POLITECNICO DI TORINO

Master's Degree
in Ingegneria Informatica
(Computer Engineering)

Master's Degree Thesis

Modeling and classifying textual data through Transformer-based architecture: a comparative approach in Natural Language Processing

Politecnico di Torino

Supervisor
Prof.ssa Tania Cerquitelli
.....
Company Supervisors
Alessio Bosca
.....
Gianpiero Sportelli
.....

Candidate
Valentina Margiotta
.....

Academic Year 2020-2021

Ai miei genitori, a mio fratelllo e alla mia famiglia tutta, grazie per esserci sempre stati.

Summary

In the scenario of deep learning models applied to Natural Language Processing (NLP), the Transformer architecture has brought great interest thanks to its attention mechanism, which helps models to focus on specific parts considering the relationship between words, independently of where they are placed in a sentence.

Many models based on this type of architecture have been developed by the research communities. In my Master Thesis I examined the following language models:

- Bidirectional Encoder Representations from Transformers (BERT)
- Efficiently Learning an Encoder that Classifies Token Replacement Accurately (ELECTRA)
- Generative Pre-Trained 2 (GPT-2)

These models have been applied to the NLP text classification task on different datasets both in English and in Italian in order to evaluate and compare the performances obtained with the models pre-trained on an English corpus rather than with a multilingual model.

Specifically, two types of classification have been considered in this Master Thesis, multi-class and multi-label. They differ in that in a multi-label classification problem, the training set is composed of sentences, each of which can be assigned to multiple categories instead of a single label.

In this context, therefore, the above models have been trained and fine-tuned with the aim of assessing their reliability and accuracy in predicting the correct labels associated with the text given as input to the models.

Following the various experiments carried out, it was observed that all the used models were able to correctly predict with high accuracy the label associated with a textual input and it was possible to carry out a textual classification task in a short time thanks to the architecture of the transformers, which are models already intensely pre-trained on a large corpus of data and therefore they can be fine-tuned in an inexpensive way on numerous downstream NLP tasks.

Acknowledgements

Ed eccomi qui, dopo questi quasi sei anni per nulla scontanti e non sempre semplici, oggi sono arrivata al termine del mio percorso universitario. Chi se lo sarebbe aspettato? Sicuramente la mia famiglia, da sempre fonte di supporto, ma di certo non io. Se sono qui oggi lo devo soprattutto al sostegno ricevuto in questi anni da parenti e amici più stretti, a chi ci è sempre stato ma soprattutto lo devo alla mia tenacia che, nonostante i momenti di sconforto, mi ha permesso di fronteggiare a testa alta gli ostacoli incontrati durante questo percorso e di proseguire sino a tale traguardo.

Un sentito grazie a tutti coloro che mi hanno permesso di arrivare fin qui e di portare a termine questo lavoro di tesi.

Ringrazio la mia relatrice, la Prof.ssa Tania Cerquitelli, e i miei tutor aziendali, Alessio Bosca e Gianpiero Sportelli, per avermi dato fiducia e avermi dato la possibilità di lavorare a questa tesi ampliando il mio bagaglio di conoscenze.

Vorrei ringraziare i miei amici, quelli veri, che dopo tutti questi anni sono rimasti al mio fianco. Un grazie speciale a Salvo e Giulio, colleghi, amici che hanno dovuto assorbire tutte le mie ansie pre-esami, a voi un caro grazie per avermi sempre ascoltata e "sopportata". Un caloroso ringraziamento lo devo alla mia migliore amica Silvia I. per essere rimasta tale dopo quasi venti anni di amicizia, grazie per l'autostima che riesci ad infondermi e per essere sempre la mia fedele confidente. Ringrazio anche in particolar modo Silvia F. perche' la nostra amicizia è rimasta salda dopo questi anni di distanza geografica.

In più vorrei ringraziare tutti gli amici che il Collegio Einaudi mi ha permesso di conoscere e in particolare Arianna, Savino e Salvo.

A mamma e papà, al loro costante sostegno e ai loro insegnamenti senza i quali oggi non sarei ciò che sono. Vi dico grazie perche' senza di voi, tutto questo non sarebbe stato possibile. A mio fratello Alessio, grazie per essere il mio costante supporto e per credere sempre in me. Ringrazio di cuore i parenti più stretti, Janmanuel per essere più di un semplice zio, nonna Dora che mi è sempre stata vicina nonostante la distanza e mi ha trasmesso il suo grande amore e i nonni Franco ed Elena che da lassù oggi mi staranno guardando, a voi tutti, che vorrei foste qui oggi per abbracciarvi, mando un grosso bacio.

Grazie a tutti voi, *Valentina*

Contents

List of Tables

List of Figures

Your time is limited, so don't waste it living someone else's life. Don't be trapped by dogma - which is living with the results of other people's thinking. Don't let the noise of others' opinions drown out your own inner voice. And most important, have the courage to follow your heart and intuition. They somehow already know what you truly want to become. Everything else is secondary.

[Steve Jobs, Commencement speech at Stanford University in 2005]

Part I

Background and used architectures

Chapter 1

Introduction

1.1 Natural Language Processing (NLP)

In the field of machine learning and linguistics world, Natural Language Processing is focused on understanding the interaction with the human language. With the respect to the formal language, which includes the computer language, the natural one is more complex due to the intrinsic characteristics of ambiguity of human language, which make the process of automatic processing of information by a computer, written or spoken in a certain language, more difficult. The main idea behind NLP is to be able to understand words with the whole context and not just considering them individually.

Thanks above all to the contribution of increasingly advanced Artificial Intelligence techniques, NLP finds many application areas both in the written and spoken language processing sectors. Machine question answering, automatic translation, speech recognition, named entity recognition and virtual assistants such as Amazon Alexa, Google Assistant or Siri by Apple are just some examples of NLP application tasks.

In this Master Thesis, text classification task will be explored taking into account two sub-tasks, Multi-Label and Multi-Class (also named Intent Recognition) text classification.

There are many interesting applications for text classification such as sentiment analysis, stance detection and spam detection. The former, which helps the natural language processing algorithm to determine the sentiment or emotion behind a text, can analyze language used in social media posts, reviews and responses to extract attitudes and emotions in response to events, daily facts, products and more. Stance Detection is instead a classification task aiming at determining the position of the author of a given text concerning the topic treated in the text itself and this automatic system can be useful in the politic and social analysis. The latter is used by companies like Google and Yahoo to scan and classify emails analyzing the text inside them stopping spam or phishing before they even enter in our inbox.

1.2 Pre-Transformer Models: RNN and LSTM

As previously mentioned, in NLP the role of context in a sequence is very important. Considering a sample, each word can indeed be related more or less from the other words in the sentence. For this reason, a neural network has to take into account tokens that have been preceded by the current one.

Therefore, a convolutional neural network (CNN) is not a good solution for data that come in a sequence or when data in different parts of a sequence can affect each other. This kind of networks also fail interpreting temporal information and have no memory about its previous states. In order to deal with sequential data, like sentences, recurrent neural networks are used. RNNs are ideal for solving problems where the sequence is more important than individual item themselves.

Let's suppose, for example, to have a sentence in which the model has to predict the last word. RNN propagates information from the beginning of the sentence through to the end, starting with the first word of the sentence, the hidden value at the far left, thus the first value is computed. Then, it propagates the computed information, considering the second word in the sequence and the previous hidden state in order to get new values. So, the computations made at the last step have information from all the words in the sentence and at the final step the RNN is able to predict the word.

The main advantage of RNNs is that they propagate information within the sequence and the computations share most of the parameters.
Furthermore, RNNs have an internal memory (state) that allows the previous inputs to affect the future predictions. Knowing what the previous words were, leads to predict the next word in a sentence with more accuracy.
At each time-step, RNN takes as input, aside from previous outputs, also the hidden states of neurons and makes a prediction. Then, it propagates a new hidden state to the next time-step. So, hidden states propagates information though time and the basic recurrent units have two inputs at each time, respectively the previous hidden state and the current input (as shown in Figure 1.1).

However, there are two main disadvantages with RNNs. The first one is that the RNNs architecture, optimized for recalling the immediate past, has some problems with longer sequences. The second issue that affects the power of RNNs is the vanishing (gradients can tend to zero) or exploding (gradients can tend to infinity) gradients that can cause the model training to fail. This can arise due to the fact that RNNs propagates information from the beginning of the sequence through to the end. In fact, the gradients are calculated during backpropagation when the weights receive an update that is proportional to the gradients with respect to the current weights of that time step. The exploding gradients consists, in a network with many layers, in a large update of weight that cause the instability of the whole network, situation that can lead to numerical overflow.

There are some different solutions to the vanishing problems such as the use of an identity matrix with a ReLU activation or it can be possible to perform the gradient clipping in order to limit the magnitude of the gradients to the specified value chosen. However, the best known solution to the vanishing gradient problem is obtained with LSTMs.

Long short-term memory (LSTM) is an evolution of RNN which learn when to remember and when to forget. Unlike RNN, LSTM has a cell state (as shown in Figure 1.2), representing the memory, and the hidden state, where computations are performed during training in order to decide what changes to make. An LSTM has three gates: the forget gate decides what to keep, the input one decides what to add and finally the output gate decides the next hidden state. Due to the fact that the gates regulate how much data can get into next time-step and how weights can be optimized, the vanishing problem no longer persists.

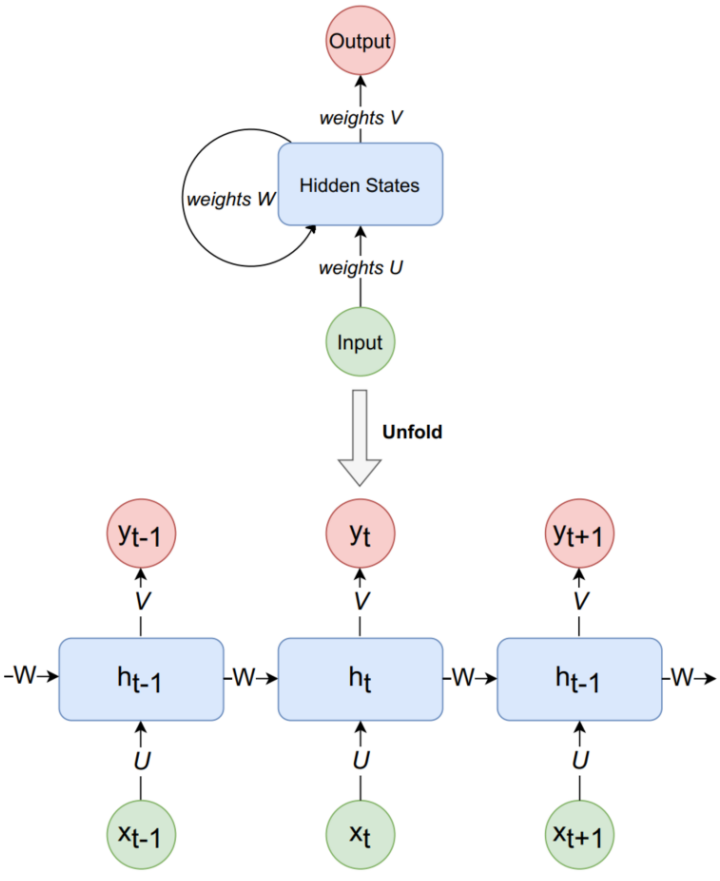


Figure 1.1: RNN Architecture

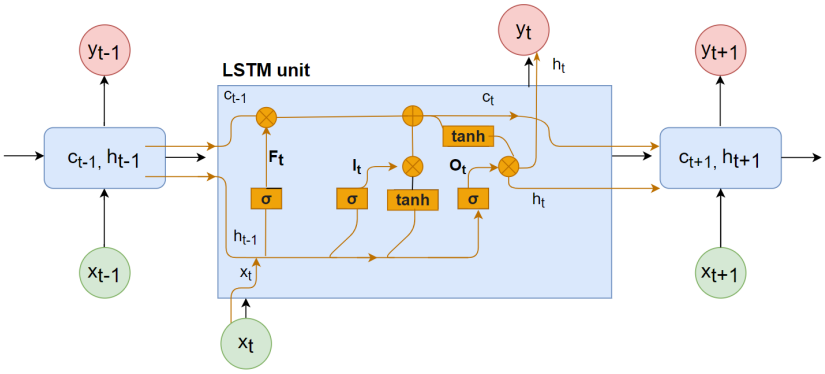


Figure 1.2: LSTM Architecture

1.3 Transformer Model

In 2014 Google introduced the traditional sequence-to-sequence model (Seq2Seq) for machine translation [24]. This model was a revelation at that time. In fact, before that, the translation was based on a very basic procedure, each typed word was converted into the target language without taking grammar or the overall sentence structure into account. Seq2Seq, using deep learning, has therefore revolutionised the translation process by considering not only the current input during the translation phase but also the neighbouring context.

Essentially, Seq2Seq maps variable-length sequences to fixed-length memory, turning one sequence into another sequence of words. This is referred to as sequence transformation. The model, as illustrated in Figure 1.3, is composed of two components, an encoder and a decoder, both of them are typically a LSTM cell. The former takes the input sequence one step at a time, storing it in its internal hidden states which are then passed into the decoder network that will use them to predict the sequence. As a consequence, the encoder transforms each element into a corresponding hidden vector containing the item and its context, while, the decoder reverses the process, it transforms the vector into an output element, using the previous output as the input context.

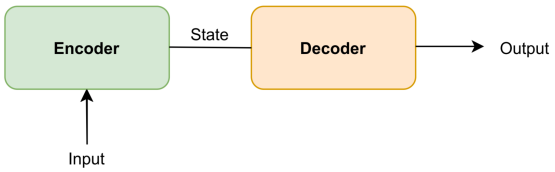


Figure 1.3: Seq2Seq encoder-decoder model overview

However, since traditional Seq2Seq model is based on the use of fixed length memory, in fact encoder hidden state is of a fixed sizes, its big limitation is given by the information bottleneck, a situation that occurs having long sequences. Furthermore, the sequential and recurrent nature of models previously explained such as RNNs and LSTMs doesn't allow parallel computation, which becomes a problem at longer sequence lengths. Processing very long sequences brings also to the loss of information and vanishing gradients problems.

In order to overcome all these problems with sequential networks, in 2017 a new network architecture was proposed with the paper *"Attention Is All You Need"* [26]: the Transformer. The new concept introduced with transformer-based architecture is the "Attention". This model, not implying any recurrent networks, is entirely based on self-attention to compute representations of its inputs and output without using aligned sequence as RNNs.

The mechanism of self-attention, called also intra-attention [26], relates different positions of a single sequence for the purpose of calculating a representation of the sequence itself and it has been applied with great success in many tasks such as textual entailment and learning task-independent sentence representations.

The goal of attention mechanism is to assign a score to each word in a segment based on its relevance within the segment itself. There are three components used for calculating attention weights and all of them are vectors: keys, queries and values. The attention function can be seen as a mapping of query and set of key-value pairs to an output which is computed as a weighted sum of the values where the weight of each value is obtained as output of a function that associate the query with the corresponding key. The attention mechanism therefore allows the model to focus on the most important parts of the sequence for each step, thus allowing to take into account also the context of the word in the sentence. In this sense, attention mechanisms allowed the modelling of dependencies without considering their distance in the input or output sequences.

Another important characteristic of transformers is that they do not require any sequential computational per layer but only one single step is required. Another difference with respect to RNNs is that with transformers the number of gradient steps that need to be taken from the last output to the first input is just one. Furthermore, the Transformer overcomes the biggest problem of the recurrent networks related to the length of the sequence by no longer suffering from vanishing gradients problems.

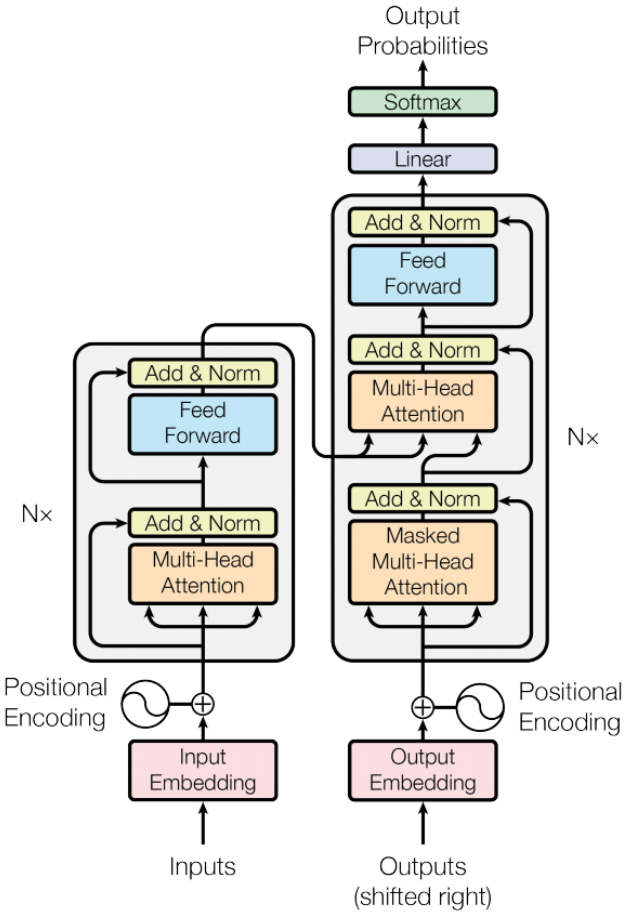


Figure 1.4: Transformer model architecture [26]

Analyzing in detail the architecture of Transformer, this is represented by an encoder-decoder stacked structure. As shown in Figure 1.4 both encoder and decoder modules are composed of a block repeated Nx times (Nx equals to 6). That layer has a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. Around each of these, there is a residual connection followed by a normalization layer.

Moreover, the decoder has a third sub-layer that performs multi-head attention over the output of the encoder stack. Another difference to the encoder stack is that in the decoder the multi-head attention mechanism is masked. Since the decoder is auto-regressive and it generates the sequence word by word, this masking prevents the decoder from looking at future tokens and from computing attention score for them. As a result, it is ensured that the prediction for the i-th position can only depend on the known outputs at position below i.

The particular self-attention mechanism introduced with transformer model is based on a scaled dot-product. Considering as input the matrix Q (representing queries), the matrix K (representing keys of dimension d_k) and the matrix V (representing values of dimension d_v), the attention function gives as output the result of softmax function applied on the dot product of the query with all keys, divided each by $\sqrt{d_k}$.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1.1}$$

Therefore, from the dot product multiplication of Q and K, a score matrix is computed which determines how much importance should be placed on one word compared to the others. The higher the score, the greater is the focus. Subsequently, the scores are scaled, dividing them by the square root of the size of the keys and queries, in order to obtain more stable gradients. The attention weights are obtained applying the softmax of the scaled scores, which gives the probability values between 0 and 1. Higher softmax scores therefore indicate the value of the words which the model associates more importance. Finally, the output vector is obtained by multiplying the attention weights by the matrix V.

Instead, the multi-head attentions is performed running the attention of the scaled dot-product several times in parallel. The multi-head model jointly attends to information from different representations at different positions over the projected versions of Q, K and V. Then, the attention function described above is applied in parallel, reaching different output values which are then concatenated and weighted. The projection operation of Q, K and V is shown in equation 1.2

$$\begin{aligned} MultiHead(Q, K, V) &= Concat(head_1, ..., head_h)W^O \\ where\ head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{1.2}$$

The two most commonly used attention functions analyzed above are shown in Figure 1.5.

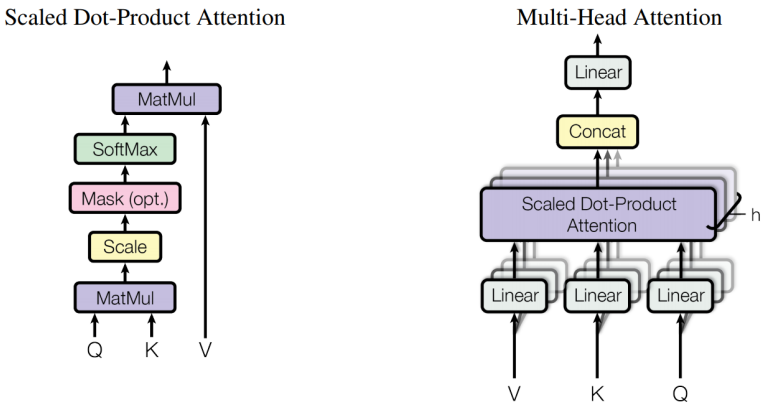


Figure 1.5: Scaled Dot-Product Attention and Multi-Head Attention [26]

Transformers also incorporate a positional encoding stage which encodes each inputs position in the sequence since the words order and the position is very important for any language. Since, unlike the recurrent layer, the multi-head attention layer computes the outputs of each inputs in the sequence independently, allowing the computational parallelization, there is the need to incorporate the positional encoding stage into the Transformer in order to model the sequential information for a given sequence.

Therefore, to add information about the position of the tokens within the sequence, positional encodings are added to the input embedding at the beginning of the encoder and decoder stacks and this is done using sine and cosine functions of different frequencies.

Chapter 2

Transformer-based architectures

The great success of the Transformer in a wide variety of NLP related tasks has led to the development of many pre-trained models. In this Master Thesis, three different models will be explored such as BERT, ELECTRA and GPT-2.

Transfer Learning

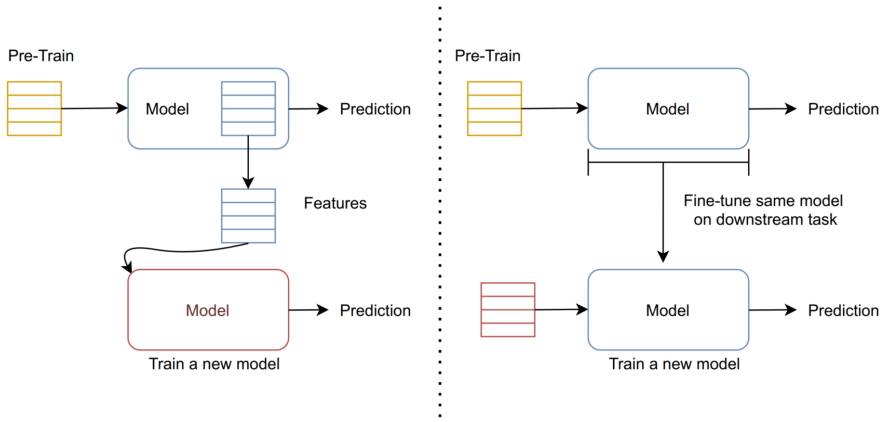


Figure 2.1: Transfer learning approaches: Feature-based (left) and Fine-Tuning (right)

Before going into more details with the transformer models, it is important to understand what is Transfer Learning (TL) in natural language processing. The main idea behind TL is to use the knowledge of an already trained machine learning model, that has learned from a task with a lot of available labeled training data, by applying it to a different task. As a consequence, instead of starting the learning process from scratch, we solve a related task through learned patterns. There are two approaches to transfer learning (as shown in Figure 2.1): "feature-based", which consists in extracting features

from a pre-trained model and then feeding them into a new model to get predictions and "fine-tuning" the pre-trained model keeping learnt weights as initial parameters and then fine tune them on downstream tasks such as translation, summarization, Q&A and text classification.
Reduction of training time, predictions improvement, allows you to use smaller datasets. These are the main advantages to transfer learning.

2.1 Transformer bidirectional encoder models

These models are non auto-regressive (autoencoding), that is, they can incorporate the context on both side of a word to gain better results. Being autoencoding models, they are based on the encoder part of the original Transformer architecture and they do not use mask in order to look at all the tokens in the attention heads. Usually, for pre-training, the inputs are corrupted by masking some tokens in order to then reconstruct the original sentence.

2.1.1 BERT

One of the most common and powerful pre-trained unsupervised natural language processing model proposed in 2018 by the Google AI Language team is BERT [17].
BERT, which stands for *Bidirectional Encoder Representations from Transformers*, is a famous transformer used for learning text representations and it makes use of transfer learning and pre-training.

BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly taking into account both the left and right context in all layers. As a consequence, the pre-trained BERT model can be fine-tuned with just one additional output layer on a wide range of tasks without any specific task architecture modifications. During the fine-tuning, the model is first initialized with the pre-trained parameters and then those parameters are fine-tuned using labeled data from the downstream tasks. Minimal differences exist between the pre-trained architecture and the final downstream one.

For the pre-training procedure, a corpus of 3.3 billion of words has been used by the researchers. In particular, they used the BookCorpus of about 800M words and 2,500M words from text passages of English Wikipedia. The BookCorpus is a dataset consisting of 11,038 unpublished free books from 16 different genres.

BERT overcomes the unidirectionality constraint of previous models as OpenAI GPT by using a "Masked Language Model" (MLM) pre-training objective enabling the representation of both the left and the right context, which allows you to pre-train a deep bidirectional Transformer. In particular, this technique consists in replacing randomly 15% of the input tokens with [MASK]. Then, the BERT model is pre-trained in order to predict with cross entropy loss those masked tokens rather than reconstructing the entire input. As a result, the MLM objective is to predict the original id of the masked word

based only on its context.

However, while MLM allows a bidirectional pre-trained model to be obtained, it has the disadvantage of having a mismatch between pre-training and fine-tuning, as the token [MASK] does not appear during the fine-tuning. This problem was solved by not always replacing the masked words with the [MASK] token but by choosing only 15% of the tokens and if the i-th token is chosen, it is replaced with the [MASK] token 80% of the time, with a random token 10% of the time and in the remaining 10% of the time the i-th token remains unchanged.

BERT makes use also of "Next Sentence Prediction" (NSP) to jointly pre-trains text-pair representations. With this strategy, during the training process, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. Only 50% of the time the second sentence in the pair is actual the next sentence in the original document, while 50% of the time it is a random sentence from the corpus. The next sentence prediction task is useful in many downstream tasks such as question answering and natural language inference which are based on understanding the relationship between two sentences.

So with the BERT model, MLM and NSP are trained together during the unsupervised learning pre-training phase with the goal of minimizing the combined loss of the two strategies. Instead, fine-tuning is implemented as supervised learning and no masking or prediction of the next sentence takes place. As a result, fine-tuning is very fast and requires a relatively small number of samples.

As far as input representation concerned, before entering the model the input is processed following a specific procedure of 3 steps:

- A special classification token ([CLS]) is inserted at the beginning of the first sentence.
- In order to distinguish the sentence pairs that are packed together into a single sequence, it is used a special token separation ([SEP]) and it is also added to each token a sentence embedding indicating whether it belong to sentence A or sentence B.
- A positional embedding is added to each token to indicate its position in the sequence.

So, at the end, as shown in Figure 2.2, given a token, its input representation is composed by summing the corresponding token, the sentence and the position embeddings.

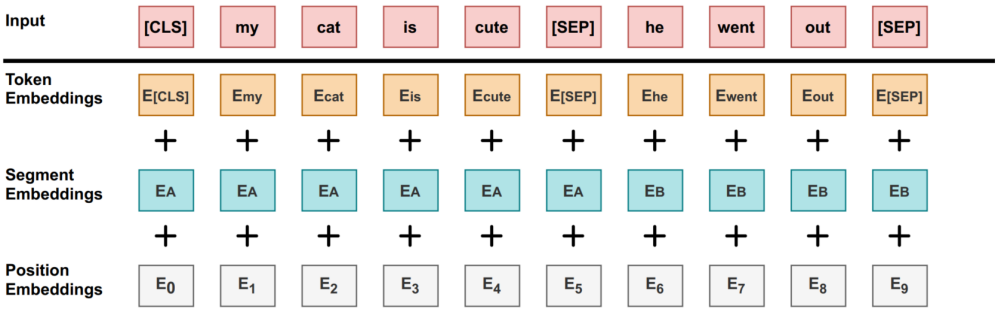


Figure 2.2: BERT input representation

Model Architecture

BERT's model architecture is a Transformer encoder stack based on the original Transformer implementation described in Transformer Model section.

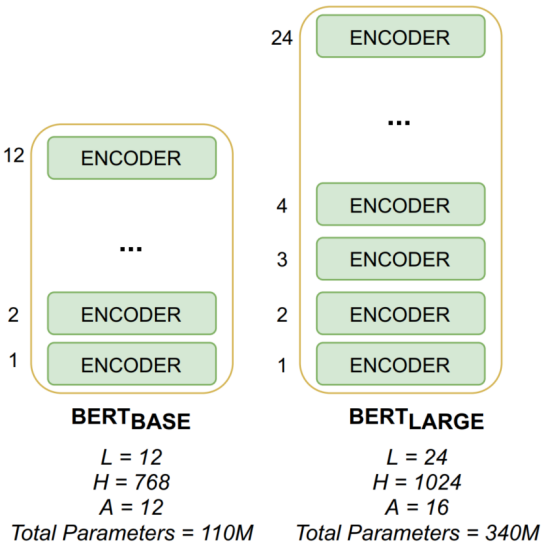


Figure 2.3: Model sizes for BERT

As showed in Figure 2.3, denoting the Transformer blocks, i.e. the number of layers as L, the hidden size as H and the number of self-attention heads as A, two BERT model sizes have been proposed.

Fine-Tuning BERT and experimental results

Applying BERT to a specific task is relatively straightforward. In fact, the model can be used for a wide variety of language tasks just plugging in the task-specific inputs and outputs into BERT and then fine-tune all the parameters [17]. For example, for classification tasks such as sentiment analysis, a layer on top of the transformer output is added for the [CLS] token like is done for Next Sentence classification.
Moreover, compared to pre-training, the fine-tuning procedure is relativity computationally inexpensive, in fact, the results obtained can be replicated, starting from the pre-trained model, in few hours using a GPU.

Fine-tuning BERT, the researcher team has achieved state-of-the-art results on different natural language tasks, in particular 11 NLP tasks are taken into account for the experiments. The average of results obtained on the GLUE[1] benchmark, a collection of

[1]General Language Understanding Evaluation

different natural language understanding tasks, are shown in the following Table 2.1.

The General Language Understanding Evaluation (GLUE) benchmark contains a variety of tasks covering textual entailment (RTE and MNLI), question-answer entailment (QNLI), paraphrase (MRPC), question paraphrase (QQP), textual similarity (STS), sentiment (SST) and linguistic acceptavility (CoLA).

System	**Average**	**GLUE score**
Pre-OpenAI SOTA	74.0	-
BiLSTM+ELMo+Attn	75.1	-
OpenAI GPT	75.1	72.5
$\text{BERT}_{\text{BASE}}$	79.6	78.3
$\text{BERT}_{\text{LARGE}}$	82.1	80.5

Table 2.1: GLUE Test results for BERT models. The "GLUE score" is taken from the evaluation server [16]. The "Average" score is different than the official GLUE score, since the BERT researchers exclude the WNLI set [17].

As shown in [17] the official paper, both $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$ outperforms previous systems, obtaining 4.5% and 7.0% respective average accuracy improvement over the prior state of the art (OpenAI GPT). These results were obtained by using a batch size of 32 and fine-tuning for 3 epochs over the data for all GLUE tasks. Specifically, for each task, they selected the best fine-tuning learning rate.

Different experiments were also done on the SQuAD v1.1 and v2.0[2], a collection of 100K crowd-sourced question/answer pairs. The results, taken from the paper [17], are shown in Table 2.2. We can observe that BERT pushes SQuAD 1.1 question answering Test F1 score to 93.2 and SQuAD v2.0 Test F1 score to 83.1.

System	**SQuAD 1.1 Test F1**	**SQuAD 2.0 Test F1**
$\text{BERT}_{\text{BASE}}$	-	-
$\text{BERT}_{\text{LARGE}}$	93.2	83.1

Table 2.2: SQuAD v1.1 and v2.0 Test results for BERT models. [17].

[2]Stanford Question Answering Dataset. SQuAD v2.0 task extends the first version by allowing for the possibility that no short answer exists in the provided paragraph

2.1.2 ELECTRA

Even if the BERT model achieves good results in different supervised learning tasks, the (MLM) objective that permits to "see" the text to both the left and right of the token during the prediction allowing the bidirectionality of the model itself, it has a limitation. In fact, the MLM disadvantage consists in predicting only a small subset, 15% of masked tokens, reducing the amount learned from each sentence. Instead of using MLM objective, with ELECTRA model [13] a new efficient pre-training task called "Replaced Token Detection" (RTD) was proposed and it looks to be more efficient as it considers every input token, rather than just a small number of masked tokens, and produces better results.

ELECTRA stands for *Efficiently Learning an Encoder that Classifies Token Replacement Accurately* [13]. Instead of masking the input, the RTD approach corrupts it by replacing some input token with plausible alternatives sampled from a small generator network. Moreover, in place of training a model that predicts original identities of the masked tokens as MLM does, a discriminative model is trained which predicts whatever each token in the corrupted input was replaced by a generator sample or not. So, the key benefit of this kind of discriminative task is that the model learns from all input tokens, making the computation more efficient.

As a consequence, given the same model size, data and compute, the contextual representations learned by this kind of approach outperform the ones learned by BERT [13].

Model Architecture

ELECTRA is a bidirectional encoder model made of two neural networks, a *generator* G and a *discriminator* D as showed in Figure 2.4. The generator can be any model that produces an output distribution over tokens, but usually it is used a MLM that is jointly trained with the discriminator. So, the G is trained to perform MLM and then learns to predict the original identities of the masked tokens. While, the D is trained to distinguish tokens in the data from tokens that have been replaced by generator samples [13].

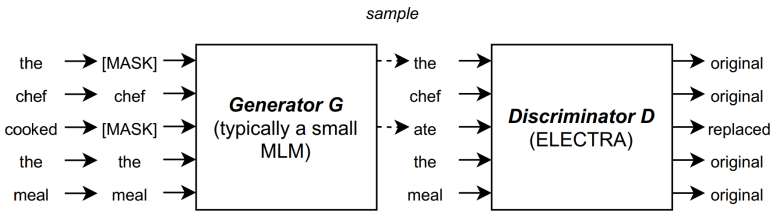


Figure 2.4: RTD in ELECTRA [13]

Although this training setup is similar to a Generative Adversarial Network (GAN), there are some differences. In fact, the generator is trained with the maximum likelihood rather than adversarially due to the difficulty of applying GANs to text. Moreover, if the generator generates the correct token, that token is considered "real" instead of "fake".

Fine-Tuning ELECTRA and experimental results

After pre-training, the generator is dropped and the discriminator (the ELECTRA model) is fine-tuned on downstream tasks. For experiments, as shown in Figure 2.5, three model sizes have been proposed and they are currently English-only: $\text{ELECTRA}_{\text{SMALL}}$, $\text{ELECTRA}_{\text{BASE}}$ and $\text{ELECTRA}_{\text{LARGE}}$.

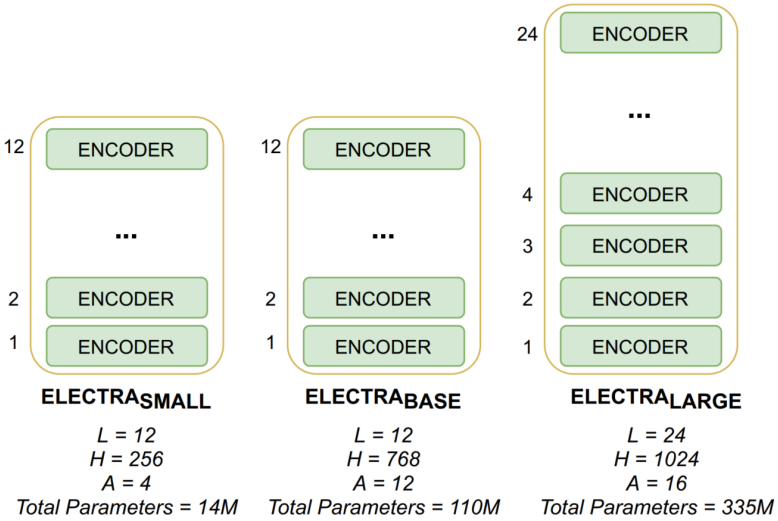


Figure 2.5: Model sizes for ELECTRA

The pre-training is done on the same data as BERT with the difference for large model which is pre-trained on a bigger dataset of about 33B tokens. For fine-tuning on GLUE, a simple linear classifier is added on top of ELECTRA.

ELECTRA outperforms the BERT model by 9 points on GLUE average score. Results obtained on GLUE benchmark are shown in Table 2.3.

System	Average	GLUE score
$\text{BERT}_{\text{LARGE}}$	79.8	80.5
$\text{ELECTRA}_{\text{SMALL}}$	79.7	77.4
$\text{ELECTRA}_{\text{BASE}}$	85.7	82.7
$\text{ELECTRA}_{\text{LARGE}}$	88.6	85.2

Table 2.3: GLUE Test results for ELECTRA models. The "Average" score excludes QNLI. (Sources [13] and [14]).

ELECTRA scores also better than MLM based methods on SQuAD. The model generally performs better at SQuAD 2.0 than 1.1 due to the fact that with RTD, distinguishing real tokens from plausible fakes, it can distinguish better answerable questions from fake unanswerable ones [13]. This model achieves a new state-of-the-art for a single model on the SQuAD 2.0 Q&A dataset. Results are shown in Table 2.4.

Another difference with respect to the BERT model in terms of performance concerns the loss calculation. In fact, loss in the discriminator model is defined over all input tokens instead of only masked tokens. The difference in terms of obtained score on GLUE benchmark was highlighted by the authors of the paper [13] who compared the original ELECTRA model with an ELECTRA 15% model which only calculates the discriminator loss over the masked tokens. It is shown that the original ELECTRA approach gets a 85.0 score while ELECTRA 15% gets 82.1, however better than BERT's score 82.2 (considering for all the model the base version). These results leads to affirm that ELECTRA's improvement can be due to learning from all tokens and to the reduction of the pre-train/fine-tune mismatch of MLM in BERT.

System	SQuAD 1.1 Dev	SQuAD 2.0 Dev	SQuAD 2.0 Test
$\text{BERT}_{\text{BASE}}$	88.5	-	-
$\text{BERT}_{\text{LARGE}}$	90.9	81.8	83.1
$\text{ELECTRA}_{\text{BASE}}$	80.8	83.3	-
$\text{ELECTRA}_{\text{LARGE}}$	94.9	90.6	91.4

Table 2.4: SQuAD v1.1 and v2.0 Dev and Test F1 score results for ELECTRA models. [13].

2.2 Transformer unidirectional decoder models

Unlike the previously analyzed encoder-based models, the Transformer unidirectional decoder models are auto-regressive. Typically, these models are pre-trained on the classic language modeling task in which the key is to predict the next token having read all the previous ones. They are based on the decoder part of the original Transformer architecture and they use an attention mask on top of the full sentence in order to look only what was before in the attention heads.

2.2.1 GPT-2

Successor of GPT, *OpenAI Generative Pre-Trained 2* (GPT-2) is a casual unidirectional transformer-based language model with 1.5B parameters pre-trained on a very large corpus of 40GB of Internet text called WebText [20]. The WebText dataset contains the text subset of 45 million links taken from Reddit, a social media platform [21]. Particularly, all Wikipedia documents were removed from this dataset in order to avoid overlap of training data with test evaluation task, as these are common data source for other datasets.
Proposed in 2019 in *Language Models are Unsupervised Multitask Learners* [21], its main purpose is to predict the next word, given all of the previous words within some text. With respect to his predecessor, GPT-2 has more than 10x the parameters and it is trained on more than 10X the amount of data.

GPT-2 is able to automatically generate texts with a credible, consistent and very similar style to that processed by humans. For this reason, that model finds its maximum use in the Natural Language Generation (NLG) field for the texts generations rather than for text classification areas as it will also highlight in the experiments carried out later in this Master Thesis.

Model Architecture

Unlike BERT and ELECTRA, GPT-2's model architecture is a Transformer decoder stack. Another difference with the previously explained models is that GPT-2 outputs one token at time, then that token is added to the sequence of inputs which becomes the input to the model in its next step. This is the functional mechanism of auto-regression models. GPT-2 has an architecture like that of the first GPT version but with a normalization layer to the input of each sub-block and after the final self-attention layer.

Being a decoder-only blocks model, in GPT-2 there is also a different self-attention mechanism with respect to BERT, in fact, the model doesn't change the word to [MASK] like BERT but it uses a masked self-attention preventing a position to peak at tokens to its right. So, this kind of self-attention mechanism blocks information from tokens that are to the right of the position being calculated. Each decoder block is therefore composed by masked self-attention layer and by a feed forward layer.

Four different size variants of OpenAI GPT-2 has been developed depending on the number of transformer decoder blocks staked up (as shown in Figure 2.6).

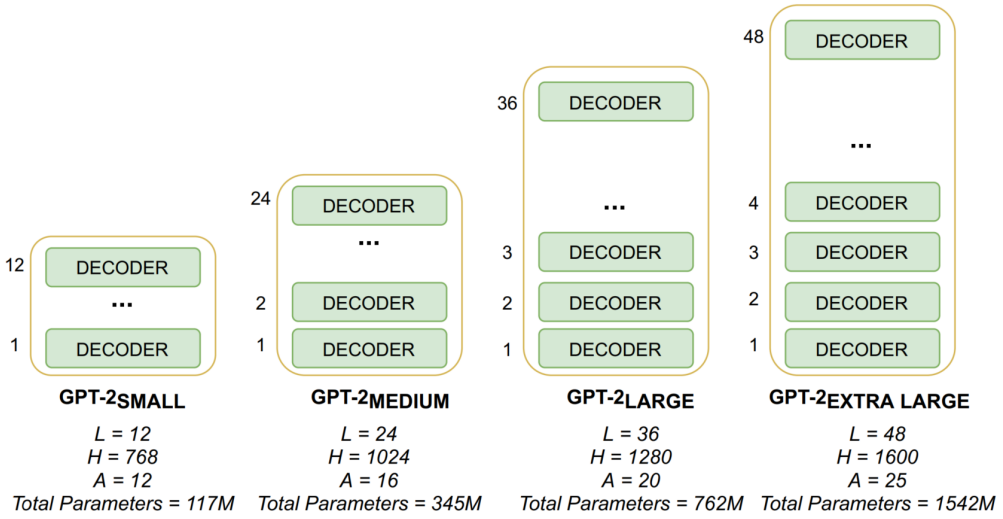


Figure 2.6: Model sizes for GPT-2

Fine-Tuning GPT-2 and experimental results

Unlike the BERT model that learns the sentence separator [SEP], the classifier token [CLS] and sentence A/B embeddings during the pre-training, GPT-2 uses the [SEP] and [CLS] tokens only during the fine-tuning phase. Another difference with respect to the BERT model is that, GPT-2 uses the same learning rate of 5e-5 for all fine-tuning experiments, while BERT chooses a task-specific fine-tuning learning rate which obtains the best result on the development set.

GPT-2 basically combines the unsupervised pre-training and the supervised fine-tuning approaches in a multi-task manner. Like any other auto-regressive model, it maximizes the joint probability but taking into account the multi-task learning. As a consequence, since the system should be able to perform many different tasks, the model objective is to predict the output with probability $p(output|inputs, task)$ given the input conditionally on a certain task to be performed in addition to the input data.

In order to train the model for different tasks such as translation, summarization and Q&A, GPT-2 proposes to take data in a particular formats and then just training the model for a text completion objective. Moreover, since the model is not at all trained explicitly for any of the previously mentioned tasks, OpenAI researchers associate the GPT-2 model to a "zero-shot" task transfer [21], as a consequence, the model is only evaluated on a variety of domain-specific language modeling tasks as a final test [20].

As far as input representation concerned, GPT-2 uses the Byte Pair Encoding (BPE), a way of splitting up words to apply tokenization that is a compromise between character and word level language modeling. Using the byte sequence representation, the model is able to assign a probability to any Unicode string, allowing the model to be evaluated on any dataset regardless of any pre-processing steps, tokenization or vocabulary size.

As for experimental results, GPT-2 achieves state of the art results on 7 out 8 language modeling datasets in a zero-shot setting [21]. Examples of such datasets are the LAMBADA dataset, which task is to predict the final word of sentence which require at least 50 tokens of context for a human to successfully predict and the Children's Book Test (CBT) dataset, created to examine the performance of language modelings on different categories of words such as named entities, nouns, verbs and prepositions. As reported in Table 2.5, on the first dataset mentioned above GPT-2 improves by 4% the accuracy and, on the second one, it achieves new state of the art results of 93.3% on common nouns (CBT-CN) and 89.1% on named entities (CBT-NE).

System	**LAMBADA** (ACC)	**CBT-CN** (ACC)	**CBT-NE** (ACC)
SOTA	59.23	85.7	82.3
GPT-2_{SMALL}	45.99	87.65	83.4
GPT-2_{MEDIUM}	55.48	92.35	87.1
GPT-2_{LARGE}	60.12	93.45	88.0
GPT-$2_{\text{EXTRA-LARGE}}$	63.24	93.30	89.05

Table 2.5: GPT-2 zero-shot results on many datasets.[21]

Chapter 3

Environment and libraries used

In this section all the technologies that have been employed during the comparative analysis of different transformer models fine-tuned for text classification task will be described. In particular, all the libraries used are available as Python packages.

3.1 Pytorch and HuggingFace

The pre-trained models used such as BERT, ELECTRA and GPT-2 are taken from the Transformers library [10] made available by the Hugging Face Hub [25]. Transformers library provides general-purpose architectures for NLU and NLG with different pre-trained models in many languages. This API is fully integrated with one of the most popular deep learning libraries like the Pytorch framework [8] which has been chosen for the development of the project, indispensable for working with the Tensor class (`torch.Tensor`) on a CUDA-capable Nvidia GPU.

Figure 3.1: Pytorch logo [11]

Figure 3.2: Hugging Face logo [25]

3.2 Kaggle and Google Colab

In order to take advantage of the CUDA cores that are specialized cores, Google Colaboratory [4] and Kaggle [7] have been used for accelerating the deep learning operations with free GPU. The former provides a free NVIDIA TESLA K80 GPU, while the latter provides free access to NVIDIA TESLA P100 GPU (named Kaggle GPU in the next paragraphs).

Part II

Experimental Results

Chapter 4

Models application on NLP text classification tasks

As far as text classification concerned, in the following analyses two problems will be considered: Multi-Label and Multi-Class text classification.

4.1 Multi-Label Text Classification

A multi-label classification is characterized by the fact that each text can be simultaneously and independently assigned to multiple labels or classes. Therefore, in a multi-label classification problem, the dataset is composed of instances and each of them can be assigned with multiple categories that are represented as a set of target labels and the task is to predict the label set.

4.1.1 Dataset (English): GoEmotions Dataset

For fine-tuning the pre-trained models on the Multi-Label classification task, it has been chosen the *GoEmotions* dataset as an English corpus developed at Google Research [6]. It is the largest human annotated English dataset of 58k selected comments extracted from Reddit, labeled for 27 emotion categories or Neutral [15]. In particular, the emotion categories are: admiration, amusement, anger, annoyance, approval, caring, confusion, curiosity, desire, disappointment, disapproval, disgust, embarrassment, excitement, fear, gratitude, grief, joy, love, nervousness, optimism, pride, realization, relief, remorse, sadness, surprise.

For evaluating the transformer-based models these emotions are mapped into Ekman's taxonomy plus Neutral. The Ekman's taxonomy includes the following categories: anger, disgust, fear, joy, sadness and surprise. As a consequence, the models will be trained and fine-tuned to predict for each comment as a maximum a number of labels equals to `NUM_LABELS = 7`

The dataset is split into training, test and validation set. Their size is respectively 43.410, 5.427 and 5.426 comments. The distribution of labels in each split set is shown in Table 4.1. We can therefore observe a greater number of comments labeled as "Joy" and "Neutral" in all three sets.

Dataset	**Anger**	**Disgust**	**Fear**	**Joy**	**Sadness**	**Surprise**	**Neutral**
Train	5579	793	726	17410	3263	5367	14219
Dev	717	97	105	2219	390	624	1766
Test	726	123	98	2104	379	677	1787

Table 4.1: GoEmotions labels distributions Train/Dev/Test datasets

The Google researchers present a strong baseline for emotion prediction model for GoEmotions. They use the BERT model in the experiments adding a dense output layer on top of the pre-trained model for fine-tuning purposes with a sigmoid cross entropy loss function for the multi-label classification task. As hyperparameters, they use a batch size of 16, a learning rate of 5e-5 for training and a dropout probability of 0.7. They also find that training the model for 4 epochs is necessary for learning data, while more epochs lead to overfitting . Their best performance achieves an average F1-score of 0.64 as shown in Table 4.2. These results will be subsequently compared with the results obtained during this Thesis fine-tuning different models.

Ekman Emotion	**Precision**	**Recall**	**F1**
anger	0.50	0.65	0.57
disgust	0.52	0.53	0.53
fear	0.61	0.76	0.68
joy	0.77	0.88	0.82
neutral	0.66	0.67	0.66
sadness	0.56	0.62	0.59
surprise	0.53	0.70	0.61
macro-average	**0.59**	**0.69**	**0.64**
std	0.10	0.11	0.10

Table 4.2: GoEmotions baseline results using Ekman's taxonomy on test set [15]

4.1.2 Dataset (Italian): AMI Dataset

In order to evaluate the performance of models such as BERT, ELECTRA and GPT-2, that have been trained on an English Corpus, an Italian dataset was also chosen for the analysis carried out. For this purpose, the dataset used for the Automatic Misogyny Identification (AMI) shared task has been chosen. This task was proposed at the Evalita [1] 2020 evaluation campaign and it is aimed at automatically identifying misogynous content in Italian Twitter [3].

Going into more details, the model has to recognize if a text is misogynous or not, and in case of misogyny, if it expresses an aggressive attitude. As a consequence, the number of labels that the models will be able to predict is equal to `NUM_LABELS = 2`.

The dataset is taken from the European Language Grid catalogue [2]. It consists in a training set of 5.000 tweets and in a test set of 1.000 tweets. In particular, in my experiments, I will consider 20% of training set as validation set. The distribution of labels of the three sets is shown in Table 4.3.

Dataset	Misogynous	Aggressiveness
Train	1880	1443
Dev	478	381
Test	500	176

Table 4.3: AMI dataset labels distribution Train/Dev/Test datasets

As the metric evaluation, each label to be predicted (i.e. "Misogyny" and "Aggressiveness") has been evaluated independently on the other using a Macro F1-Score. In the Figure 4.1, all the results obtained in the competition and also the AMI baseline are shown. The score is based on the Average Macro F1-score, computed as follows:

$$Average\_F1\_score = \frac{F_1(Misogyny) + F_1(Aggressiveness)}{2}$$

[1] EVALITA is a periodic evaluation campaign of NLP and speech tools for the Italian Language http://www.evalita.it/

Rank	Run Type	Score	Team
**	c	0.744	UniBO **
1	u	0.741	jigsaw
2	u	0.738	jigsaw
3	c	0.734	fabsam
4	u	0.731	YNU_OXZ
5	c	0.731	fabsam
6	c	0.717	NoPlaceForHateSpeech
7	u	0.701	YNU_OXZ
8	c	0.695	fabsam
9	c	0.693	NoPlaceForHateSpeech
10	c	0.687	AMI_the_winner
11	u	0.684	MDD
12	c	0.683	PoliTeam
13	c	0.682	MDD
14	c	0.681	PoliTeam
15	u	0.668	MDD
16	c	0.665	AMI_the_winner
17	c	0.665	AMI_BASELINE
18	c	0.647	PoliTeam
19	c	0.634	UniBO
20	c	0.626	AMI_the_winner
21	c	0.490	NoPlaceForHateSpeech

Figure 4.1: Results with baseline for AMI task [18]

4.1.3 Fine-Tuning Models

The purpose is to build a model that will be able to determine different types of emotions in a given text snippet for the first dataset (GoEmotions) and for the second one (AMI) to determine if a tweet is misogynous and aggressive or not. Different models such as BERT English/Multilingual, ELECTRA and GPT-2 will be considered. These models, described in details in Table 4.4, are downloaded from the HuggingFace Transformers library [10].

Architecture	Model id	Details of the model
BERT	`bert-base-uncased`	12-layer, 768-hidden, 12-heads, 110M parameters. Trained on lower-cased English text.
	`bert-base-multilingual-cased`	12-layer, 768-hidden, 12-heads, 179M parameters. Trained on cased text in the top 104 languages with the largest Wikipedias.
GPT-2	`gpt2`	12-layer, 768-hidden, 12-heads, 117M parameters. OpenAI GPT-2 English Model.
ELECTRA	`google/electra-base-discriminator`	12-layer, 768-hidden, 12-heads, 110M parameters. Trained on lower-cased English text.

Table 4.4: Pre-trained models from HuggingFace library

Data preparation and Tokenisation

The data that will be fed to the model are represented, after some pre-processing, by a dataframe in the following format: text and labels (as one-hot encoding vector). An example is shown in Figure 4.5 for both datasets used.

The data will be represented by class `CustomDataset` which is defined to accept the `tokenizer`, the `dataframe` and `max_length` as input and then it generates tokenized output and tags that are used by the transformer models for training. The `tokenizer` perform tokenization over the `text` column of the dataframe, which involves breaking up of input text into its individual words, and generates the necessary outputs, namely: `input_ids`, `attention_mask`, `token_type_ids`. `labels` is instead the list of classes labelled as 0 or 1 in the dataframe. In order to perform tokenization, the tokenizer included with the transformer models will be downloaded such as `BertTokenizer` for BERT, `ElectraTokenizer` for ELECTRA and `GPT2Tokenizer` for GPT-2. The `CustomDataset` class is used to create 3 datasets: Training, Validation and Test dataset. The former is used to fine-tune the model, whereas the latter two are used to evaluate the performance of the model.

The PyTorch `Dataloader` class is used to create the training, validation and testing dataloader that load data to the neural network in a defined manner. This is needed

Dataset	Data Format	
	text	labels
GoEmotions	... to make her feel threatened Enjoy the ride! I'm really sorry about your situation ...	... [0,0,1,0,0,0,0] [0,0,0,1,0,0,0] [0,0,0,0,1,0,0] ...
AMI	... No non mi sento meglio, ho solo tanta tristezza, perché ironizzare anche sullo stupro CHE DIAMINE È UNA COSA SERIA è assurdo! ...	... [0,0] ...

Table 4.5: Data examples for GoEmotions and AMI datasets

because all the data from the dataset cannot be loaded to the memory at once, hence the amount of data loaded to the memory and then passed to the neural network needs to be controlled through some parameters such as the `batch_size` and `max_len`.

Configuration section and Hyper-parameters

In order to fine-tune the models, some key variables have been defined which represent the configurations and hyperparameters. `MAX_LEN` represents the maximum length used in the tokenization, `TRAIN_BATCH_SIZE` and `VALID_BATCH_SIZE` are the size of the batches used in the dataloaders, `EPOCHS` is the number of training epochs defined for fine-tuning the models on the specific NLP task, hence it defines how many times the complete data will be passed through the network. `LEARNING_RATE`, `EPS` and `WEIGHT_DECAY` are used in the AdamW optimizer and `WARMUP_FRACTION` is used to calculate the warmup steps for the scheduler in order to update the model parameters and learning rate during the training.

Model Architecture

In order to train the model on the chosen datasets, we consider a modified pre-trained model (for Bert, Electra and GPT-2) in order to give outputs for classification, that is respectively `BertForSequenceClassification` for Bert model, `ElectraForSequenceClassification` for Electra model and `GPT2ForSequenceClassification` for GPT-2. Both of these models derive from the specific base model with a sequence classification head on top, hence a linear layer on top of the pooled output.

As a way to fine-tune the network, during the training operation, when the model is put in train mode (`model.train()`), the dataloader passes the training dataset to the model in batches, then subsequent output from the model and the actual target are compared to calculate the loss which is used to optimize the weights of the neurons in the network during backpropagation. On the other hand, during the validation stage, in which we pass the validation dataset to the model, the weights of the model are not updated, hence only

the final output is compared with the expected one in order to calculate the accuracy of the model.

4.1.4 Loss Function and Optimizer

In the scenario of a multi-label classification problem, the usage of Binary Cross-Entropy with logits (`BCEWithLogitsLoss` from the PyTorch package `torch.nn` [1]) is preferred as the loss function, which allows the model to independently assign the probabilities to the labels optimizing the model. This function, which combines a Sigmoid layer and the BCELoss in one single class, takes the output of the dense layer with no activation as input and applies the Sigmoid function to improve numerical stability. The Sigmoid mathematical function, reported below 4.1, squishes any real number into a range between 0 and 1 in order to treat model predictions like probabilities.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{4.1}$$

The BCELoss is defined considering the binary cross entropy that is a measure of the difference between two probability distribution p and q. In particular, the BCE function is the negative average of the log of corrected predicted probabilities as defined in 4.2 and it is strictly related to the entropy concept which is a measure of the uncertainty associated with a given distribution. Considering C as the number of classes, the true probability p_c is the true label, whereas the given distribution q_c is the predicted value of the current model.

$$H(p, q) = -\sum_{c=1}^{C} p_c \cdot log(q_c) \tag{4.2}$$

During the training, the classifier uses each of the N samples in its training set to compute the cross-entropy loss, as a consequence, since the probability of each sample is $1/N$, the loss function is given by the following equation 4.3.

$$J(p, q) = \frac{1}{N} \sum_{N=1}^{N} H(p, q) \tag{4.3}$$

During the training phase, in order to update the model parameters, the `AdamW` optimizer has been used. The AdamW, proposed in the paper [19], is an adaptive learning rate optimization algorithm that modifies the typical implementation of weight decay in Adam, by decoupling weight decay from the gradient update. To overcomes the problem that adaptive gradient methods do not generalise as well as the stochastic gradient descent (SGD) with momentum when tested on a different set of deep learning tasks, the AdamW optimizer instead of using the classic L2 normalization, it simply uses a weight decay fix. Experimentally, it has been shown that AdamW yields better training loss and that the models generalize much better than models trained with Adam allowing the new version to compete with SGD with momentum.

4.1.5 Evaluation Metrics and Results

After the completion of each training epoch in which the model makes predictions on the training dataset fed to it in batches, and backpropagtes the prediction error to adjust the model's parameter, the model is then evaluated measuring the performance on the validation ans test sets and for this reason it is put in evaluation mode (`model.eval()`). During the forward pass, the model, in addition to calculating the loss, gets also the "logits" output. A logit, called also a score, is a raw unscaled value associated with a label, before applying an activation function like the sigmoid or the softmax function. Since we are considering now the multi-label classification problem, the sigmoid function will be used as the activation function to get probabilities from predictions output by the model.

In order to evaluate the fine-tuned models and to get a measure of the models' performance, the accuracy and F1-score are computed. The accuracy is the proportion of correctly classified samples out of all the samples and it is calculated with the `accuracy_score` function from the `sklearn.metrics` module of the scikit-learn API. Another measure of the model's accuracy is the F1-score which is a way of combining the precision and recall of the model, and it is defined as the harmonic mean of the model's precision and recall. In particular, the precision (P), called also the Positive Predicted Value (PPV), is the ratio of relevant instances among the retrieved ones. So, it is a measure of the accuracy of the positive prediction and it is defined as

$$P = \frac{TP}{(TP + FP)} \tag{4.4}$$

where TP is the number of true positives and FP the number of false positives. Whereas, the recall (R), called also sensitivity or True Positive Rate (TPR), is the ratio of positive instances correctly detected by the classifier and it is defined as

$$R = \frac{TP}{(TP + FN)} \tag{4.5}$$

where FN is the number of false negatives.
Given the P and R, the F1-score is defined as

$$F_1 = 2 \cdot \frac{P \cdot R}{(P + R)} = \frac{2TP}{(2TP + FP + FN)} \tag{4.6}$$

and it reaches its best value at 1 and worst score at 0. All of these metrics have been obtained through the `classification_report` function from the `sklearn.metrics` module. In addition to calculating the metrics mentioned for each class, this function also computes the "micro" and "macro" values. The macro-averaged F1-score, or the Macro-F1 for short, is computed as an arithmetic mean of the per-class F1-scores and this does not take label imbalance into account, taking all classes as equally important. While, the Micro-F1 considers all the samples together, computing the metrics globally by counting the total

TP, FN and FP.

Each pre-trained model considered has been trained for a different number of epochs and subsequently evaluated based on the previously discussed metrics.

Results on GoEmotions Dataset

Let's consider first the GoEmotions Dataset. For each model analyzed, it was selected as the number of epochs the one with which the best performances were obtained. In particular, 2 epochs were chosen for the English Bert model, 3 epochs for the Multilingual Bert and the Electra models and finally 5 epochs for the GPT-2 model. It has been shown that training for more epochs results in overfitting, hence the training data are modelled too well that the model can not generalise correctly. These values together with the remaining selected hyperparameters are shown in Table 4.6.

Hyperparameters	**Bert**	**Bert Multilingual**	**Electra**	**GPT-2**
Epochs	2	3	3	5
Batch Size	16	16	16	16
Learning Rate	2e-5	2e-5	2e-5	2e-5
Adam eps	1e-6	1e-6	1e-6	1e-6
Weight Decay	0.01	0.01	0.01	0.01
Warmup fraction	0.1	0.1	0.1	0.1
Machine	Kaggle GPU	Kaggle GPU	Kaggle GPU	Kaggle GPU
Training and Validation Time	0:23:24	0:35:13	0:35:02	1:01:35

Table 4.6: Hyperparameters selected for the analyzed models with GoEmotions Dataset

The performances obtained on the dev and test set are shown in Table 4.7 and 4.8 respectively. Looking at the results achieved, it can be seen that all the models obtain a good accuracy of correctly classifying the input text. Table 4.8 shows that the transformers pre-trained on large corpus generally achieve a good F1-score in the classification task. The exception is GPT-2, which scores under BERT and ELECTRA. GPT-2 is the model producing the lowest F1-score, which demonstrates that the greatest potential of this model is evident in the text generation task rather than in the classification. In fact, its main ability is to generate coherent and good text from minimal prompts.

Moreover, it can also be seen that, as expected, the Electra model slightly outperforms the Bert model. This is partly due, as illustrated in the previous chapters, to the different type of objectives with which the models in question were pre-trained, i.e. masked language model (MLM) for Bert and replaced token detection (RTD) for the Electra model.

Model	**Accuracy (%)**	**Micro-F1 (%)**	**Macro-F1 (%)**
Bert	62.67	70.23	63.01
Bert-Multilingual	61.77	69.41	62.40
Electra	62.49	70.45	63.26
GPT-2	59.10	67.76	59.25

Table 4.7: Results on dev set using Ekman's taxonomy (GoEmotions Dataset)

Model	**Accuracy (%)**	**Micro-F1 (%)**	**Macro-F1 (%)**
Bert	62.07	70.01	62.30
Bert-Multilingual	61.54	69.30	63.09
Electra	62.31	70.08	63.33
GPT-2	57.20	66.75	59.02

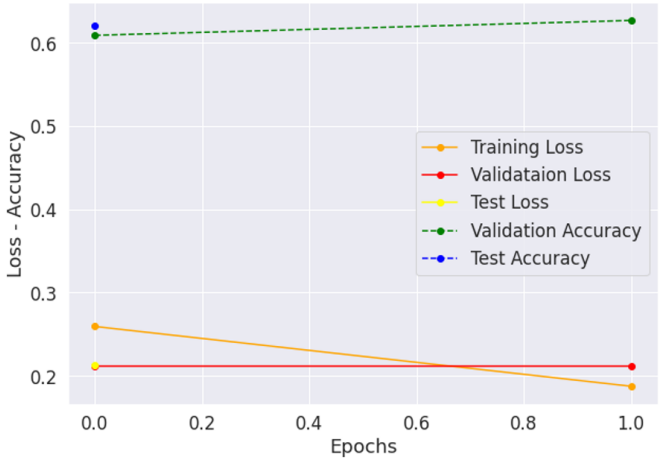
Table 4.8: Results on test set using Ekman's taxonomy (GoEmotions Dataset)

Going into more details, it is possible to observe from the `classification_report` (Table 4.13) how the measures achieved with the various models on the test set (i.e. **0.62** for Bert-English, **0.63** for Bert-Multilingual and for Electra, **0.59** for GPT-2) are in line with the baseline obtained by the GoEmotions team [15] whose macro-average F1-score for the Ekman's taxonomy is **0.64** (as illustrated in Table 4.2 in Section Dataset (English): GoEmotions Dataset).
Furthermore, the models obtain the best performance on higher frequency emotions such as the "joy" emotion with a F1-score always greater than **0.80**.
Instead, all the models have a low recall value for the "disgust" emotion which means that they are able to correctly classify only 30%-40% of comments labeled as disgusting.

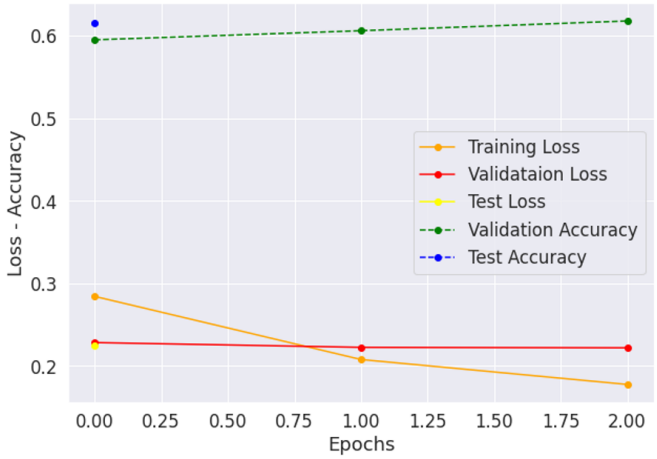
Let's consider now the iterative process of the training model. As mentioned previously, in each epoch the model makes a guess about the output, calculating the error in its guess which takes the name of loss value. As a consequence, the main objective of the loss function is to measure the degree of dissimilarity of obtained results with respect to the target value by minimizing this loss value during the training phase. As can be seen from the plots in the Figure 4.2, the expected behaviours have been obtained as results. In fact, the training and validation loss decreases with each epoch, while the validation accuracy increases with each epochs. The loss values obtained during the different fine-tuning phases for the analyzed model are also reported in the Table 4.9.

Model	Training Loss	Validation Loss	Test Loss
Bert	0.20	0.21	0.21
Bert-Multilingual	0.18	0.22	0.22
Electra	0.17	0.22	0.22
GPT-2	0.21	0.22	0.23

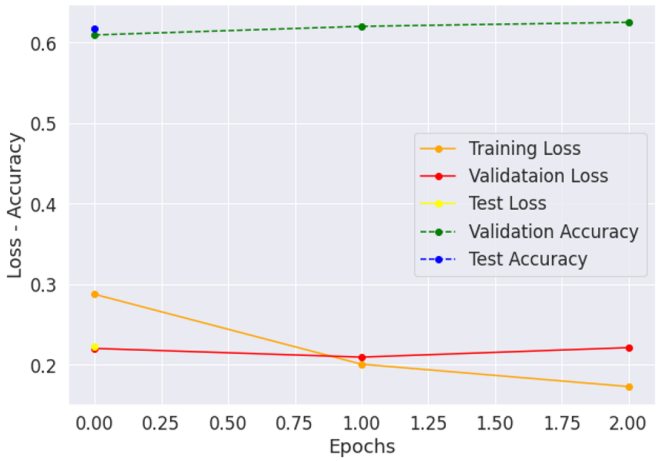
Table 4.9: Loss function values (GoEmotions Dataset)



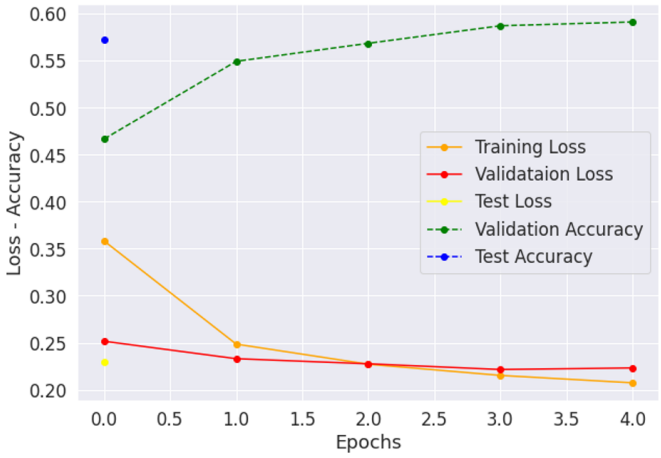
(a) Bert-English Model (2 Epochs)



(b) Bert-Multilingual Model (3 Epochs)



(c) Electra Model (3 Epochs)



(d) GPT-2 Model (5 Epochs)

Figure 4.2: Loss and accuracy plots for the models across analyzed epochs

Label	Precision	Recall	F1-score	Support
anger	0.60	0.52	0.56	726
disgust	0.64	0.34	0.44	123
fear	0.69	0.62	0.65	98
joy	0.83	0.85	0.84	2104
sadness	0.68	0.54	0.60	379
surprise	0.62	0.61	0.62	677
neutral	0.74	0.58	0.65	1787
macro-average	**0.69**	**0.58**	**0.62**	5894

Table 4.10: Results on Test set using Bert-English Model (GoEmotions Dataset)

Label	Precision	Recall	F1-score	Support
anger	0.59	0.49	0.54	726
disgust	0.66	0.42	0.51	123
fear	0.71	0.62	0.66	98
joy	0.83	0.83	0.83	2104
sadness	0.68	0.56	0.61	379
surprise	0.62	0.61	0.62	677
neutral	0.70	0.60	0.64	1787
macro-average	**0.68**	**0.59**	**0.63**	5894

Table 4.11: Results on Test set using Bert-Multilingual Model (GoEmotions Dataset)

Label	Precision	Recall	F1-score	Support
anger	0.59	0.55	0.57	726
disgust	0.56	0.36	0.44	123
fear	0.69	0.72	0.71	98
joy	0.81	0.86	0.84	2104
sadness	0.68	0.57	0.62	379
surprise	0.60	0.62	0.61	677
neutral	0.74	0.58	0.65	1787
macro-average	**0.67**	**0.61**	**0.63**	5894

Table 4.12: Results on Test set using Electra Model (GoEmotions Dataset)

Label	Precision	Recall	F1-score	Support
anger	0.57	0.45	0.51	726
disgust	0.54	0.36	0.43	123
fear	0.63	0.56	0.59	98
joy	0.81	0.82	0.81	2104
sadness	0.66	0.54	0.59	379
surprise	0.62	0.56	0.59	677
neutral	0.73	0.51	0.60	1787
macro-average	**0.65**	**0.54**	**0.59**	5894

Table 4.13: Results on Test set using GPT-2 Model (GoEmotions Dataset)

Results on AMI Dataset

Let's consider now the AMI Dataset. For each model analyzed, in Table 4.14 are reported the hyperparameters chosen in order to achieve the best performances. Unlike the previous dataset taken into account (GoEmotions), this dataset on the Automatic Misogyny Identification is small size, therefore, based on the available resources, it was possible to train the model for a greater number of epochs in order to achieve better performances.

Hyperparameters	**Bert**	**Bert Multilingual**	**Electra**	**GPT-2**
Epochs	8	8	8	3
Batch Size	16	16	16	16
Learning Rate	2e-5	2e-5	2e-5	2e-5
Adam eps	1e-6	1e-6	1e-6	1e-6
Weight Decay	0.01	0.01	0.01	0.01
Warmup fraction	0.1	0.1	0.1	0.1
Machine	Kaggle GPU	Kaggle GPU	Kaggle GPU	Kaggle GPU
Training and Validation Time	0:08:53	0:09:05	0:08:55	0:03:33

Table 4.14: Hyperparameters selected for the analyzed models with AMI Dataset

The performances obtained on the dev and test set are shown in Table 4.15 and 4.16 respectively. Evaluating the results obtained, it is clear how all the models reach a high accuracy in predicting the labels of the tweets provided in input to each model. In particular, as expected, Bert's model trained on a multilingual corpus (Bert-Multilingual) including the Italian language, obtains a macro-F1 score close to 100% on the dev set. Moreover, considering the test set, the Bert-Multilingual model improves the macro-F1 score of more or less 7% points with respect to the Bert-English model. This result can be explained by the fact that the multilingual model was pre-trained on a larger corpus of data containing also the Italian language data. Consequently, it is able to classify a larger number of Italian terms or with higher accuracy then a model that has only been pre-trained on an English corpus.

Model	**Accuracy (%)**	**Micro-F1 (%)**	**Macro-F1 (%)**
Bert	96.30	97.56	97.42
Bert-Multilingual	98.50	99.07	99.02
Electra	94.00	95.97	95.81
GPT-2	70.60	74.32	73.72

Table 4.15: Results on dev set (AMI Dataset)

Model	Accuracy (%)	Micro-F1 (%)	Macro-F1 (%)
Bert	44.10	61.54	57.41
Bert-Multilingual	55.70	68.63	64.05
Electra	53.50	66.21	62.30
GPT-2	50.00	60.87	56.92

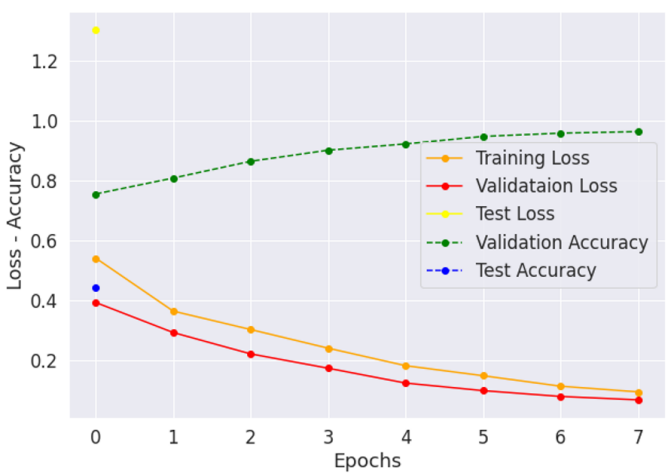
Table 4.16: Results on test set (AMI Dataset)

Comparing the values obtained (Table 4.21) with those present in the paper for the AMI task challenge (as shown in Figure 4.1 in Section Dataset (Italian): AMI Dataset), the macro-average F1-score obtained with the different transformer models considered is almost always higher than 60%, except in the case of the GPT-2 model which, as in the previous dataset, it appears to be less performing.
Furthermore, the models are able to predict more correctly the tweets present with a greater frequency, as in this case the tweets labeled as "misogynous", achieving in most cases a F1-score greater than **0.70**.

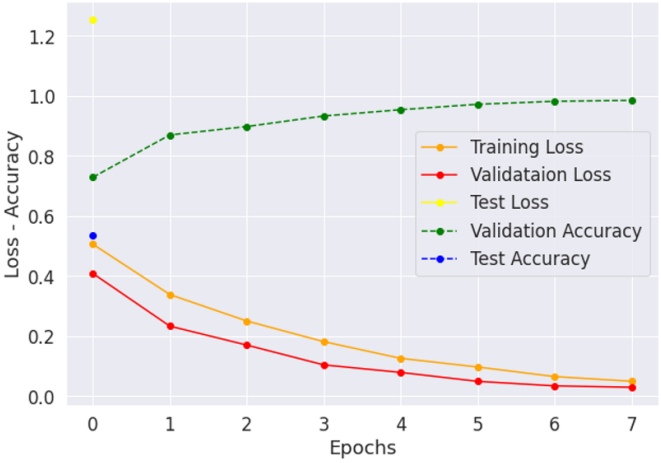
As for the loss function, the loss values obtained during the different fine-tuning phases for the analyzed model are reported in the Table 4.17 and shown with the corresponding graphs in the Figure 4.3. As can be seen from the plots, the models pay too much attention to training data and do not generalize well to the test dataset. As a result, the variance, which reflects the variability of the model prediction, is found to be high and as a consequence, these models show very low error on the training and validation set but an high error on the test set.

Model	Training Loss	Validation Loss	Test Loss
Bert	0.09	0.06	1.30
Bert-Multilingual	0.05	0.02	1.20
Electra	0.13	0.10	0.90
GPT-2	0.50	0.45	0.59

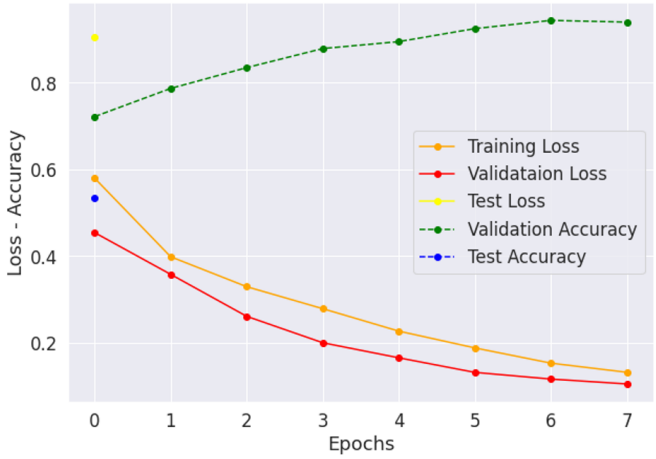
Table 4.17: Loss function values (AMI Dataset)



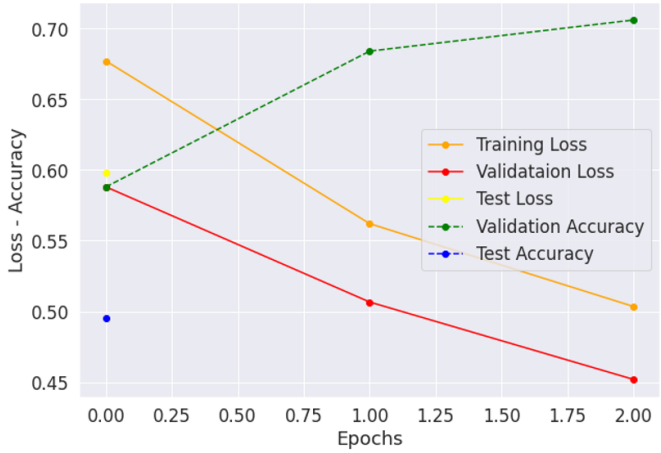
(a) Bert-English Model (8 Epochs)



(b) Bert-Multilingual Model (8 Epochs)



(c) Electra Model (8 Epochs)



(d) GPT-2 Model (3 Epochs)

Figure 4.3: Loss and accuracy plots for the models across analyzed epochs

Label	Precision	Recall	F1-score	Support
misogynous	0.60	0.92	0.73	500
aggressiveness	0.27	0.86	0.42	176
macro-average	**0.44**	**0.89**	**0.57**	676

Table 4.18: Results on Test set using Bert-English Model (AMI Dataset)

Label	Precision	Recall	F1-score	Support
misogynous	0.68	0.92	0.78	500
aggressiveness	0.33	0.86	0.48	176
macro-average	**0.51**	**0.89**	**0.63**	676

Table 4.19: Results on Test set using Bert-Multilingual Model (AMI Dataset)

Label	Precision	Recall	F1-score	Support
misogynous	0.66	0.87	0.75	500
aggressiveness	0.34	0.88	0.50	176
macro-average	**0.50**	**0.87**	**0.62**	676

Table 4.20: Results on Test set using Electra Model (AMI Dataset)

Label	Precision	Recall	F1-score	Support
misogynous	0.68	0.74	0.71	500
aggressiveness	0.30	0.72	0.42	176
macro-average	**0.49**	**0.73**	**0.57**	676

Table 4.21: Results on Test set using GPT-2 Model (AMI Dataset)

4.2 Multi-Class Text Classification

Unlike the previous task analyzed, in a multi-class classification each sample is assigned to exactly only one label or class and it cannot belong to one or more than one class as it happens in multi-label classification.

4.2.1 Dataset (English): Covid19 Tweets Dataset

For fine-tuning the pre-trained models on the Multi-Class classification task it has been chosen the *Coronavirus tweets NLP* dataset as an English dataset. The dataset is made up of English tweets relating to Covid19 and it is available in the Kaggle dataset catalog [5]. The purpose of this task is to predict the sentiment related to the tweet. There are five classes (`NUM_LABELS = 5`) in the sentiment variable such as Extremely Negative (0), Extremely Positive (1), Negative (2), Neutral (3) and Positive (4).

Two sets are provided, the training dataset which contains 41.157 tweets and the test dataset which contains 3.798 tweets. For evaluating the model, I have considered 10% of train set as the validation set. The distribution of labels of the three sets is shown in Table 4.22.

Dataset	**Extremely Negative (0)**	**Extremely Positive (1)**	**Negative (2)**	**Neutral (3)**	**Positive (4)**
Train	4929	5937	8925	6964	10286
Dev	542	641	1032	791	1110
Test	592	599	1041	619	947

Table 4.22: Coronavirus tweets NLP dataset labels distribution Train/Dev/Test datasets

4.2.2 Dataset (Italian): SardiStance Dataset

As an Italian dataset for the Multi-Class classification problem, the SardiStance Dataset was chosen. It derives from first shared task regarding Stance Detection (SD) in Italian tweets indeed known as SardiStance [12], proposed at Evalita 2020. The task of SD consists in automatically determining from the text whether the author given a certain context is in favour, against or neutral with respect to a certain target (`NUM_LABELS = 3`). There are many fields of application of SD, such as in politics, social media and public services.
The dataset (taken from the European Language Grid catalogue [9]) consists in collected Italian tweets about the "Movimento delle Sardine", retrieving tweets containing the keywords "sardina" and similar hashtags. The training set is composed by 2.132 tweets, whereas the test set is made up of 1.110 tweets. In order to evaluate the model, 10% of the train set was taken as the validation set. The distribution of labels of the three sets is shown in Table 4.23.

Dataset	Against (0)	Favour (1)	Netrual/None (2)
Train	935	523	461
Dev	99	64	50
Test	742	196	172

Table 4.23: SardiStance dataset labels distribution Train/Dev/Test datasets

As metric evaluation for the competition, it has been considered by the organizers the Average Macro F1-score computed over the two main classes ("Favour" and "Against"), computed as follows:

$$Average\_F1\_score = \frac{F_1(favour) + F_1(against)}{2} \tag{4.7}$$

In the Figure 4.4, all the results obtained in the competition and also the SardiStance baseline are shown.

team name	run	F1-score			
		AVG	AGAINST	FAVOUR	NONE
UNITOR	1	**.6853**	.7866	**.5840**	.3910
UNITOR	1	**.6801**	.7881	**.5721**	.3979
UNITOR	2	.6793	**.7939**	.5647	.3672
DeepReading	1	.6621	.7580	.5663	**.4213**
UNITOR	2	.6606	.7689	.5522	.3702
IXA	1	.6473	.7616	.5330	.3888
GhostWriter	1	.6257	.7502	.5012	.3810
IXA	2	.6171	.7543	.4800	.3675
SSNCSE-NLP	2	.6067	.7723	.4412	.2113
DeepReading	2	.6004	.6966	.5042	.3916
GhostWriter	2	.6004	.7224	.4784	.3778
UninaStudents	1	.5886	.7850	.3922	.2326
baseline		*.5784*	*.7158*	*.4409*	*.2764*
TextWiller	1	.5773	.7755	.3791	.1849
SSNCSE-NLP	1	.5749	.7307	.4192	.3388
QMUL-SDS	1	.5595	.7091	.4099	.2313
QMUL-SDS	2	.5329	.6478	.4181	.3049
MeSoVe	1	.4989	.7336	.2642	.3118
TextWiller	2	.4715	.6713	.2718	.2884
SSN_NLP	1	.4707	.5763	.3651	.3364
SSN_NLP	2	.4473	.6545	.2402	.1913
Venses	1	.3882	.5325	.2438	.2022
Venses	2	.3637	.4564	.2710	.2387

Figure 4.4: Results with baseline for SardiStance task [12]

4.2.3 Fine-Tuning Models

As for the multi-class classification problem, the previous models (i.e. BERT English/-Multilingual, ELECTRA and GPT-2 described in Table 4.4) have been fine-tuned in order to classify the sentiment of the tweets associated with Covid19 and to perform the stance detection task within the tweets on the "Movimento delle Sardine".

Data preparation and Tokenisation

The data that will be fed to the model are represented, after some pre-processing, by a dataframe in the following format: text and label (an integer representing the corresponding label). An example is shown in Figure 4.24 for both datasets used.

Dataset	Data Format	
	text	label
Covid19 Tweets	... Find out how you can protect yourself and loved ones from #coronavirus.? Panic food buying in Germany due to #coronavirus has begun. But the #organic is left behind! Do you remember the last time you paid $2.99 a gallon for regular gas in Los Angeles?Prices at the pump are going down. A look at how the #coronavirus is impacting prices. ...	... 1 0 3 ...
SardiStance	... #Salvinivergognati ha scatenato la macchina del fango contro le #Sardine, quattro post da stamattina. Bene. Vuol dire che ha paura, e fa bene Sono i movimenti dal basso che distruggono le bugie sovraniste della Lega. Avanti tutta ...	... 1 ...

Table 4.24: Data examples for Covid19 Tweets and SardiStance dataset

As explained in previous subsection Data preparation and Tokenisation, the data will be represented by the class `CustomDataset`. Also the `tokenizer`, the `Dataloader` and the architecture of the models will be the same and a configuration section will be used in order to setup the hyperparameters.

4.2.4 Loss Function and Optimizer

Being a one-of-many classification problem, the Multi-Class classification task is treated as a single classification problem of samples in one of `NUM_LABELS` classes. For multi-class classification, the vanilla Cross-Entropy Loss (`CrossEntropyLoss` from the PyTorch package `torch.nn`) is used, which predict one of several classes for each batch example. This loss function is also called "Softmax Loss" because it combines a Softmax activation function plus a Cross-Entropy Loss. The Softmax function calculates the probabilities of

each target class over all possible target classes and the output probabilities are in the range 0 to 1 with the sum of them equals to 1. As a consequence, the target class will have the high probability.

The Softmax mathematical function (4.8) calcuates the ratio of the exponential of the given input value and the sum of exponential values of all the values in the input. So, it is computed as:

$$S(x_i) = \frac{e^{x_i}}{\sum_j^C e^{x_j}} \tag{4.8}$$

where x_j are the scores inferred by the network for each class in C and the softmax activation for a class x_i depends on all the scores in x.

After the logits, i.e. the input unnormalized raw values, have been converted into probability by the Softmax function, the Cross-Entropy loss is calculated as:

$$CE = -\sum_{i=1}^{C} p_i \cdot log(S(x_i)) \tag{4.9}$$

where p_i is the binary truth label for class i and $S(x_i)$ is the softmax probability. In particular, `CrossEntropyLoss` (in the `torch.nn` module) expects as the target for each value the class index instead of one hot encoded vectors and, discarding the elements of the summation which are zero due to target label, the loss can be described as:

$$CE = -log(\frac{e^{x_p}}{\sum_j^C e^{x_j}}) \tag{4.10}$$

where x_p is the network score for the positive class.

As in the previous explained task, also in this multi-label classification problem the AdamW optimizer is used to update the model parameters.

4.2.5 Evaluation Metrics and Results

After the training loop, in order to evaluate the model performances on the validation and test set, the Softmax activation function is applied and it is then picked the class index of the highest probability.

To get a measure of the models' performances, the same previous metrics are computed. The confusion matrix, the "precision vs recall" and "ROC" curves were also considered. In particular, the confusion matrix is a way of visualising in a tabular the performance of the predictions' model understanding which classes are most easily confused. Each entry in the matrix represents the number of predictions made by the model where it classifies the classes correctly or incorrectly. In general, each row represents the instances in a predicted class, while each column depicts the instances in an actual class. However, since it will be used the `confusion_matrix` function from `scikit-learn` module, it is important to note that its convention is that the rows represent actual values and the columns the

predicted ones. Starting from this matrix, it is therefore possible to derive the TP, the TN, the FP and the FN values in order to measure the precision and recall. Moreover, on the diagonal of the confusion matrix there is the counting of TP, so the higher these values the better the predictive ability of the model.

To understand better the use of the confusion matrix, an example is shown below 4.5 in which the TP, FN and FP values have been highlighted for the class "label 0". Specifically, there is only one cell (highlighted in green) where the true label was "label 0" and the predicted label was the corrected one, so it is the true positive (TP) value. The false positives (FP) values are highlighted in orange and represent all those cases where "label 0" was predicted but the actual label was different. Finally, the false negatives (FN) are the ones in pink in which the actual label was "label 0" but the model predicted something else.

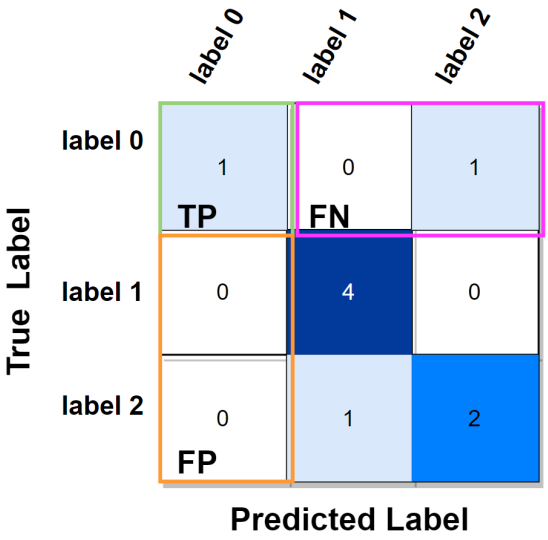


Figure 4.5: Confusion Matrix example

Instead, the ROC (Receiver Operating Characteristics) curve is created by plotting on the y-axis the true positive rate (TPR), i.e. the recall, against the false positive rate (FPR) on the x-axis, which is defined as the ratio of the number of false positives (FP) and the total number of ground truth negatives (FP+TN). Through the analysis of the ROC curves, the ability of the classifier to predict a certain class rather than another is evaluated by calculating the area under the ROC curve (Area Under Curve, AUC).
Typically, the ROC curve is only defined for a binary classification problem in which a label can be predicted as positive or negative. In order to extend the ROC curve to our multi-class classification task, for each class will be estimated an independently ROC curve. Ideally, the TPR should be equal to 1 (thus having no false negative), while the FPR should be 0 (therefore no false positive). Consequently, with the best ideal classifier we would get a ROC curve positioned at the top left end of the plot. The goal is therefore to have a ROC curve as close as possible to that corner of the plot.

Another curve represented in the multi-class experiments done is the Precision (on the y-axis) vs Recall (on the x-axis) curve which is a useful measure when the classes are very imbalanced. An ideal skilful model is represented by a curve that bows towards point (1,1).

Results on Covid19 tweets Dataset

For the Multi-Class classification task, let's consider first the experiments carried out on the English dataset concerning Covid19. For each fine-tuned model, the corresponding selected configuration are reported in Table 4.25. The results achieved will be compared with those present in the article *"Comparative Analysis of Transformer Based Pre-Trained NLP Models"* [23] in which we can take as a reference only the results obtained by the authors with the BERT model.

Hyperparameters	**Bert**	**Bert Multilingual**	**Electra**	**GPT-2**
Epochs	5	5	5	5
Batch Size	16	16	16	16
Learning Rate	2e-5	2e-5	2e-5	2e-5
Adam eps	1e-6	1e-6	1e-6	1e-6
Weight Decay	0.01	0.01	0.01	0.01
Warmup fraction	0.1	0.1	0.1	0.1
Machine	Kaggle GPU	Kaggle GPU	Kaggle GPU	Kaggle GPU
Training and Validation Time	0:48:49	0:49:50	0:50:58	0:53:23

Table 4.25: Hyperparameters selected for the analyzed models with Covid19 tweets Dataset

All the models get good accuracy in predicting the sentiment of the input text. Specifically, the Bert model trained on an English corpus seems to perform better with an F1 score of **97.76** on the dev set and **86.66** on the test set. If we compare this last mentioned value, it can be observed that the fine-tuned Bert model outperforms that used by the researchers in the paper [23] by about 2 points (0.85). In general, the remaining models, with the exception of the GPT-2 model, achieve results close to 0.85 as macro-f1 score.

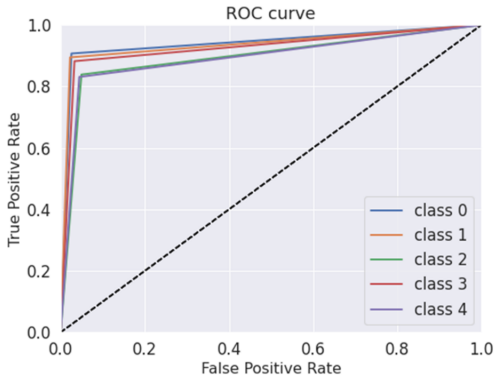
Model	**Accuracy (%)**	**Macro-F1 (%)**
Bert	97.67	97.76
Bert-Multilingual	96.77	96.80
Electra	95.57	95.64
GPT-2	73.47	74.12

Table 4.26: Results on dev set (Covid19 tweets Dataset)

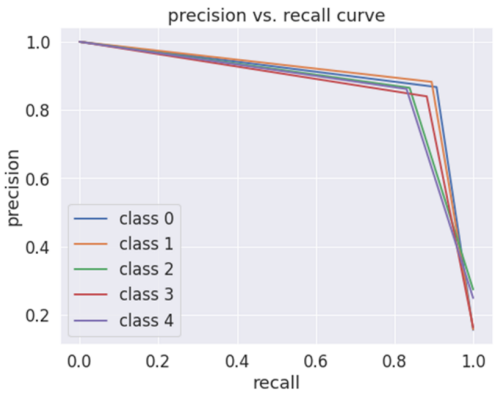
Model	Accuracy (%)	Macro-F1 (%)
Bert	86.31	86.66
Bert-Multilingual	85.42	85.85
Electra	85.07	85.43
GPT-2	62.92	64.08

Table 4.27: Results on test set (Covid19 tweets Dataset)

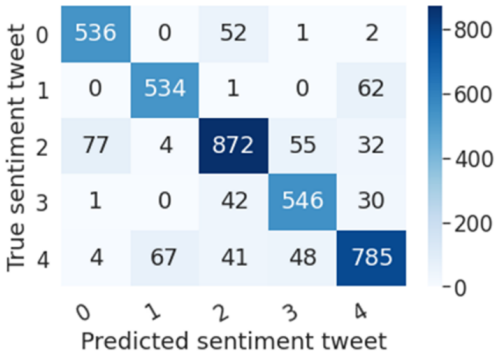
Figures 4.6 and 4.7 report the ROC and prevision/recall curves for the BERT-English and Bert-Multilingual models respectively. Looking at the ROC curves (a), the class 0 ("Extremely Negative") has a high AUC value compared to all other models. While, BERT is having difficulty in correctly classifying class 2 ("Negative") and 4 ("Positive"). In fact, the accuracy percentage is lower for the "Negative" (79.03% for Bert-English and 81.26% for Bert-Multilingual) and "Positive" (83.06% for Bert-English and 80.95% for Bert-Multilingual) classes. While, as confirmed by the ROC curve, the class classified with greater accuracy is the "Extremely Negative" with an accuracy of 90.69% for Bert-English and 92.03% for Bert-Multilingual.
Looking at the Precision vs Recall curve (b), class 4 ("Positive") has a low F1-score while class 0 ("Extremely Negative") has a high F1-score. Class 1 ("Extremely Positive") ranks well with a low misclassification error compared to all other classes.



(a) ROC curve

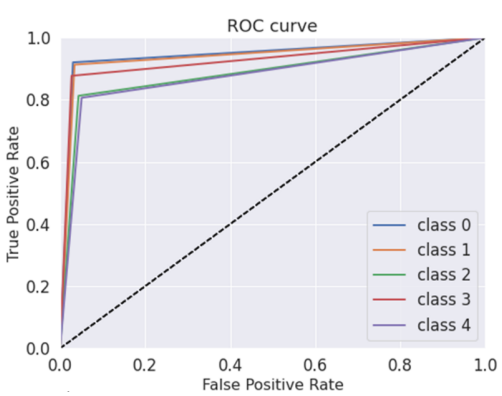


(b) Precision vs Recall curve

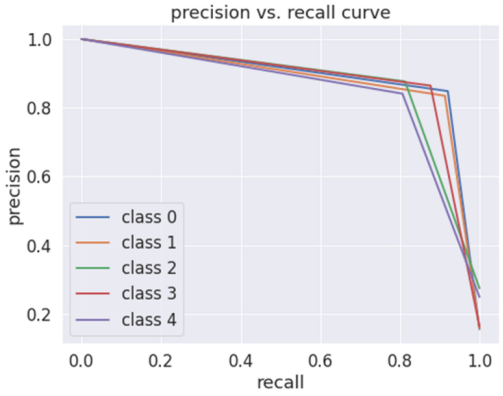


(c) Confusion Matrix

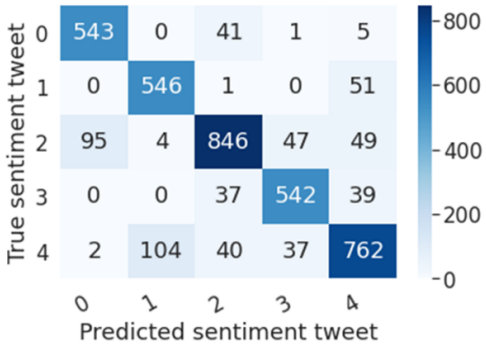
Figure 4.6: ROC curve, Precision/Recall curve and Confusion Matrix for the Bert-English model on test set



(a) ROC curve



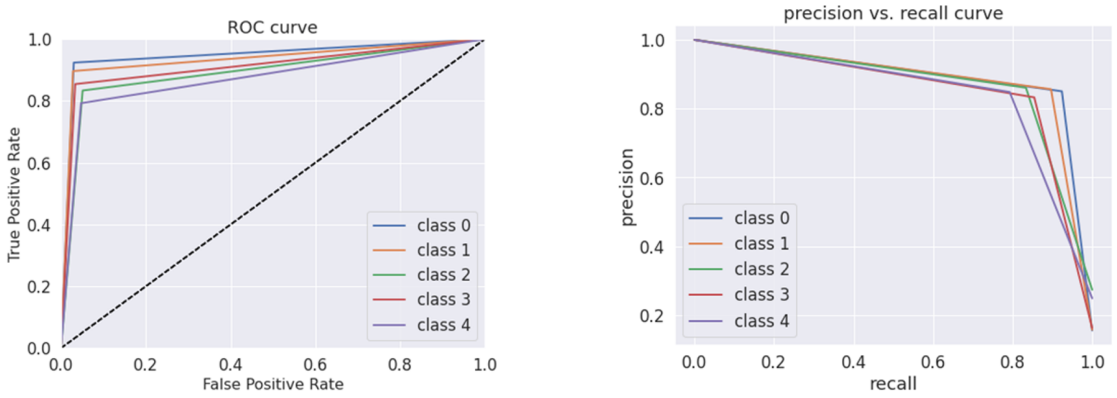
(b) Precision vs Recall curve



(c) Confusion Matrix

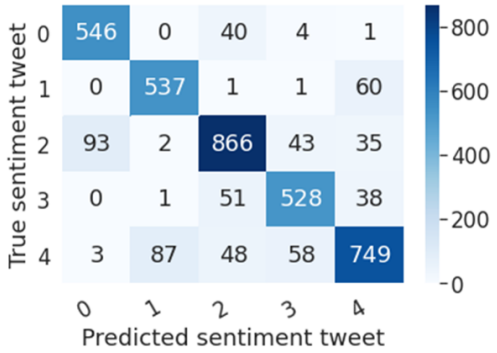
Figure 4.7: ROC curve, Precision/Recall curve and Confusion Matrix for the Bert-Multilingual model on test set

Considering now the Electra model, also in this case the "Extremely Negative" (0) class has a high F1-score (Figure 4.8 (b), while the class with the lowest F1-score is the "Positive" (4) one. The accuracy is respectively 92.39% for the former class and 79.26% for the latter.



(a) ROC curve

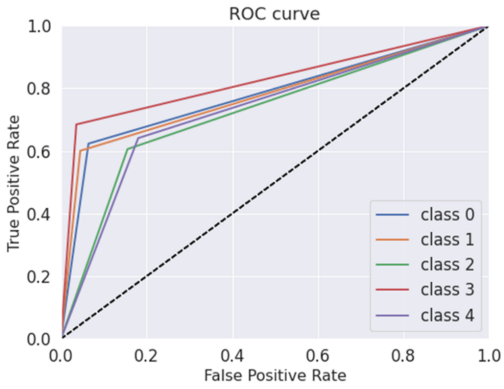
(b) Precision vs Recall curve



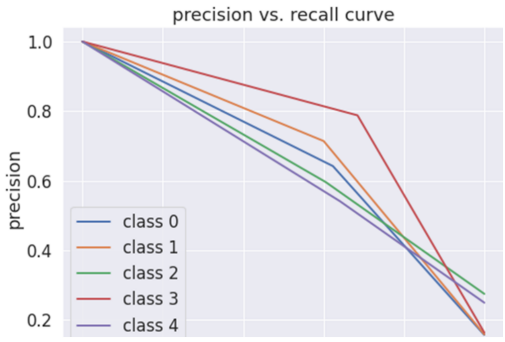
(c) Confusion Matrix

Figure 4.8: ROC curve, Precision/Recall curve and Confusion Matrix for the Electra model on test set

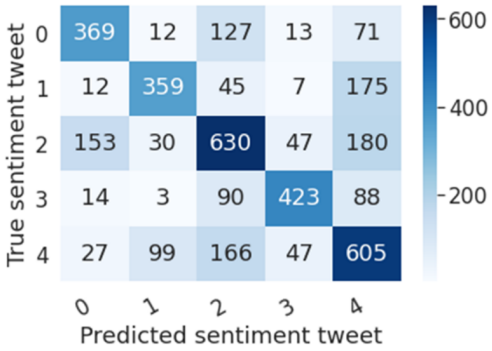
Looking at the figure 4.9, it is evident that the GPT-2 model has worse performance. In particular, from the precision/recall curve (b), it can be observed that class 4 ("Positive") has a lower f1-score unlike class 3 ("Neutral") which has a higher f1-score. This last result obtained differs from the previous models analyzed in which it was class 0 (Extremely Negative) to have a higher score.



(a) ROC curve



(b) Precision vs Recall curve



(c) Confusion Matrix

Figure 4.9: ROC curve, Precision/Recall curve and Confusion Matrix for the GPT-2 model on test set

In Table 4.31 are reported the classification report values obtained on the test set for the various models taken into consideration.

Label	Precision	Recall	F1-score	Support
Extremely Negative	0.85	0.92	0.88	590
Extremely Positive	0.84	0.91	0.87	598
Negative	0.86	0.84	0.85	1040
Neutral	0.84	0.88	0.86	619
Positive	0.86	0.83	0.85	945
macro-average	**0.86**	**0.87**	**0.86**	3792

Table 4.28: Results on Test set using Bert-English Model (Covid19 Tweets Dataset)

Label	Precision	Recall	F1-score	Support
Extremely Negative	0.87	0.90	0.88	591
Extremely Positive	0.88	0.89	0.88	597
Negative	0.87	0.81	0.84	1040
Neutral	0.86	0.87	0.87	618
Positive	0.84	0.81	0.82	945
macro-average	**0.85**	**0.87**	**0.86**	3792

Table 4.29: Results on Test set using Bert-Multilingual Model (Covid19 Tweets Dataset)

Label	Precision	Recall	F1-score	Support
Extremely Negative	0.85	0.92	0.86	591
Extremely Positive	0.86	0.89	0.88	599
Negative	0.86	0.83	0.85	1039
Neutral	0.83	0.86	0.84	618
Positive	0.85	0.79	0.82	945
macro-average	**0.85**	**0.86**	**0.85**	3792

Table 4.30: Results on Test set using Electra Model (Covid19 Tweets Dataset)

Label	Precision	Recall	F1-score	Support
Extremely Negative	0.64	0.62	0.63	592
Extremely Positive	0.71	0.60	0.65	598
Negative	0.60	0.61	0.60	1040
Neutral	0.79	0.68	0.73	618
Positive	0.54	0.64	0.59	944
macro-average	**0.66**	**0.63**	**0.64**	3792

Table 4.31: Results on Test set using GPT-2 Model (Covid19 Tweets Dataset)

Results on SardiStance Dataset

Looking now at the Italian dataset for the multi-class classification problem, the average-F1-score considered in the shared SardiStance Evalita task 4.7 is also added as a metric in the results obtained on the dev and test set shown in Table 4.33 and 4.34. These results were obtained with the hyperparameters indicated in Table 4.32.

Hyperparameters	**Bert**	**Bert Multilingual**	**Electra**	**GPT-2**
Epochs	6	6	6	6
Batch Size	16	16	16	16
Learning Rate	2e-5	2e-5	2e-5	2e-5
Adam eps	1e-6	1e-6	1e-6	1e-6
Weight Decay	0.01	0.01	0.01	0.01
Warmup fraction	0.1	0.1	0.1	0.1
Machine	Kaggle GPU	Kaggle GPU	Kaggle GPU	Kaggle GPU
Training and Validation Time	0:03:36	0:03:12	0:03:08	0:03:19

Table 4.32: Hyperparameters selected for the analyzed models with SardiStance Dataset

Compared to the computed baseline 4.4 for the SardiStance task in which the average F1-score is equal to 0.5784, the fine-tuned models analyzed achieve better results in most cases. In particular, considering the test set (shown in Table 4.33), Bert-Multilingual model scores the highest average-F1 score of 0.62, overcoming the base Bert model by 4 points and the Electra model by 2 points. The GPT-2 model is still confirmed to be the least performing model.

Model	**Accuracy (%)**	**Macro-F1 (%)**	**Average-F1 (%)**
Bert	80.29	77.32	0.84
Bert-Multilingual	95.19	94.49	0.95
Electra	72.60	66.88	0.76
GPT-2	48.56	26.79	0.40

Table 4.33: Results on dev set (SardiStance Dataset)

Model	**Accuracy (%)**	**Macro-F1 (%)**	**Average-F1 (%)**
Bert	63.68	47.07	0.58
Bert-Multilingual	63.41	54.67	0.62
Electra	60.87	48.19	0.60
GPT-2	66.03	27.25	0.40

Table 4.34: Results on test set (SardiStance Dataset)

Specifically, unlike the remaining models considering, GPT-2 has a greater difficulty in correctly classifying tweets labeled as "FAVOR" (as reported in Table 4.38).
While, all the models reach a low F1-score in predicting the "NONE/NEUTRAL" class which also has a smaller number of samples than the remaining two classes. This behavior can also be seen by observing the plots of the Precision/Recall curve (Figure (b) in 4.10, 4.11, 4.12 and 4.13) in which the curve associated with class 2 ("NONE") underlies the remaining two curves representing the "FAVOR" and "AGAINST" classes. Moreover, by looking at the ROC curves (Figure (a) in 4.10, 4.11, 4.12 and 4.13), they show that class 0 ("AGAINST") has a high AUC, which indicates that the models are able to classify this class with greater accuracy.

Label	**Precision**	**Recall**	**F1-score**	**Support**
AGAINST	0.73	0.80	0.77	738
FAVOR	0.39	0.39	0.39	194
NONE	0.34	0.20	0.25	172
macro-average	**0.49**	**0.47**	**0.62**	1104

Table 4.35: Results on Test set using Bert-English Model (SardiStance Dataset)

Label	**Precision**	**Recall**	**F1-score**	**Support**
AGAINST	0.82	0.70	0.75	738
FAVOR	0.42	0.56	0.48	194
NONE	0.37	0.47	0.41	172
macro-average	**0.53**	**0.57**	**0.55**	1104

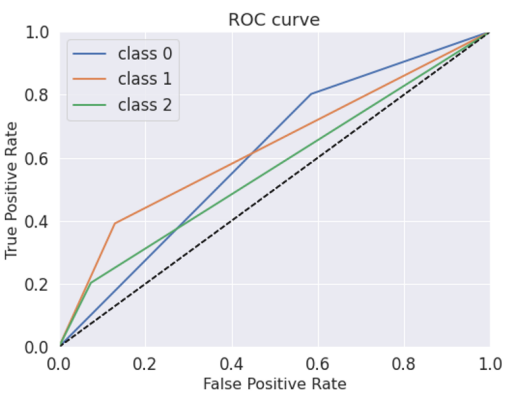
Table 4.36: Results on Test set using Bert-Multilingual Model (SardiStance Dataset)

Label	**Precision**	**Recall**	**F1-score**	**Support**
AGAINST	0.76	0.72	0.74	737
FAVOR	0.37	0.52	0.44	195
NONE	0.31	0.25	0.28	172
macro-average	**0.48**	**0.50**	**0.48**	1104

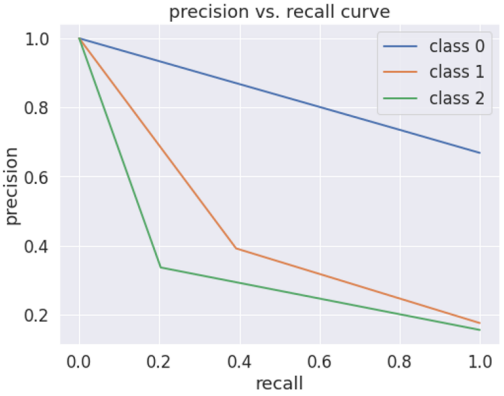
Table 4.37: Results on Test set using Electra Model (SardiStance Dataset)

Label	**Precision**	**Recall**	**F1-score**	**Support**
AGAINST	0.67	0.99	0.80	738
FAVOR	0.00	0.00	0.00	195
NONE	0.15	0.02	0.02	171
macro-average	**0.27**	**0.33**	**0.27**	1104

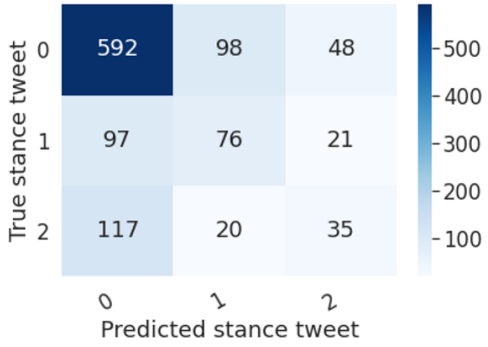
Table 4.38: Results on Test set using GPT-2 Model (SardiStance Dataset)



(a) ROC curve

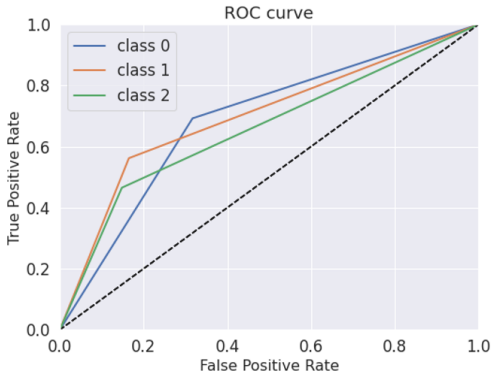


(b) Precision vs Recall curve

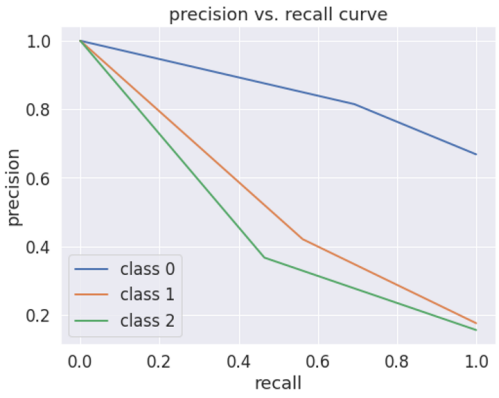


(c) Confusion Matrix

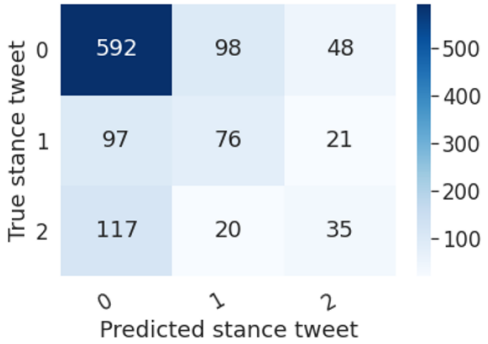
Figure 4.10: ROC curve, Precision/Recall curve and Confusion Matrix for the Bert-English model on test set



(a) ROC curve

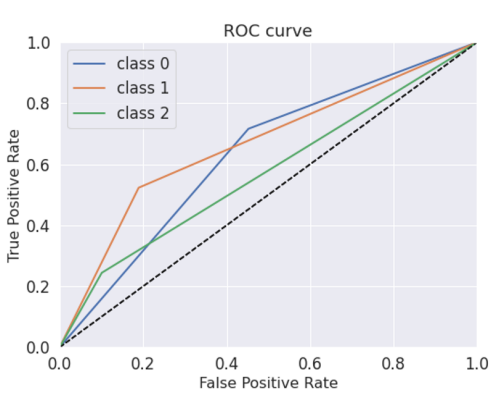


(b) Precision vs Recall curve

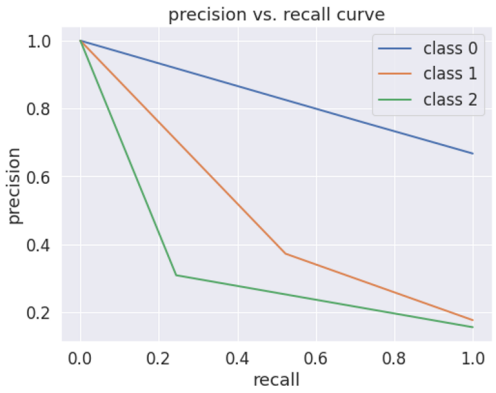


(c) Confusion Matrix

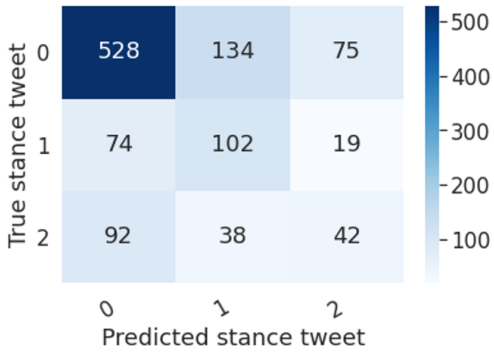
Figure 4.11: ROC curve, Precision/Recall curve and Confusion Matrix for the Bert-Multilingual model on test set



(a) ROC curve

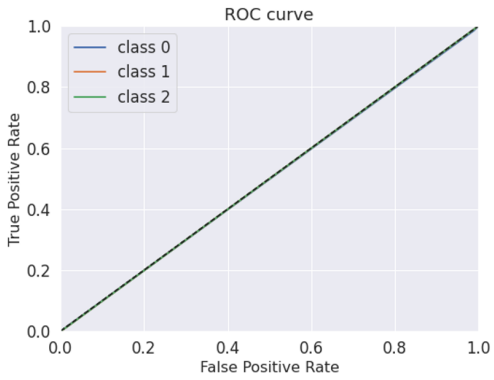


(b) Precision vs Recall curve

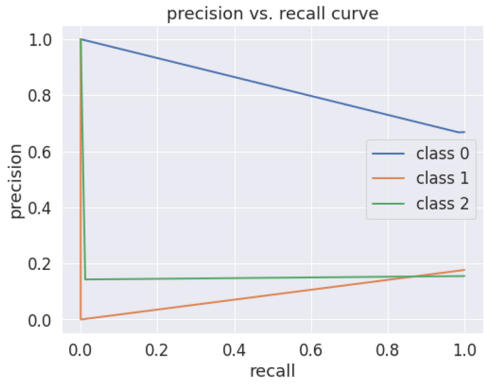


(c) Confusion Matrix

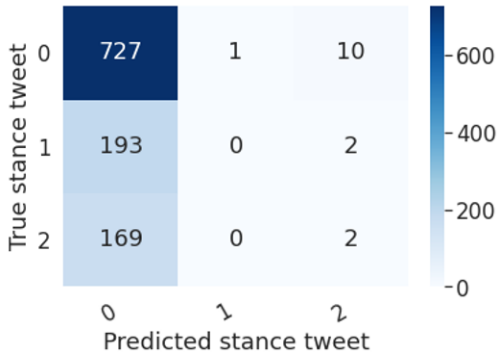
Figure 4.12: ROC curve, Precision/Recall curve and Confusion Matrix for the Electra model on test set



(a) ROC curve



(b) Precision vs Recall curve



(c) Confusion Matrix

Figure 4.13: ROC curve, Precision/Recall curve and Confusion Matrix for the GPT-2 model on test set

Chapter 5

Conclusions

Looking at the obtained results several conclusions can be inferred:

- All the transformer models have been fine-tuned very quickly for the classification task. This is one of the main benefit behind Transformer-based models, which being already intensely pre-trained, they already understand the language and so we can fine-tune them in an inexpensive way on numerous downstream NLP tasks.
- The application of a pre-trained model (in our case the Bert-Multilingual model) on a multilingual corpus to an Italian language dataset leads to better performances with greater accuracy in the results compared to the use of models pre-trained only on a set of English data.
- The dataset also plays an important role in obtaining reliable results. In fact, the classification of unbalanced or small annotated datasets involves the phenomenon of overfitting in which the model is able to preform better on the training data than on data it has never seen before. As a consequence, the model learns specific representations at the training data and does not generalize well to test data.
- Being NLP tasks different in nature, following the various experiments carried out, it can be concluded that is preferable to use auto-regressive models, such as GPT-2, for long-text generation task where that model has been very successful. Learning left-to-right language models, their disadvantage lies in the unidirectionality of the attention mechanism, which does not allow the interaction of the context tokens to be entirely captured. On the other hand, autoencoding models, such as Bert and Electra, based on the bidirectional representation of the context, are more successful at natural language understanding tasks such as text classification considered in this thesis.

5.1 Future developments

In this Master Thesis, different models have been analyzed by applying them to a textual classification task. However, such models could be applied in future experiments to a variety of NLP tasks such as Q&A, summarization, text generation, and so on.

It would also be interesting to analyze new frameworks such as the T5 model. The Text-To-Text Transfer Transformer (T5) was created at Google [22] with the aim of obtaining a framework that would convert all text-based language problems into a text-to-text format where the input and output are always text strings, unlike the BERT models that can only output a class label or an input part. Its main advantage is the possibility to perform multiple different tasks with a single trained model. For example, as shown in the Figure 5.1, to tell the T5 model that you wanted to perform a certain task, you will give the model an input string indicating both the task that you wanted to do and the data that you want to perform that task on.

This is just another example of a Transformer architecture based model that can be explored in a future works.

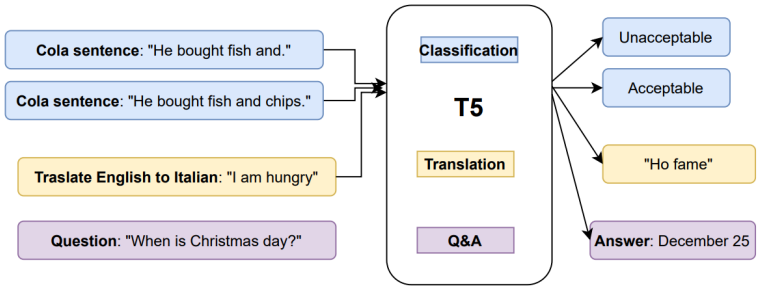


Figure 5.1: Example of classification, translation and Q&A tasks with T5 model

Appendices

Appendix A

Implemented scripts for fine-tuning the pre-trained models

The scripts I created in order to train and fine-tune the models analysed in this thesis have been saved as Jupyter Notebook and they can be viewed on GitHub at the following address: GitHub Repository Master's Thesis Project

Bibliography

[1] *BCEWithLogitsLoss Pytorch*. URL https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html.

[2] *AMI 2020 Dataset hosted at ELG*, . URL https://live.european-language-grid.eu/catalogue/corpus/7005.

[3] *Automatic Misogyny Identification (AMI): Shared Task at Evalita 2020*, . URL https://amievalita2020.github.io/.

[4] *Google Colaboratory*. URL https://colab.research.google.com/.

[5] *Coronavirus tweets NLP - Text Classification Dataset*. URL https://www.kaggle.com/datatattle/covid-19-nlp-text-classification.

[6] *GoEmotions: A Dataset of Fine-Grained Emotions*. URL https://github.com/google-research/google-research/tree/master/goemotions.

[7] *Kaggle*. URL https://www.kaggle.com/.

[8] *"PyTorch, the PyTorch logo and any related marks are trademarks of Facebook, Inc.* URL https://pytorch.org/.

[9] *SardiStance Dataset hosted at ELG*. URL https://live.european-language-grid.eu/catalogue/corpus/5245.

[10] *Transformers Library Hugging Face*. URL https://huggingface.co/transformers/.

[11] Soumith Chintala. *Pytorch logo - "PyTorch, the PyTorch logo and any related marks are trademarks of Facebook, Inc"*. URL https://github.com/pytorch/pytorch/blob/master/docs/source/_static/img/pytorch-logo-dark.png.

[12] Alessandra Teresa Cignarella, Mirko Lai, Cristina Bosco, Viviana Patti, Rosso Paolo, et al. Sardistance@ evalita2020: Overview of the task on stance detection in italian tweets. In *EVALITA 2020 Seventh Evaluation Campaign of Natural Language Processing and Speech Tools for Italian*, pages 1–10. Ceur, 2020.

[13] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*, 2020.

[14] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. *ELECTRA*, 2020. URL https://github.com/google-research/electra.

[15] Dorottya Demszky, Dana Movshovitz-Attias, Jeongwoo Ko, Alan Cowen, Gaurav Nemade, and Sujith Ravi. *GoEmotions: A Dataset of Fine-Grained Emotions*, 2020.

[16] Jacob Devlin. *GLUE benchmark leaderboard for BERT model.* URL https://gluebenchmark.com/leaderboard.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019.

[18] Paolo Rosso Elisabetta Fersini, Debora Nozza. *AMI @ EVALITA2020: Automatic Misogyny Identification.* In Valerio Basile, Danilo Croce, Maria Di Maro, and Lucia C. Passaro, editors, *Proceedings of the 7th evaluation campaign of Natural Language Processing and Speech tools for Italian (EVALITA 2020)*, Online, 2020. CEUR.org.

[19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[20] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, Daniela Amodei, Jack Clark, Miles Beundage, and Ilya Sutskever. *GPT-2*, 2019. URL https://openai.com/blog/better-language-models/.

[21] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. *Language Models are Unsupervised Multitask Learners.* 2019.

[22] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2020.

[23] Saurav Singla. Comparative analysis of transformer based pre-trained nlp models. *INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING*, 8:40–44, 11 2020. doi: 10.26438/ijcse/v8i11.4044.

[24] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*, 2014.

[25] The Hugging Face Team. *Hugging Face.* URL https://huggingface.co/.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*, 2017.

Acronyms

AMI Automatic Misogyny Identification. 43

AUC Area Under Curve. 61

BERT Bidirectional Encoder Representations from Transformers. 25

ELECTRA Efficiently Learning an Encoder that Classifies Token Replacement Accurately. 25

FN False Negative. 48, 61

FP False Positive. 48, 61

GPT-2 Generative Pre-Trained 2. 25

LSTM Long short-term memory. 19

MLM Masked Language Model. 26, 31

NLG Natural Language Generation. 34, 37

NLP Natural Language Processing. 17

NLU Natural Language Understanding. 37

NSP Next Sentence Prediction. 27

P Precision. 48

R Recall. 48

RNN Recurrent Neural Network. 18

ROC Receiver Operating Characteristics. 60, 61

RTD Replaced Token Detection. 31

SD Stance Detection. 57

Seq2Seq sequence-to-sequence. 20

TL Transfer Learning. 25

TN True Negative. 61

TP True Positive. 48, 61