# DATING ITALIAN DOCUMENTS USING BERT

NLP COURSE PROJECT

**Gabriele Graffieti**
Ph.D. student in Data Science and Computation
Department of Computer Science and Engineering (DISI)
University of Bologna
`gabriele.graffieti@unibo.it`

February 16, 2021

## ABSTRACT

In this report, we present a BERT-based approach for solving the document dating problem presented in the DaDoEval competition at EVALITA 2020. In this work the dating problem is addressed as a text classification problem. BERT and BERT-based models have shown an extraordinary ability in many natural language processing tasks, often beating the current state-of-the-art by a large margin. One of the main advantages of these models is that they only need to be fine-tuned on the actual task in order to perform well, so a huge corpus of text is not needed. One of the main problems in the dating document task addressed in this work is the length of the documents, often surpassing the maximum sequence length of the BERT model. In order to overcome these difficulties, we did experiments using different truncation methods. Moreover, we also experimented with different embeddings in order to give to the classification head the best document representation possible. Our best solution is in line with the state-of-the-art in document dating, showing results aligned with the other participants in the competition.

***K*eywords** Document dating · Natural language Processing · BERT

## 1 Task Description

The task of dating documents consists of assigning a temporal span to the examined document, i.e. recognizing when a document was written or published. Dating documents is extremely important for many different tasks, such as information retrieval, temporal reasoning, text summarization, and analysis of historical text [13]. In the latter case, the task of dating documents is extremely important, since many false or fake documents have been produced (e.g. the famous *Donation of Constantine*). Dating document is a difficult task since many different aspects of the document have to be taken into consideration, such as temporal structure, style, word choices, citation of other events, and many more.

Apart from being a key step in all the aforementioned tasks, document dating is also an important task per se, since it can be used to process large archival collections. In particular, when some documents in a collection have not been dated, supervised approaches could be applied to learn from the documents with a date which time span can be assigned to those who are not provided with temporal metadata.

The task has already been addressed in other languages, namely French, English, Polish, etc. Along this line, for the 2020 edition of the EVALITA conference[1] it was proposed a document dating task using the Alcide De Gasperi's corpus of public documents [12] as a use case. The task is named DaDoEval, from Dating Documents Evaluation.

The corpus consists of a collection of documents and letters issued by Alcide De Gasperi between 1901 and 1954. The fact that all the documents in the corpus were issued by the same person removes the effect of different author styles in the classification. However, the limited time span covered by the documents could pose serious issues for the classification since the changes in language could be minor and not so relevant for the dating process.

[1] `http://www.evalita.it/2020`

The DaDoEval includes 6 sub-tasks, that can be tackled in isolation:

1. *Coarse-grained classification on same-genre data*: participants are asked to assign each document in the test set to one of the main time periods that historians have identified in De Gasperi's life, reported in the table below. Each document in the training set is labeled with one of the five periods and test data are of the same genre of the training data.
2. *Coarse-grained classification on cross-genre data*: participants are asked to assign each document in the test set to one of the main time periods that historians have identified in De Gasperi's life, reported in the table below. Each document in the training set is labeled with one of the five periods but test data are of a different genre compared to the ones included in the training data.
3. *Fine-grained classification on same-genre data*: participants are asked to assign each document in the test set to one temporal slice of 5 years. Each document in the training set is labeled with a temporal slice and test data are of the same genre of the training data.
4. *Fine-grained classification on cross-genre data*: participants are asked to assign each document in the test set to one temporal slice of 5 years. Each document in the training set is labeled with a temporal slice but test data are be of a different genre compared to the ones included in the training data.
5. *Year-based classification on same-genre data*: participants are asked to assign each document in the test set to its exact year of publication. Each document in the training set is labeled with the exact year of publication and test data are of the same genre of the training data.
6. *Year-based classification on cross-genre data*: participants are asked to assign each document in the test set to its exact year of publication. Each document in the training set is labeled with the exact year of publication and test data are of the different genre compared to the ones included in the training data.

The periods defined by historians for tasks 1 and 2 are reported in Table 1.

| **Class** | **Period** | **Years** |
|---|---|---|
| 0 | Habsburg years | 1901-1918 |
| 1 | Beginning of political activity | 1919-1926 |
| 2 | Internal exile | 1927-1942 |
| 3 | From fascism to the Italian Republic | 1943-1947 |
| 4 | Building the Italian Republic | 1948-1954 |

Table 1: Periods used to classify the documents in tasks 1 and 2.

For this project we decided to focus on tasks 1 and 3, so coarse-grained and fine-grained classification on same genre data.

## 2 Background

One of the major goals of natural language processing has always been to produce a strong *word embedding*, namely an information-rich representation of a word in the form of numerical values. The majority of machine learning or artificial intelligence algorithms were developed to work with real numbers, or in general continuous data, differently from words that are discrete. Transforming words to real-valued vectors allows using common and effective machine learning techniques and models, such as neural networks or support vector machines to classify or categorize text.

A good embedding maintains the semantic relations between words, namely transform to close vector words that have close semantic relations. As an example, the embedding for the word "cake" should be near the embedding of the word "bakery" and distant from the embedding of "typewriter". Embeddings of single words are the basic blocks for building sentence or periods embedding, which enclose the overall meaning of a piece of text.

The first revolution in NLP comes in 2013 when Mikolov *et al.* proposed the word2vect algorithm [5] to automatically calculate word embeddings from large text corpora. The method is pretty simple, and consisted of a bottleneck neural network trained in the following way: 1) select a random word *w* from the corpus 2) pick a random word $w_c$ from the context of *w* (the n previous and following words of *w*) 3) transform *w* and $w_c$ in one-hot vectors 4) input *w* to the neural network and try to predict $w_c$ (skip-gram model). After the training procedure is finished take only the first part of the network (from the input layer to the bottleneck) and use it to produce the word embeddings.

The word2vect algorithm is simple and straightforward, but it has two main drawbacks: it does not manage out of vocabulary words (since the words have to be transformed to one-hot vectors new words are not allowed) and is not contextualized, i.e. given the same word it always produces the same embedding. This can be problematic since the embedding of the word "rose" in the sentence "My boyfriend bought me a rose for Valentine's day" is the same as the one in the sentence "The interest rate rose dramatically in 2020" even if in the two sentences the meaning of "rose" is completely different.

One of the first contextualized word embedding models was ELMo [7]. ELMo uses two recurrent neural networks trained with the aim of predicting the next word given a sequence of words, so the context is included into word prediction. Both the recurrent network are 2-layers LSTM networks, one to predict the next word given the left context, and the other to predict the next (previous) word given the right context. ELMo deals with out of vocabulary words using a character CNN network. The final embedding is calculated by concatenating the embeddings produced by the two LSTM networks in each layer (including the input embedding) and then performing a weighted sum of the resulting vectors.

## 2.1 Transformers

ELMo overcomes the major issues of word2vect, but it has two major problems. First of all the model is not parallelizable, since each word has to be presented sequentially to the network. This is a major issue since training on large corpora could be very slow and even the generation of embeddings could be slow compared to other models or even to word2vect. The other major disadvantage of ELMo is that it struggles with long-term context dependencies since words are processed one at a time and even an LSTM network suffers from forgetting.

The transformer model [14] was initially proposed as a model for machine translation, and it is composed of an encoder and a decoder part. Each of the two components is composed of several encoding or decoding layers, in turn, composed of peculiar layers. A detailed visual representation of a transformer model is shown in Figure 1

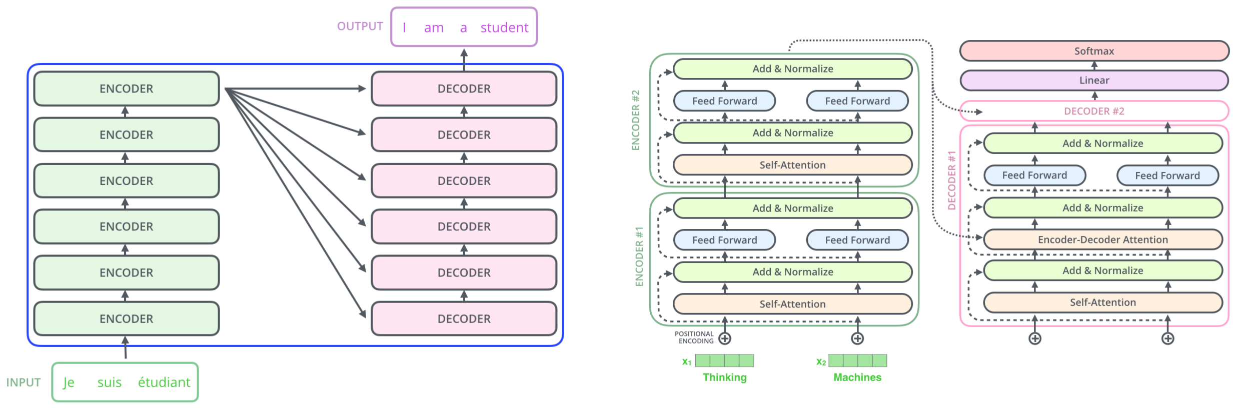

Figure 1: The general (left) and detailed (right) transformer architecture. Images from *The Illustrated Transformer* (`http://jalammar.github.io/illustrated-transformer`).

Each transformer's encoding block is composed of a self-attention layer followed by a feed-forward layer. After each layer is inserted an add and normalization layer. As the model processes each word (each position in the input sequence), self-attention allows it to look at other positions in the input sequence for clues that can help lead to a better encoding for this word. First, for each token in input three representations are calculated: the query, key, and value vectors. Those vectors are obtained by calculating the input embedding by trainable matrices. Then for the word in exam is calculated the score, namely the dot product between the query vector of the token in exam and the key vector of every other input vector. All the scores are then normalized using a softmax and multiplied for the value vector of each word. The weighted value vectors are then summed up and the output of the self-attention layer for the position examined is produced. To expand the model ability more attention heads are often used in each encoding layer. After the self-attention layer, there is a residual connection (the input of the attention layer is summed to the output) and layer normalization is applied. The encodings are then passed to multiple feed-forward neural networks and normalized again, before being passed to another encoder layer.

The decoder part of the transformer model is similar to the encoding. The output of the top encoder is transformed into a set of attention vectors that are used by each decoder in its encoder-decoder attention layer, which helps the decoder focus on appropriate places in the input sequence. The self-attention layers in the decoder operate in a slightly different way than the ones in the encoder. In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence. This is done by masking future positions before the softmax step in the self-attention calculation. The Encoder-Decoder Attention layer works just like multi-headed self-attention, except it creates its queries matrix from the layer below it, and takes the keys and values matrix from the output of the encoder stack.

### 2.2 BERT

BERT [2], which stands for *Bidirectional Encoder Representations from Transformers*, is a language model build using the encoding part of the transformer architecture. BERT takes as input a sequence of tokens and outputs the contextualized embeddings for each token. The base model of BERT is composed of 12 encoding layers with 12 attention heads each. The output embeddings have dimension 768 and the model takes a sequence of maximum length of 512 tokens. It also exists a large BERT model composed of 24 encoding layers with 16 attention heads each, and an embedding dimension of 1,024.

The original BERT implementation was trained on two tasks: masked language modeling and next sentence prediction. The former task consists of masking about 15% of the word in the input with the `[MSK]` token. The goal of the model is to predict the masked word outputting the corresponding embedding. For the second task, the `[CLS]` token is inserted as the first token in the sequence. The model is fed with two sentences, separated by the `[SEP]` token. The goal of the model is to output the probability that the second sentence follows the first one. The embedding of the `[CLS]` token is used as input to a neural network in this case. This means that the embedding of the `[CLS]` token contains aggregated information about the two sentences in the input sequence, combining all the information in the single word embeddings. For text classification (e.g. sentiment analysis or hate speech detection) the embedding of the `[CLS]` token is used as input to a classification head, typically a feed-forward neural network.

The most important characteristic of BERT is that it can be fine-tuned to a huge variety of task without the need to retrain all the models from scratch. The long training on huge corpora is done only one time, and after that, the resulted pretrained model is used as a starting point for almost any NLP task. This allows the model to be trained and reaching state-of-the-art performance without the need of an enormous training set. Apart from the classification or task-specific heads used on top of the base model, also the parameter of BERT are updated when fine-tuned. This allows producing more precise embedding for the examined task, using a reasonable amount of data and time. Fine-tuning BERT (even the basic model) has increased the state-of-the-art in many NLP tasks, both for the English [3] and other languages [11].

## 3 Approach

Since the introduction of contextualized word embeddings, the field of natural language processing changed dramatically. The availability of pretrained language models trained on a huge amount of data has been a key point in the recent advances in the field. Language models such as OpenAI GPT [8] or BERT [2] have surpassed state-of-the-art results in a wide variety of NLP tasks [3], including text classification such as sentiment analysis or hate speech detection. The major advantage of these large language model is that once trained on a large language corpus, they can be fine-tuned and adapted to any task without any major modification or retraining.

In the last few years specific BERT models trained on large Italian corpora have been released, notably *Gilberto*[2] and *Umberto*[3]. Both the model are trained using the Italian subset of the OSCAR corpus [6] and follow the pretraining philosophy of *Roberta* [4], but *Umberto* uses more sophisticated techniques during training, such as the sentencePiece unsupervised tokenizer[4] and whole word masking. As for the English language, even for Italian, the introduction of large language models improved the state-of-the-art in many NLP tasks, including document classification [11].

Given these premises, we decided to address the document dating task with BERT, in particular with the pretrained models *Gilberto* and *Umberto*.

The first problem we have to face is how to deal with long text. The corpus, described in details in subsection 4.1, is composed of documents with different numbers of words and with a high variance. The shorter document is composed of only 20 words, while the longest one contains 87,471 words. The average number of words per document is 1,082. Dealing with such long text is not easy, since the resulting tokenization produces token sequences far larger than the default BERT input length of 512 tokens. In order to use a pretrained BERT model, we have to truncate the token sequence to a fixed value less or equal to 512. In [1] is shown that simply truncating the token sequence to 512 can yield state-of-the-art results in document classification, but the length of the documents used is much shorter than the

[2]`https://github.com/idb-ita/GilBERTo`

[3]`https://github.com/musixmatchresearch/umberto`

[4]`https://github.com/google/sentencepiece`

ones used in this work. In [10] the authors analyzed more sophisticated strategies in order to deal with long text. Since often the relevant information of a document is contained at the beginning or at the end, the authors propose to truncate the token sequence in the following ways:

**Head only** : only the first tokens are used.

**Tail only** : only the last tokens are used.

**Head+tail** : a union of the first and last tokens are used.

More complex strategies, such as divide the token sequence (or even the text) in subsequences of length less than 512, extract the embedding for each subsequence, and combine or use all the embeddings together can be tested, but in general, a simple truncation yield the best results for text classification [10]. For this project, we decided to explore only simple truncation (head only, tail only, and head+tail).

Another design choice we have to make is what information to use to classify the document. BERT output a contextualized embedding for each input token, so many strategies can be explored. Usually, for text classification, the embedding corresponding to the `[CLS]` token is used as input for a feed-forward neural network use to classify the input sentence. However, since the `[CLS]` embedding is trained on next sentence prediction, is not a good sentence content representation. To use the embedding of the `[CLS]` token the BERT model cannot be used as-is, but it must be fine-tuned to the examined task.

BERT is composed of 12 different encoding layers, and each layer captures different features of the input. Using the output of the last layer is not always the best choice, since it could be “too close” to the target function the original BERT model was trained for. Using the embeddings of the second-to-last or aggregate the embedding of few layers can increase the performance for the specific tasks [2]. In this project, we explore three different embeddings: last layer, second-to-last layer, and the sum of the embeddings of the last four layers.

On top of the BERT model, we used a classification head for dating the documents. The classification head is a feedforward neural network for all the experiments. It takes as input the 768-dimensional embedding of the `[CLS]` token calculated as described above, and it’s composed of two linear layers plus dropout between them. The classifier head architecture is the same for every experiment, apart from the number of output classes, which is 5 for task 1 (coarse-grained classification) and 11 for task 3 (fine-grained classification). The detailed architecture is shown in Table 2.

| **Layer** | **Description** |
|---|---|
| Dropout | p=0.1 |
| Linear | in=768, out=768 |
| Tanh | activation function |
| Dropout | p=0.1 |
| Linear | in=768, out=num_classes |

Table 2: The architecture of the classifier on top of the BERT model.

## 4 Experiments

### 4.1 Dataset

The dataset used in this work is the same used for the DaDoEval task in EVALITA 2020[5]. The corpus consists of 2,759 documents written by Alcide De Gasperi between 1901 and 1954. The corpus was split into a training and a test set following an 80-20 ratio, thus having 2,210 documents for training and the remaining 549 for testing. The classes are not well-balanced, with some periods having only few training documents. For example, in task 1 (coarse-grained dating) the class 2 (internal exile, period 1927-1942) has 150 documents, while class 4 (building the Italian Republic, 1948-1954) contains 632 documents (more than four times the class 2). The imbalance is even more pronounced in task 3 (fine-grained dating), where the span 1926–1930 has only 16 documents, while the period 1946-1950 has 599 documents (about 37 times more). The detailed distribution of documents in the different classes is shown in Table 3.

While written in Italian, the documents contain many words in foreign languages, especially German, French and English. Moreover, some documents contain whole sentences or periods written in another language, especially the

[5]`https://dhfbk.github.io/DaDoEval`

ones belonging to the period after the second world war when De Gasperi had many exchanges with foreign politics and officers. The German is mainly present in the documents belonging to the Habsburg years (1901-1918) when De Gasperi was a member of the Austrian Parliament.

| Task | Class | Train | Test |
|---|---|---|---|
| Coarse-grained | 0 | 572 | 140 |
| | 1 | 342 | 109 |
| | 2 | 150 | 37 |
| | 3 | 514 | 98 |
| | 4 | 632 | 165 |
| Fine-grained | 1901-1095 | 85 | 21 |
| | 1906-1910 | 256 | 65 |
| | 1911-1915 | 211 | 48 |
| | 1916-1920 | 109 | 42 |
| | 1921-1925 | 246 | 73 |
| | 1926-1930 | 16 | 2 |
| | 1931-1935 | 76 | 22 |
| | 1936-1940 | 62 | 13 |
| | 1941-1945 | 191 | 36 |
| | 1946-1950 | 599 | 129 |
| | 1951-1955 | 399 | 98 |

Table 3: Document distribution in the different classes for coarse and fine-grained dating.

## 4.2 Data Preparation

The corpus is composed of many `.txt` files, one for each document, plus a `.tsv` file that indicates the class of each document. The data was already split into train and test using the classical 80-20 split.

Both the training and the test data have been preprocessed in order to remove any leading or trailing whitespace (or any other space character, like tab). This preprocessing step is essential since both the *Gilberto* and *Umberto* models are based on *Roberta* [4], which tokenizer has been trained to treat spaces like parts of the tokens so a word will be encoded differently whether it is at the beginning of the sentence (without space) or not. The model was originally pretrained without any leading space, so we need to remove them to be in the same conditions as the original training. After that, all the newline characters `"\n"` have been replaced by spaces.

The resulting text is then concatenated to the corresponding class and all the training data is saved in a single `.csv` file to simplify the access to the data.

The validation set is derived from a further splitting of the preprocessed training set. Since the data is not balances, we can choose between two different strategies: balanced and unbalanced validation set. The former gives useful information about the classification accuracy on all the classes, even the least represented. Sometimes when the training data is unbalanced the model focuses only on the predominant classes and does not care at all about accuracy on the least represented categories. A balance validation set can help to detect this unwanted behavior and stop the training before the model starts to overfit. On the other hand, an unbalanced validation set (unbalanced in the same way the training and the test are unbalanced) can give a better approximation of the performance on the test set. Moreover, a balanced validation set further reduces the number of examples of under-represented classes in the training set. For all these reasons we decided to use an unbalanced validation set. For each experiment, the validation set is randomly selected as a subset of the training set with an 80-20 split.

## 4.3 Training and Hyperparameters

We used Pytorch (version 1.7.1) and the HugginFace transformers library [6] (version 4.3.0) for loading the data and performing the computation. For *Gilberto* we used the model `db-ita/gilberto-uncased-from-camembert`, while for *Umberto* we used `Musixmatch/umberto-commoncrawl-cased-v1`. Both the models are derived from *Roberta* and share the same architecture of the *CamemBERT* model, so 12 encoding layers and an hidden size of 768. Both the mode were pretrained on similar data, while for *Umberto* more refined techniques were used (see section 3 for more details).

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

For all the tasks we did not employ any hyperparameter search since the default fine-tuning hyperparameters have shown good results in similar tasks. We used Adam as optimizer, with a learning rate of 2e-5, $\beta_1 = 0.9$ and $\beta_2 = 0.999$ with cross-entropy loss for all the experiments. We trained the models for 5 epochs and retained the model that performs better on the validation set. We validated the model two times per epoch, one after half the iterations and one at the end of each epoch. The maximum sequence lenght (MSL) is set to 512 for all the experiments since preliminary tests resulted that shorter MSL yields lower accuracies. We used a batch size of 8 due to GPU memory limitations. All the experiments are repeated 3 times, so the results are presented as the mean and the standard deviations of the runs.

We train our models on an NVidia TITAN X GPU with 3,584 Cuda cores and 12GB of memory. A five epochs training took about 10 minutes to be completed.

### 4.4 Results

First, we evaluated the effects of different input sequences to the model, maintaining the embedding fixed. After that, we select the best token input sequence and evaluate the effect of different embeddings. These tests are conducted with both the models using the task 1 goal, so the simpler coarse-grained scenario. After this first phase, the best model architecture on the coarse-grained scenario is trained on task 3 (fine-grained scenario). In the end, the best models are compared with the submissions to the DaDoEval competition.

#### 4.4.1 Effect of different input sequences

As mentioned in section 3 we would like to test three different strategies for truncating the input to the MSL of 512: head only, tail only, head+tail. In the first strategy, only the first 512 tokens of the sequence are taken. In the tail-only strategy, only the last 512 tokens of the document are used as input for the model. In the head+tail strategy, the first 256 and the last 256 tokens are concatenated together forming a hybrid input sequence to the model. In all these tests the embedding used as input for the classifier head is the embedding of the `[CLS]` token after the last layer of BERT. The results are shown in Table 4.

| | **Head only** | **Tail only** | **Head+tail** |
|---|---|---|---|
| ***Gilberto*** | $78.96 \pm 1.32$ | $78.77 \pm 0.86$ | $78.38 \pm 1.67$ |
| ***Umberto*** | $\mathbf{83.91 \pm 0.45}$ | $75.68 \pm 2.10$ | $80.26 \pm 1.50$ |

Table 4: Macro average F1-score for different input token sequance truncation methods on the coarse-grained dating task.

For both the models, the best truncation method resulted in taking only the first 512 tokens of the sequence. Especially for *Umberto* this simple truncation method yields an overall macro F1-score of almost 84, which is a surprising result since the model was not trained on long text and only standard fine-tuning has been used. *Umberto* performs better than *Gilberto* using the first 512 tokens and the union of the first 256 and the last 256 tokens, but the score dropped to 75 if only the tail of the sequence is used. This can be explained with the different training procedures of the two models, in particular, *Umberto* was trained using *sentencePiece*, which takes into account whitespaces. The same word with leading whitespace is tokenized differently than the same word in the same context without leading whitespace. Since *Umberto* is trained without leading whitespace at the start of a sentence, starting the token sequence in the middle of a period is different than starting from the beginning.

In general *Gilberto* shows more stability regarding the type of input sequence, with all the test yield scores around 78.

#### 4.4.2 Effect of different embeddings

We now want to estimate the effect of using different embeddings as input for the classification head. As mentioned before, by default BERT (and the two models used in this project) uses the embedding of the `[CLS]` yielded by the last encoding layer as a summary of the whole sentence. As described in section 3, there are other ways to extract aggregate sentence information. In the following experiment, we used the embedding of the `[CLS]` token yielded by the second-to-last (STL) layer and the sum of the embeddings of the last four layers. In all the experiments we fed the model with the first 512 tokens since the previous experiment resulted in the best strategy on this particular task. The results are shown in Table 5.

Differently from the previous experiments, in this case, the two models exhibit different behaviors depending on what embedding is used as input for the classification head. For *Gilberto* using the embedding of the `[CLS]` token yielded by the second to last layer is beneficial, while using the sum of the embeddings of the last four layers decreases the

| | Last layer | STL layer | Sum last 4 layers |
|---|---|---|---|
| ***Gilberto*** | $78.96 \pm 1.32$ | $80.29 \pm 0.63$ | $77.94 \pm 1.09$ |
| ***Umberto*** | $83.91 \pm 0.45$ | $83.94 \pm 0.44$ | $\mathbf{84.55 \pm 0.92}$ |

Table 5: Macro average F1-score for different embeddings on the coarse-grained dating task.

score by more than 1 point compared to the basic last layer embedding. On the contrary, for *Umberto* the best result is obtained using the sum of the last four layers embedding, while using the STL layer embedding does not show any significant improvement over using the last layer embedding.

A detailed report of the best run for the coarse-grained dating task is shown in Table 6.

| | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| 0 | 93.89 | 87.86 | 90.77 | 140 |
| 1 | 78.57 | 90.83 | 84.26 | 109 |
| 2 | 88.57 | 83.78 | 86.11 | 37 |
| 3 | 82.22 | 75.51 | 78.72 | 98 |
| 4 | 87.43 | 88.48 | 87.95 | 165 |
| **Accuracy** | - | - | 86.16 | 549 |
| **Macro avg** | 86.14 | 85.29 | 85.56 | 549 |
| **Weighted avg** | 86.47 | 86.16 | 86.17 | 549 |

Table 6: Details of the best run on the coarse dating task. The model used is *Umberto* truncating the input sequence to 512 tokens and using as document embedding the sum of the embedding of the `[CLS]` token of the last four layers.

#### 4.4.3 Fine-grained task and comparison with state-of-the-art

As mentioned in section 1, the same two tasks addressed in this paper are used for the DaDoEval task at EVALITA 2020 [9], so we can compare our approach with other participants to the competition.

For the coarse-drained dating, two teams participated in the competition, named "matteo-brv" and "rmassidda". The former team used a Support Vector Machine classifier on top of a set of style-based features: TF-IDF weighted character and word n-grams, and number of word tokens per document. The latter team used a double representation extracted using Sentence-Bert and a bag-of-entities obtained using a Named Entity Recognition system and fed them to a multi-layer neural network. For more details about the methods of the two teams refer to [9].

In the competition, the teams were evaluated via the macro F1-score and only 2 runs are allowed. For a fair comparison we select the best and the worst of our runs. Results for the coarse-grained dating task are shown in Table 7.

| Team | Run | Macro F1 |
|---|---|---|
| matteo-brv | 1 | 93.4 |
| matteo-brv | 2 | 93.4 |
| rmassidda | 1 | 85.8 |
| **our** | best | 85.6 |
| rmassidda | 2 | 85.5 |
| **our** | worst | 83.2 |
| baseline | - | 82.7 |

Table 7: Comparison with other participants to the DaDoEval competition on the coarse grained dating task.

Our model performs pretty well compared to the baseline and the other participants, almost matching the results of the "rmassadda" team that used a more sophisticated approach. Is worth noting that large language models seem to be far away from the best solution for this task, maybe due to the limited input sequence length that can decrease the performance when long text is examined.

For the fine-grained classification task, we used the same model structure of the best model resulted from the coarse-grained dating task. So we truncated the input sequence to the first 512 tokens and use as document embedding the sum of the embeddings of the `[CLS]` token of the last four layer of the BERT model. In this task, there are 11 classes with heavy unbalanced data (see table Table 1). In particular, class 5, corresponding to the period 1926-1930, contains only 16 training images and 2 test images. This is particularly problematic for the evaluation since the macro F1-score can be heavily influenced by the results on a single class composed of only two test images. For these reasons in the result is also indicated the weighted F1-score, where the mean of the f1-scores is calculated weighting each class according to the number of corresponding documents.

The fine-grained dating task corresponds to task 3 of the DaDoEval competition [9]. In this task, only the team "rmassadda" participated with two runs. The results are reported in Table 8.

| Team | Run | Macro F1 | Weighted F1 |
|---|---|---|---|
| rmassidda | 2 | 63.8 | - |
| rmassidda | 1 | 57.9 | - |
| baseline | - | 48.5 | - |
| **our** | best | 46.6 | 57.4 |
| **our** | worst | 43.1 | 55.0 |

Table 8: Comparison with other participants to the DaDoEval competition on the fine-grained dating task.

In none of the three runs, one of the two test documents corresponding to the years 1926-1930 (class 5) has been correctly classified, yielding a default F1 score of zero for that class. This lowers the macro F1 score for our model in an unfair manner, so the weighted F1 score could be the preferred choice to evaluate the performance of the model.

If we take into consideration the macro F1-score our model performs slightly worse than the baseline model, and more than 10 points worse than the model of rmassidda. If the weighted F1 score is considered, our model is in line with the performance of the competition winner, easily outmatching the baseline. The mean and the standard deviation of the three runs are: $45.43 \pm 1.97$ for the macro f1-score and $56.28 \pm 1.22$ for the weighted F1-score.

## 5 Conclusions and Future Work

In line with the observation described in other works [1, 10], truncating the input sequence to the first 512 tokens and using the sum of the embeddings of the last four layers has yielded the best results, even in this difficult task. Our model was not specifically trained to deal with long sequences, so the results are satisfactory and in line with similar models used for the same task. In accordance with [11], the best BERT model for modeling the Italian language seems *Umberto*, arguably due to the more advanced techniques used during training. Our model performs well in the first task (coarse-grained dating), showing results in line with other participants in the same competition. For the second task, the performances decreased in comparison to the other baselines, but this is mainly due to the calculation of the macro F1-score that is highly penalized if some underrepresented class is predicted with very low accuracy. The weighted F1-score shows that the overall performance of our model is only slightly lower than the current state-of-the-art, even if the BERT model used was not specifically trained for dealing with long sequences.

As future work, we want to compare our basic BERT model with models specifically trained for sentences or longer text, such as sentenceBERT od DocBERT [1]. Another interesting exploration could be to not truncate the input sequence to 512 but divide the long text into smaller sequences that can be entirely fed to the model. Each subsequence embedding can then be used in multiple ways, e.g. as a sequence of embedding using a recurrent neural network for the classification, or combining all the embeddings together in order to use a more standard feed-forward network head.

## References

[1] Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. Docbert: Bert for document classification. *ArXiv*, abs/1904.08398, 2019.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, jun 2019. Association for Computational Linguistics.

[3] Han He and Jinho D. Choi. Establishing strong baselines for the new decade: Sequence tagging, syntactic and semantic parsing with bert. In *FLAIRS Conference*, 2020.

[4] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[5] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[6] Pedro Javier Ortiz Suarez, Benoit Sagot, and Laurent Romary. Asynchronous pipelines for processing huge corpora on medium to low resource infrastructures. Proceedings of the Workshop on Challenges in the Management of Large Corpora (CMLC-7) 2019. Cardiff, 22nd July 2019, pages 9 – 16, Mannheim, 2019. Leibniz-Institut für Deutsche Sprache.

[7] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke 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, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[8] A. Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[9] Rachele Sprugnoli Stefano Menini, Giovanni Moretti and Sara Tonelli. Dadoeval @ evalita 2020: Same-genre and cross-genre dating of historical documents. *Proceedings of the Seventh Evaluation Campaign of Natural Language Processing and Speech Tools for Italian. Final Workshop (EVALITA 2020)*, 2020.

[10] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to Fine-Tune BERT for Text Classification? In Maosong Sun, Xuanjing Huang, Heng Ji, Zhiyuan Liu, and Yang Liu, editors, *Chinese Computational Linguistics*, pages 194–206, Cham, 2019. Springer International Publishing.

[11] Fabio Tamburini. How "bertology" changed the state-of-the-art also for italian nlp. *Proceedings of the Seventh Italian Conference on Computational Linguistics*, 2020.

[12] Sara Tonelli, R. Sprugnoli, and G. Moretti. Prendo la parola in questo consesso mondiale: A multi-genre 20th century corpus in the political domain. In *CLiC-it*, 2019.

[13] Shikhar Vashishth, Shib Sankar Dasgupta, Swayambhu Nath Ray, and Partha Talukdar. Dating Documents using Graph Convolution Networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1605–1615, Melbourne, Australia, jul 2018. Association for Computational Linguistics.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.