' names using polyleven library [18], which provides a more efficient implementation, compared to other libraries like NLTK [12].

After these steps, the new taxonomy is a two-level hierarchy that has 4 classes at the first level and 33 classes at the second level. Anyway, in the experiments presented in this work all the classes with less than 65 examples are excluded, in order to have enough examples per class for training and testing: with this choice, all the classes have at least 10 examples in the test set. Therefore, the dataset employed has 2 classes on the first level and 16 classes on the second level of the hierarchy. From now on, we refer to this version of the dataset.

In figure 3.3 there are a few examples taken from the dataset at the end of all these steps.

Exercises' text	Macro topic tags	Topic tags
Have you _____ sushi? never ate / ever eated / ever eaten	macro_topic_1	topic_1
Identifica l'alternativa corretta. He always has breakfast after _____ (have / having) a shower. _____ (Besides / Instead of) reading very well, that three-year-old child can even write!	macro_topic_1	topic_2
Write ten lines about a film you have recently enjoyed and explain why.	macro_topic_1 macro_topic_2	topic_1 topic_3 topic_13 topic_14

Figure 3.3: Few examples of English exercises from the labeled dataset.

3.1.3 Labels distribution

This section contains statistics about the distributions of the labels in the dataset.

As said in section 3.1.2, there are two macro-topics: 1783 exercises are labeled with the first one and 922 with the other one. The histogram represented in figure 3.4 instead, shows the number of exercises associated with each topic (i.e., the classes at the second level of the hierarchy). As we can see the dataset has a moderate *imbalance*.

The histogram in figure 3.5, instead, deepens the multi-label nature of the dataset and shows the number of exercises per number of labels (considering only the second level of the taxonomy). It exhibits that many exercises (1720 out of 2546) have only one label, 357 exercises have no label assigned to them, and 469 exercises have at least two labels.

3.1.4 Training and test sets

A portion of the dataset is used as the test set to evaluate the results and compare different approaches. Hence, the dataset is split into training

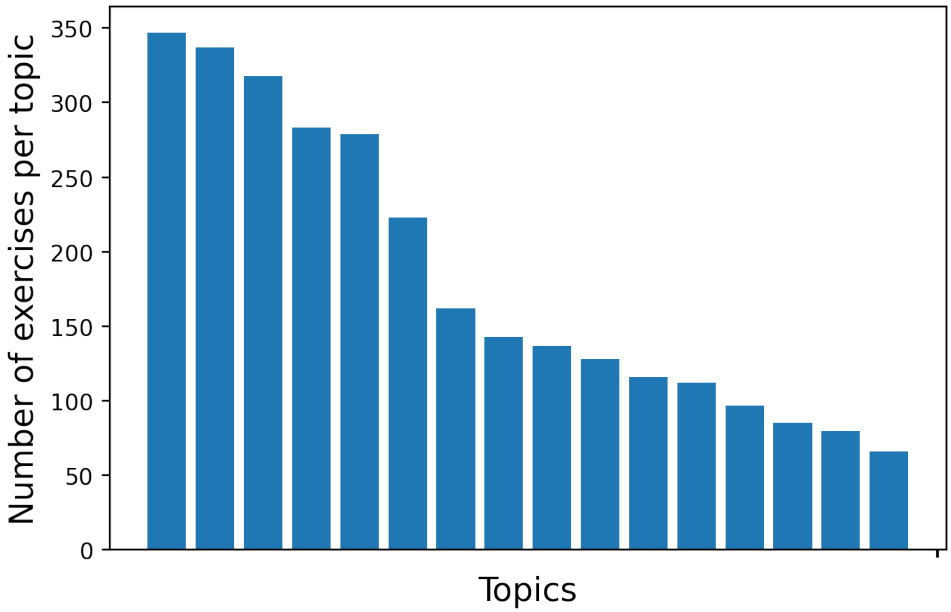

Figure 3.4: Number of exercises per topic.

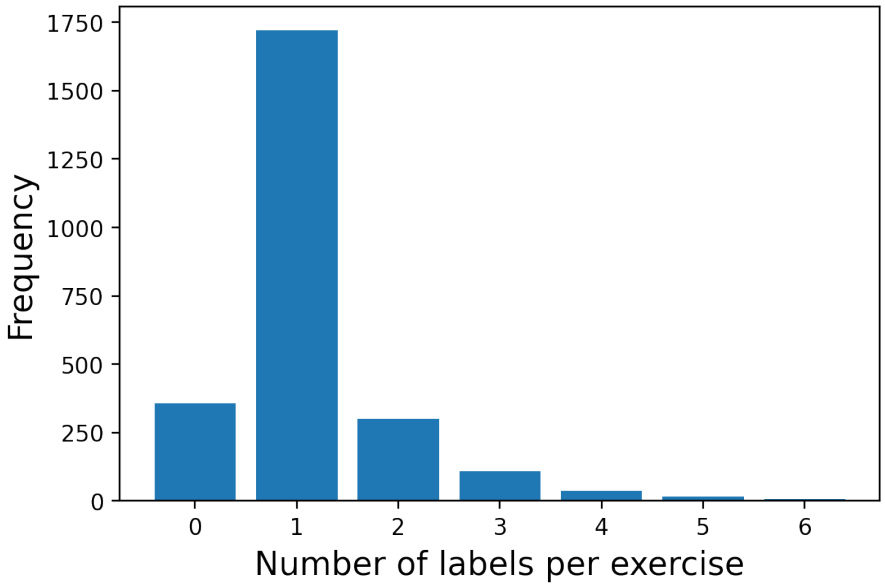

Figure 3.5: Number of exercises per number of associated labels.

and test sets using scikit-learn [22] in a *stratified* fashion, so that the distribution of the classes in the training and test sets are the same. In order to use stratification, the function provided by scikit-learn requires that there are no examples whose combination of labels appears only one time in the dataset. For this reason, these examples are removed before

executing the splitting procedure.

Then 20% of the remaining part of the dataset is kept as the test set while the remaining 80% and those examples removed before the splitting procedure compose the training set. The result of this step consists in having a training set of 2063 exercises and a training set of 483 exercises. All the experiments presented in the next chapter are evaluated using the same test set to ensure a fair comparison between all the approaches employed.

3.2 Unlabeled dataset

The unlabeled dataset is composed of 16751 English language exercises for Italian secondary schools, and it is only used in the experiment that employs task-adaptive pre-training (see section 4.5.1).

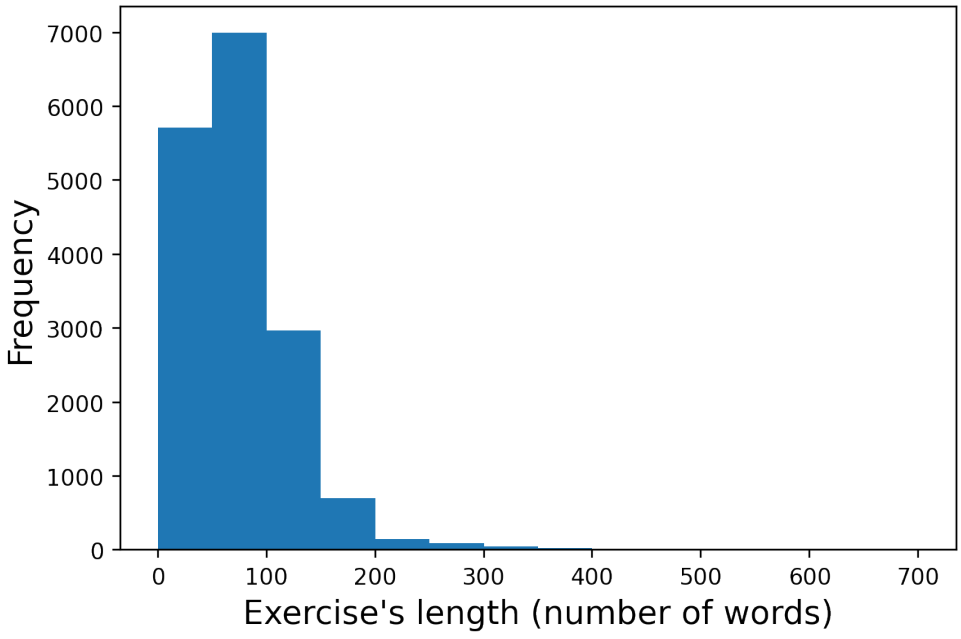

Figure 3.6: Distribution of unlabeled exercises' lengths, where length is defined as the number of words.

The texts of the unlabeled dataset have the same characteristic as the texts of the labeled dataset but they do not overlap. Figure 3.6 shows

the lengths of these exercises in terms of number of words: as we can see the histogram is analogous to the histogram of the labeled exercises.

Chapter 4

Experiments

This chapter present the experiments made to develop a text classification model that automatically classifies English language exercises for secondary school with the taxonomy presented in section 3.1.2.

All the experiments shown in this chapter apply machine learning and deep learning models trained and tested with the dataset introduced in chapter 3. All the approaches implement flat classifiers trained on the second level of the taxonomy (the targets are the topics) and they are evaluated using the same test set in order to provide a fair comparison.

If not stated otherwise the best hyperparameters have been found using a validation set, obtained by splitting the training set with stratification.

4.1 Overview

The approaches implemented in this project are:

- tf-idf features with CatBoost classifier ($\text{CatBoost}_{\text{tfidf}}$)
- LASER embeddings with k-NN classifier ($\text{k-NN}_{\text{LASER}}$)
- Multilingual BERT (mBERT)

- Multilingual BERT with task-adaptive pre-training ($\text{mBERT}_{\text{TAPT}}$)
- SetFit

The first approach employs tf-idf as feature extraction method and then these features form the input used to train a model with CatBoost [19], a gradient boosting algorithm based on decision trees.

Then, given the small size of the dataset, we decide to try also solutions based on *pre-trained models*. After some research, we decided to conduct experiments with LASER [1] and multilingual BERT [5].

LASER is combined with a k-NN classifier: LASER is used to produce the embeddings of the exercises texts, which are the input used to train the k-NN classifier. On the other end, mBERT is fine-tuned just by adding a feed-forward neural network and a sigmoid layer as activation function.

Then, we do another experiment with mBERT where we try to improve the performance by leveraging the *unlabeled data*: before the fine-tuning, we do an additional pre-training with the unlabeled data, on the MLM (Masked Language Modeling) task. This step should adapt mBERT to the domain and the structure of the school exercises' text.

The last approach employs SetFit, a framework thought for providing few-shot learning for text classification problems. This choice is made to investigate if this could be a better approach for classifying classes with few examples, which are the topics with less than 65 exercises that are not included in this dataset and classes at the third level of the taxonomy (see section 3.1.2 for more detail).

The training has been done using Google Colab's GPU, except for CatBoost classifier which does not support the training with GPU for multi-label problems. It was instead trained on a CPU Intel Core i5-1135G7.

4.2 Metrics

The metrics used to evaluate the models in this work are:

- classical text classification metrics:
  - exact match
  - F1 score
  - precision
  - recall
- hierarchical text classification metrics, proposed in Kiritchenko et al. (2006) [10] and discussed in section 2.2.2:
  - hierarchical precision
  - hierarchical recall
  - hierarchical F1 score
- custom metrics relevant for the specific task:
  - partial match
  - partial mismatch

Partial match is defined as the fraction of samples with at least one label predicted by the model and precision equal to 1. In other terms, it is the fraction of exercises classified with at least a correct label and without incorrect labels.

Partial mismatch instead is defined as the fraction of samples with at least one label predicted by the model and precision less than 1. In other terms, it is the fraction of exercises classified with at least an incorrect label.

Partial match and partial mismatch can better tell, with respect to exact match, how many exercises are correctly labeled and how many exercises are wrongly labeled. Ideally, partial match should be maximized, and partial mismatch should be minimized.

F1 score, precision, and recall scores unless noted otherwise are computed using the *macro average*, so as the arithmetic average of the classes' scores. This kind of average is chosen because of the imbalance of the dataset: indeed, macro average weights more the less populated classes with respect to micro and weighted averages. F1 score is also computed using samples average because of the multi-label nature of the dataset.

In the following sections, we present in more detail the approaches implemented in this work and their evaluations, which are made by testing the models on the test set presented in section 3.1.4.

4.3 CatBoost classifier with tf-idf features

This experiment relies on a pipeline of three steps (see figure 4.1):

- text pre-processing
- tf-idf which generates sparse vectors representing the exercises
- CatBoost classifier which predicts the output labels

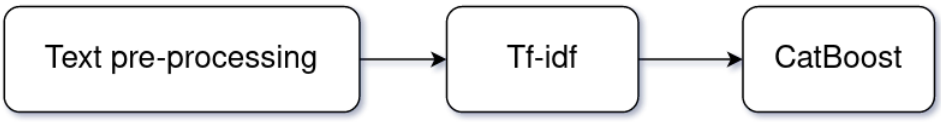

Figure 4.1: Pipeline employed with CatBoost.

The best pre-processing configuration and the best hyperparameters for tf-idf and CatBoost have been found using *cross-validation* on the training set. The pre-processing consists in normalizing the text (lowering all the capital letters) and removing the Italian stop words. English

stop words are not removed because this would worsen the results: this is because many classes about grammar are correctly identified by the model leveraging the presence of some stop words. Also, lemmatization would be harmful because it would normalize inflected forms (e.g., conjugated forms, plural forms, comparative and superlative forms, etc.) removing relevant aspects for discriminating the output classes.

The implementation of tf-idf used in this work is the one provided by scikit-learn. The best configuration extracts unigram features at word level, discarding all the words that appear less than 40 times. The total number of features extracted is 411.

The main advantages of this approach are that this method is explainable and does not require high computational resources.

4.4 k-NN with LASER embeddings

LASER is a pre-trained multilingual model that transforms sentences into *language-independent vectors*. In this approach LASER generates the embeddings of the exercises, which are are then used as input for the k-NN classifier (with k=1). In this case, there are no pre-processing steps.

Figure 4.2: Pipeline employed with LASER.

This approach also does not require high computational resources: only the k-NN classifier has to be trained and this is computationally inexpensive.

4.5 Multilingual BERT

Multilingual BERT is fine-tuned by adding a linear layer on top of the pooled output, using the Hugging Face library [7].

The text is tokenized in order to be passed to mBERT, and the exercises which exceed 512 tokens (which is the maximum input size for mBERT) are truncated. This choice was made because only 0.86% of the exercises consist of more than 512 tokens, so it would not be harmful for the classification task.

4.5.1 Multilingual BERT with TAPT

Given the limited amount of labeled data and the availability of unlabeled data, we try to leverage the unlabeled dataset by performing **task-adaptive pre-training** (TAPT) as in Gururangan et al. (2020) [6]: before fine-tuning mBERT with the labeled dataset we perform a pre-training with the unlabeled data (composed of 16751 exercises) on the MLM task, the same of BERT pre-training.

4.6 SetFit

SetFit has been applied using *paraphrase-multilingual-MiniLM-L12-v2* as sentence embedding, a multilingual transformer-based model provided by the library sentence-transformer [23], and a logistic regression as a classifier.

4.6.1 SetFit vs mBERT: comparison with less training data

Then, given the ability of SetFit to perform few-shot text classification we also compare mBERT and SetFit with less training data to:

- understand the limitations of these models
- check whether SetFit might be more appropriate to perform classification including the less populated classes that has been discarded from the dataset (see section 3.1.2 for more details).

In order to make this comparison we train both mBERT and SetFit using 25%, 50% and 100% of the training set and then evaluate them on the test set.

Chapter 5

Results

Among all the approaches mBERT with TAPT reaches the best performance (see table 5.1). Instead, $\text{CatBoost}_{\text{tf-idf}}$ and the $\text{k-NN}_{\text{LASER}}$ approaches seem inadequate for this task, because:

- CatBoost has a too low recall and struggle with more complex exercises or classes
- LASER is not able to capture semantic aspects to discriminate the exercises

Model	**F1**	**F1 (samples)**	**hF1**	**Precision**	**PM**
$\text{CatBoost}_{\text{tf-idf}}$	0.511	0.403	0.626	0.839	0.414
$\text{k-NN}_{\text{LASER}}$	0.568	0.525	0.693	0.551	0.491
mBERT	0.748	0.647	0.823	0.840	0.648
$\text{mBERT}_{\text{TAPT}}$	**0.766**	0.662	**0.833**	**0.870**	**0.669**
SetFit	0.760	**0.663**	0.822	0.782	0.615

Table 5.1: Models' comparisons with the following metrics: F1 score (average = macro), F1 score (average = samples), hierarchical F1 score, precision (average = macro), and partial match

The approaches based on SetFit and mBERT give satisfactory results, but their main disadvantage is that these models are expensive to

train and require adequate computational resources. Table 5.2 shows the training times of the models.

Model	Training time
$\text{CatBoost}_{\text{tf-idf}}$	5 m
$\text{k-NN}_{\text{LASER}}$	$\approx$ 0 sec
mBERT	1hr 10min
$\text{mBERT}_{\text{TAPT}}$	8 hr (TAPT) + 1 hr 10 min (fine-tuning)
SetFit	50 min

Table 5.2: Models' training times

In the following sections we present the quantitative and qualitative of the models in more detail.

5.1 CatBoost with tf-idf features

This approach reaches good precision, like mBERT, but it has low recall (table 5.3). This is due to the nature of the features: with tf-idf each feature is a word so the model can predict the correct class only if there are words in the input exercise's text from which to infer the correct labels. But this is not always the case, for example with gap-fill exercises where the correct label must be inferred from what is missing. So, the limitations of this approach are due to its lack of a deep understanding of semantics.

As we said CatBoost constructs an explainable classifier: this allows us to inspect which features are most important. In our classifier the most important features are function words, i.e., words that indicate grammatical relationships like auxiliary verbs, pronouns, conjunctions etc. This is because these features are important to correctly classify labels about grammar and justify the choice of not removing the English

Metric	Score
Exact match	0.489
F1 score (macro)	0.511
F1 score (samples)	0.403
Precision (macro)	0.839
Recall (macro)	0.393
Hiearchical F1 score	0.626
Hiearchical precision	0.902
Hiearchical recall	0.479
Partial match	0.414
Partial mismatch	0.073

Table 5.3: CatBoost classifier's quantitative evaluation (with tf-idf features).

stopwords during pre-processing. The 20 most important features for this classifier are: is, the, if, can, to, there, have, going, was, are, like, frasi, past, on, will, you, and, by, how yesterday.

5.2 k-NN with LASER embeddings

k-NN classifier with LASER embeddings has a much higher recall than CatBoost but also a much lower precision (see table 5.4).

Then, to further investigate LASER's ability to capture semantic aspects relevant to this classification task, we plotted a 2D visualization obtained by applying **t-SNE** to the 50 principal components. Figure 5.1 shows this visualization with the single-label exercises, where each color indicates the topic. As we can see, LASER is not able to cluster the exercises with respect to the topics: this is probably due to the textual differences (about domain and structure) between the texts of this

Metric	Score
Exact match	0.545
F1 score (macro)	0.568
F1 score (samples)	0.524
Precision (macro)	0.551
Recall (macro)	0.603
Hiearchical F1 score	0.693
Hiearchical precision	0.662
Hiearchical recall	0.728
Partial match	0.491
Partial mismatch	0.364

Table 5.4: k-NN classifiers quantitative evaluation (with LASER embeddings as input).

dataset and those used for the training of LASER. This visualization also explains why the best value for the number of neighbors of the k-NN classifier was 1. In conclusion, LASER is not adequate for this dataset.

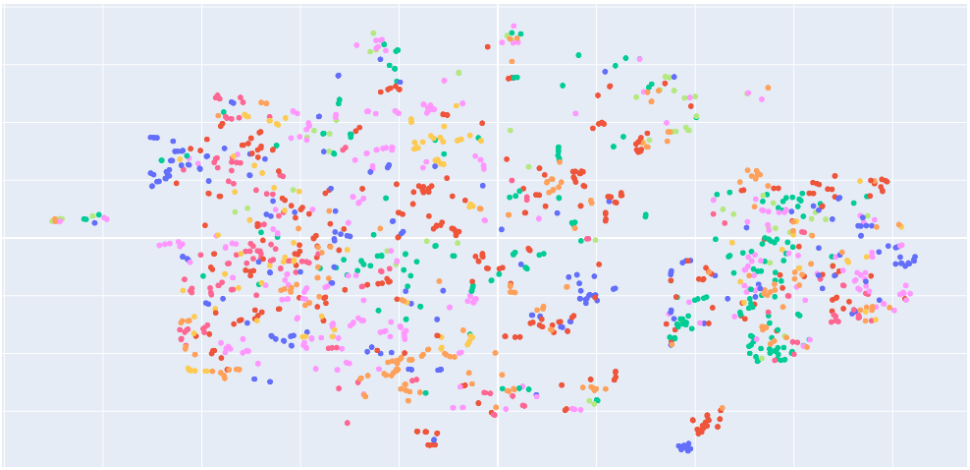

Figure 5.1: t-SNE visualization of single-label exercises' embeddings computed with LASER.

5.3 Multilingual BERT

Multilingual BERT performs well on this task: it outperforms the previous two approaches reaching a high precision (close to CatBoost) and a higher recall. Also, the approach with task-adaptive pre-training shows a slight improvement with respect to training mBERT with only fine-tuning (table 5.5), at the cost of more expensive training.

Metric	**fine-tuning**	**TAPT + fine-tuning**
Exact match	0.706	0.716
F1 score (macro)	0.748	0.766
F1 score (samples)	0.647	0.662
Precision (macro)	0.840	0.870
Recall (macro)	0.693	0.709
Hiearchical F1 score	0.823	0.833
Hiearchical precision	0.879	0.889
Hiearchical recall	0.775	0.784
Partial match	0.648	0.669
Partial mismatch	0.122	0.101

Table 5.5: mBERT's quantitative evaluation, with and without TAPT.

One of the disadvantages of neural networks, and in particular of BERT-based models, is the lack of explainability. Several methods have been proposed to explain predictions of neural networks and therefore enhance their interpretability.

So, to enhance the explainability of mBERT, we inspect some predictions on exercises of the test set with **SHAP values**, proposed in Lundberg and Lee (2017) [13]. SHAP (SHapley Additive exPlanations) is a game theoretic approach to explain the output of machine learning models. This method is applied in this work to:

- validate the model, checking if it is able to make the correct predictions leveraging textual elements relevant for the prediction
- inspect wrong predictions

Figures 5.2, 5.3, 5.4 show explanations based on SHAP values provided by the SHAP library [24]. This library is compatible with machine learning models applied to textual data, providing visualizations that enlighten in red the textual elements that contribute positively to a certain prediction and in blue those that contribute negatively. The intensity of the color is proportional to the contribution.

More specifically, figure 5.2 shows an exercise correctly classified as relating to past tenses: the textual element that contributes most to this prediction is the verb *didn't*, which is indeed a verb conjugated in the past simple. Also figure 5.3 shows a correct prediction: this exercise is about modal verbs and most of the words leveraged by the model to predict the correct label are modal verbs.

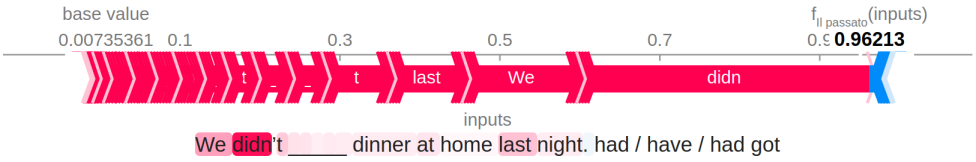

Figure 5.2: Shap values of a correct prediction of an exercise about past tenses.

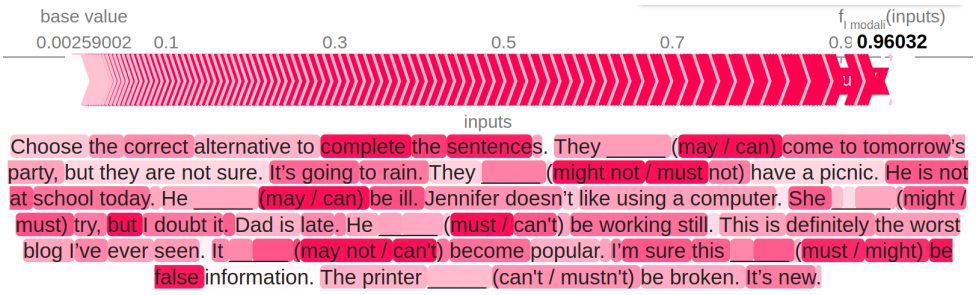

Figure 5.3: Shap values of a correct prediction of an exercise about modal verbs.

Instead, figure 5.4 shows an example misclassified by the model. This exercise is about present tenses but is classified with a class associated with future tenses. Errors like this one could probably occur because present continuous can be used both as present tense and future tenses, so these two tenses share some common grammatical and lexical patterns.

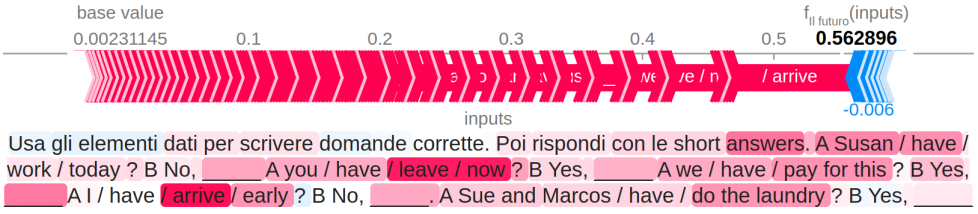

Figure 5.4: Shap values of a wrong prediction of an exercise about present tenses but classified by the model as one about future tenses.

In general, the predictions of the model are precise: when the prediction of the model and the human annotation do not agree, often the model assigns labels that seem correct; this leads to the assumption that the quality of the dataset can be improved by exploiting this model.

mBERT's main weakness is recall, which for some exercises means not being able to capture all associated labels while for others means not being able to assign any labels at all.

5.4 SetFit

SetFit reaches good results, close to mBERT. The main difference is that SetFit has a lower precision and a higher recall, and this explains the worse scores of the metrics partial match and partial mismatch for SetFit (see table 5.6).

In many cases when the predictions disagree with the annotations, the model assigns labels that seem correct, because relevant to the content of the exercises. Instead, the error analysis shows that the SetFit sometimes is misled by the presence of certain words (e.g., the model misclassifies

Metric	**Score**
Exact match	0.677
F1 score (macro)	0.760
F1 score (samples)	0.663
Precision (macro)	0.782
Recall (macro)	0.749
Hiearchical F1 score	0.822
Hiearchical precision	0.826
Hiearchical recall	0.818
Partial match	0.615
Partial mismatch	0.205

Table 5.6: SetFit's quantitative evaluation.

an exercise as one about modal verbs because of the presence of a modal verb in the exercise's text). Also, the model struggles with some exercises written in Italian language, like this one:

> Translate the sentences into English.
> In tutto il paese ci sono sia monumenti antichi sia monasteri. Se non uscirai da casa ora, perderai il treno. Se deciderò di rimanere a Dublino, probabilmente cambierò il mio stile di vita. Quando mi trasferirò a Melbourne non sarà per sempre, solo per tre o quattro anni. I turisti che visitarono Dublino, trovarono gli abitanti della città molto amichevoli.

This exercise is about conditional sentences, but the model does not assign any label to this exercise. So, we think that to better handle these kinds of exercises, we could try to employ machine translation either as a data augmentation method or as a pre-processing step.

5.4.1 SetFit vs mBERT: comparison with less training data

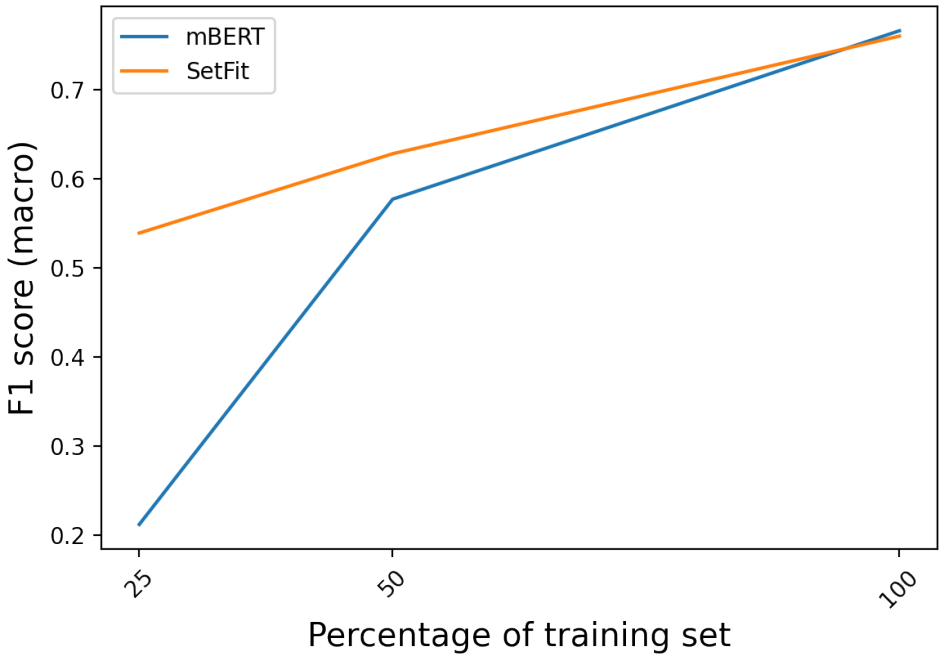

Figure 5.5: Comparison on the F1 score (average = macro) between SetFit and mBERT on 25%, 50% and 100% of the training data.

Model	F1	F1 (samples)	hF1	Precision	PM
$mBERT_{25\%}$	0.212	0.247	0.445	0.339	0.259
$SetFit_{25\%}$	**0.539**	**0.512**	**0.693**	**0.592**	**0.470**
$mBERT_{50\%}$	0.577	0.506	0.702	**0.792**	0.516
$SetFit_{50\%}$	**0.628**	**0.571**	**0.744**	0.691	**0.545**
$mBERT_{100\%}$	0.748	0.647	**0.823**	**0.840**	**0.648**
$SetFit_{100\%}$	**0.760**	**0.663**	0.822	0.782	0.615

Table 5.7: Comparison of SetFit and mBERT on 25%, 50% and 100% of the training data with the following metrics: F1 score (average = macro), F1 score (average = samples), hierarchical F1 score, precision (average = macro) and partial match

As shown in figure 5.5 and in table 5.7, SetFit outperforms mBERT

with less training data. Especially with only 25% of the training set where mBERT struggles to classify many of the classes: 9 out of 16 classes have an F1 score equal to 0; SetFit otherwise has an F1 score equal to 0 for only 1 class.

This comparison indicates that SetFit is a better option, with respect to mBERT, to extend the classification with less populated classes. Also, the results obtained with SetFit are remarkable, considering that with 25% of the training set 8 out of 16 classes have less than 30 examples.

Chapter 6

Future Work

Many directions could be taken for future work:

- extend the classification to all the classes belonging to the second level of the taxonomy: in this work, some classes are discarded because they have too few exercises associated; in section 5.4.1 we show that SetFit (respect to mBERT) has good performances on this task also with a small number of examples, so SetFit is the best candidate for tackling the classification considering all the classes at the second level of the taxonomy
- perform **data augmentation** to have more examples to train the model, especially for less populated classes
- improve the quality of the dataset: given the good performances of mBERT and SetFit, the labels predicted by the model could be added to the dataset. The best option would be to add the labels approved by one or more annotators
- tackle the classification of the third level of the taxonomy: in this case SetFit would probably not be sufficient and it is also necessary to investigate methods for **zero-shot text classification**. An approach that we want to test is to formulate text classification as

a textual entailment problem as in Yin et al. (2019) [30]

- exploit the hierarchical structure of the taxonomy to perform classification: all the approaches in this work employ flat classifiers, but considering the hierarchical information during the training phase could be helpful, especially in the case of extending the classification to the third level of the taxonomy

- test and extend this classifier to other languages: mBERT and our implementation of SetFit are based on multilingual models, so they should be able, to a certain extent, to handle texts in other languages. A quantitative test should be held to evaluate the performances of the models in other languages. In addition, to have a model that can classify languages school exercises in other languages, we could add to the training set:

  - school exercises in other languages
  - synthetic exercises generated by translating the English exercises with machine translation tools

Chapter 7

Conclusions

In this thesis we address a hierarchical multi-label text classification problem that consists in classifying English school exercises according to their content.

At first, we explore the dataset and decide to simplify the task by:

- ignoring the third level of the hierarchical taxonomies and focusing only on the first two levels; this is done because at the third level there are too few examples per class
- unifying the two taxonomies (the lower secondary education taxonomy with the upper secondary education one) because they largely overlap
- ignoring those classes at the second level which have less than 65 examples

Thus, we explore different solutions: we start with Catboost, a gradient boosting algorithm based on decision tree, using tf-idf features; then we shift to approaches based on pre-trained models like mBERT and LASER embeddings. Eventually, we train a classifier with SetFit, a few-shot text classification framework that employs pre-trained sentence embeddings based on Transformer models.

We evaluated all the approaches with text classification metrics (including hierarchical metrics) and two custom metrics.

We argue that the use of pre-trained models is beneficial for this task. However, this is true only with mBERT and SetFit and not for LASER: we think the reason is that with mBERT and SetFit the weights of pre-trained models are updated during the fine-tuning, adapting the model to our task. LASER, on the other hand, cannot be fine-tuned and is unable to cluster exercises according to their content.

Then, we compare SetFit and mBERT with less training data and show that SetFit performs better than mBERT when training with 25% of the training set.

In conclusion we think SetFit would be the most adequate approach for solving this task, considering the labels of the first two levels of the taxonomy. Instead, to address the third level of the taxonomy further experiments should be carried out exploring zero-shot text classification, data augmentation and methods to exploit the hierarchical structure of the taxonomy.

Bibliography

[1] M. Artetxe and H. Schwenk. Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. *Transactions of the Association for Computational Linguistics*, 7:597–610, 2019.

[2] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.

[3] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

[4] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*, 2017.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[6] S. Gururangan, A. Marasovi, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith. Don't stop pretraining: adapt language models to domains and tasks. *arXiv preprint arXiv:2004.10964*, 2020.

[7] Hugging Face. URL: `https://huggingface.co/`.

[8] D. Jurafsky and J. H. Martin. *Speech and language processing*. 2020, page 112. Third Edition draft.

[9] S. Kiritchenko, S. Matwin, A. F. Famili, et al. Functional annotation of genes using hierarchical text categorization. In *Proc. of the ACL Workshop on Linking Biological Literature, Ontologies and Databases: Mining Biological Semantics*, 2005.

[10] S. Kiritchenko, S. Matwin, R. Nock, and A. F. Famili. Learning and evaluation in the presence of class hierarchies: application to text categorization. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 395–406. Springer, 2006.

[11] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[12] E. Loper and S. Bird. Nltk: the natural language toolkit. 2002.

[13] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. 30, 2017. I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors. URL: `https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf`.

[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[15] F. M. Miranda, N. Köehnecke, and B. Y. Renard. Hiclass: a python library for local hierarchical classification compatible with scikit-learn. *arXiv preprint arXiv:2112.06560*, 2021.

[16] J. Pennington, R. Socher, and C. D. Manning. Glove: global vectors for word representation. In *Proceedings of the 2014 conference*

on empirical methods in natural language processing (EMNLP), pages 1532–1543, 2014.

[17] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics, 2018.

[18] polyleven. URL: `https://github.com/fujimotos/polyleven`.

[19] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. Catboost: unbiased boosting with categorical features. 2018.

[20] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. Improving language understanding by generative pre-training, 2018.

[21] N. Reimers and I. Gurevych. Sentence-bert: sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[22] scikit-learn. URL: `https://scikit-learn.org/`.

[23] sentence-transformers. URL: `https://www.sbert.net/`.

[24] SHAP. URL: `https://shap.readthedocs.io/`.

[25] C. N. Silla and A. A. Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1):31–72, 2011.

[26] R. Speer, J. Chin, and C. Havasi. Conceptnet 5.5: an open multilingual graph of general knowledge, 2017.

[27] G. Tsoumakas and I. Katakis. Multi-label classification: an overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.

[28] L. Tunstall, N. Reimers, U. E. S. Jo, L. Bates, D. Korat, M. Wasserblat, and O. Pereg. Efficient few-shot learning without prompts. *arXiv preprint arXiv:2209.11055*, 2022.

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

[30] W. Yin, J. Hay, and D. Roth. Benchmarking zero-shot text classification: datasets, evaluation and entailment approach. *arXiv preprint arXiv:1909.00161*, 2019.

Acknowledgements

I would like to thank my thesis supervisor, Prof. Torroni, for guiding me in the choice of my thesis work and supporting me during its development.

I am also grateful to Giuseppe for his daily support during the execution of this work.