uc3m | Universidad Carlos III de Madrid

Master Degree in Big Data Analytics
2021-2022

Master Thesis

“DistilBERT, the alternative to massive models for natural language processing”

Ion Bueno Ulacia

Supervised by:
Pablo Martínez Olmos
Madrid, June 2022

ABSTRACT

Natural Language Processing (NLP) encloses a wide range of applications which play a critical role in daily business, as automating reviews or developing bots for a web page. Since the publication of BERT, pre-trained language models dominate the area, with a growing trend of increasing the number of parameters to obtain state-of-the-art results and the need of processing huge amounts of data. In spite of overcoming lighter implementations, they require high computational resources and memory requirements, in addition to spend long inference times, constraining real-time situations and scaling. DistilBERT is proposed as an alternative to reduce the size and inference time of massive models maintaining the same performance. It applies distillation over BERT and obtains an architecture 40% smaller. The aim is to compare BERT and DistilBERT in the resolution of two NLP tasks, classification with more than one category and with multiple labels. The obtained results show that DistilBERT was 50% faster in both tasks, reaching the same performance than BERT in multiclass and overcoming it in multilabel classification. DistilBERT presents distillation as an optimal technique to reduce size and computational resources of current NLP state-of-the-art models.

Keywords: NLP, BERT, distillation, DistilBERT, multiclass, multilabel.

ACKNOWLEDGMENTS

Sagrario, Felipe and Aitor, my family, for their support during many years of study. They have provided me everything in every moment. I will never be able to thank them the luck I have had.

Elena Manso, her support and patience are limitless. She has been my pillar from the first second to the last one. Her confidence in me has been my motor during all this time.

My classmates, who have become friends for life. Long life to Consulate.

Finally, Pablo, my thesis supervisor. Thanks for his efforts and attention during these months, but especially, for being that professor who I will never forget.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AUC Area Under the Curve.

BERT Bidirectional Encoder Representations from Transformers.

BPE Byte Pair Encoding.

CNN Convolutional Neural Network.

CPU Central Processing Unit.

GLUE General Language Understanding Evaluation.

GPT Generative Pre-trained Transformer.

GPU Graphics Processing Unit.

GRU Gated Recurrent Unit.

IMDb Internet Movie Database.

LSTM Long Short-Term Memory.

NLP Natural Language Processing.

RNN Recurrent Neural Network.

ROC Receiver Operating Characteristic.

SQuAD Stanford Question Answering Dataset.

SWAG Situations With Adversarial Generations.

TPU Tensor Processing Unit.

1. INTRODUCTION

1.1. Motivation of Work

This work continues the bachelor thesis [1] focused on Bidirectional Encoder Representations from Transformers (BERT) [2], whose discovery meant a potential boost to the area with a new successful approach. The main points were the explanation of its architecture and mechanisms, as well as the implementation of two Natural Language Processing (NLP) tasks: sentiment analysis and questions answering.

In spite of being a strategy which requires a lot of resources, new models followed the same idea, with many more parameters and consequently much longer execution times, as GPT-3 [3]. The common approach followed by leading companies of the sector was training larger models adding progressively more parameters. In spite of that, there is another trend which aims to obtain the same performance with smaller models, using novel training techniques.

That is the case of DistilBERT [4], whose performance is really close to BERT (97%), being 40% smaller and 60% faster. This model is just a modified version of BERT with new training techniques, which can easily be used in edge applications. This reduction of the architecture is transferable to other massive models, so this strategy may have a huge impact in the area.

1.2. Objectives

The aim of the work is to compare BERT and DistilBERT models in the resolution of two common NLP tasks, classification with **multiple classes** and with **multiple labels**. The points to take into account, apart from the performance, are the spent time to fine-tune the model and the total number of parameters. Mention these models are more used in language modeling as question answering, but due to the available computational power, only simple tasks can be implemented.

In addition to the implementations, it is provided a detailed description of DistilBERT and the reasons of why it is useful.

1.3. Learning Outcomes

DistilBERT architecture and its training mechanisms, in particular the knowledge distillation and why it is so useful when trying to reduce the size of a given model.

Nevertheless, the main challenges come with the tasks' implementation, involving the management of datasets and the corresponding pre-processing, the implementation employing pre-trained models from Hugging Face [5] and fine-tuning, interpretation of the results, etc.

Mention the difficulties and problems with the management and training of BERT and DistilBERT. The main point was the execution's interruption while fine-tuning the models, due to the limited times in Google Colaboratory. This environment was employed due to the availability of GPUs, very advisable with this type of models.

1.4. Thesis Structure

First part makes an introduction of the state-of-the-art in section 2. Due to the high amount of content which should be included to reflect a complete transition between first NLP models to DistilBERT, only the most recent and relevant points are explained, starting from the transformer [6] and attention mechanisms to BERT.

Second, the architecture and training techniques of DistilBERT [4] are presented in section 3, explaining in detail the distillation technique which is the essential aspect of the implementation and a common approach when reducing the size of a model.

Finally, in section 4, both experiments (multiclass and multilabel) with BERT and DistilBERT are explained as well as the own implementations. Then, the performance of both models is compared, taking into account the spent time. In appendix A there are two figures which show the main mechanism of both models, self-attention.

The implementation as a notebook and the developed models, with all data and results are published in GitHub: ion-bueno/distilbert-from-inside.

2. STATE OF THE ART

Natural Language Processing (NLP) can be defined as how computers understand and manipulate natural languages to perform certain tasks as text classification, text modelling, translation, question answering, summarization, etc [7]. The origin dates from 1950s, so multiple models and strategies have been followed.

Some fast but very useful approaches in classification tasks are **bag of words** [8] or **TF-IDF** [9], although too simple for more complex NLP problems which involve language modeling. In this case, **Recurrent Neural Networks** (RNNs) are widely employed, from simple architectures to more complex gates as Gated Recurrent Unit (GRU) [10] and Long Short-Term Memory (LSTM) [11].

Nevertheless, these networks have two clear problems. First one, it takes a long time to train them due to the sequential nature, especially with long term sentences. Second, the difficulty to capture long-term dependencies as the sequence or spans increase, causing vanishing or exploding gradients [12].

Mention some data which will be presented below is part of the work in [1].

2.1. Attention Mechanisms

Attention was originally introduced to solve the problems with long term dependencies [13], [14], to retain the most significant information, with bias alignment over inputs [15]. They also have been widely used in design with Convolutional Neural Networks (CNNs).

The following explanation is originally from [15]. The core component is called attention layer, whose input is the **query**, for which the layer returns a set of **key-value** pairs encoded.

Assuming there are n key-value pairs $(k_1, v_1), \ldots, (k_n, v_n)$ with $k_i \in \mathbb{R}^{d_k}$, $v_i \in \mathbb{R}^{d_v}$, given a query $q \in \mathbb{R}$, it is computed a score function α which measures the similarity between query and key. A simple one could be the dot product. For each key $k_1, \ldots, k_n$ the scores are $a_1, \ldots, a_n$.

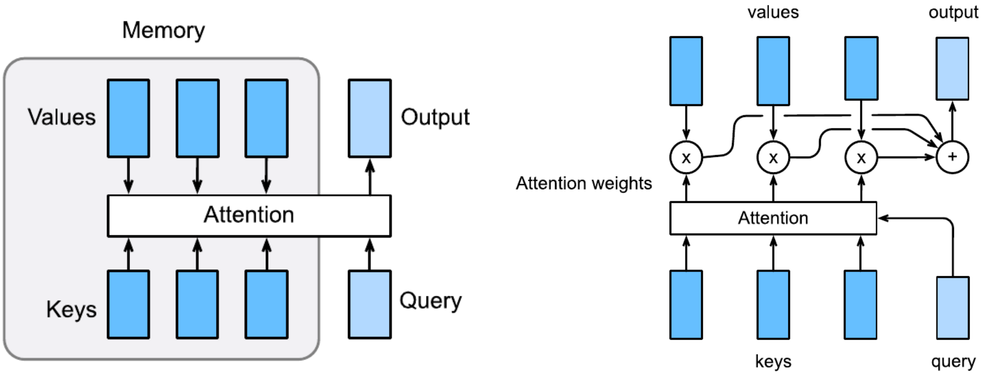
$$a_i = \alpha(q, k_i) \tag{2.1}$$

To get the attention weights per score, it is applied softmax function.

$$b_i = \frac{\exp^{a_i}}{\sum_j \exp^{a_j}} \tag{2.2}$$

The attention layers return an output $o \in \mathbb{R}^{d_v}$ with the same shape as the value. This corresponds with a weighted sum of the values:

$$o = \sum_{i=1}^{n} b_i v_i \qquad (2.3)$$



(a) Attention layer diagram.

(b) Attention output.

Fig. 2.1. Attention mechanism [15].

2.1.1. Self-Attention

Attention mechanisms were intended to be applied between the input and output sequence, for example, in machine translation. The intuition is to relate the source words with the target words.

Nevertheless, attention can be just employed within the input elements relating different positions of a single sequence in order to compute a representation of this one. The goal is mapping sets to sets, regardless of the order in the sequence.

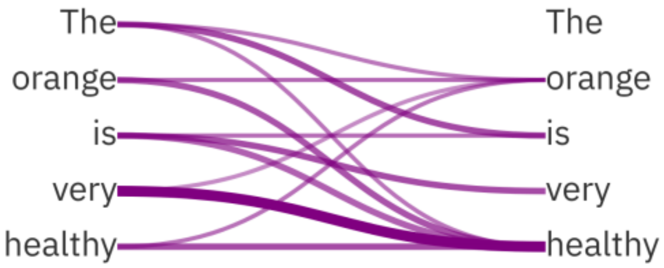


Fig. 2.2. Attention weights visualization using exBERT [16].

Self-attention has been successfully implemented in muliple tasks such as summarization [17], reading comprehension [18], embedding sentences [19] or natural language inference [20]. It is the essential component in the transformer [6] and consequently in BERT [2] and DistilBERT [4].

Visualizations of self-attention in BERT and DistilBERT are presented in appendix A.

2.2. The Transformer

Presented in [6] as the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence aligned RNNs or convolution. The model meant a new state of the art in translation quality, which was the original goal.

The model is based in an encoder-decoder structure, as other competitive neural sequence transduction models. The input of the encoder corresponds with the original sequence, whereas the generated sentence is passed through the decoder. The Transformer is an auto-regressive model because it generates one symbol at a time, consuming the previously generated.

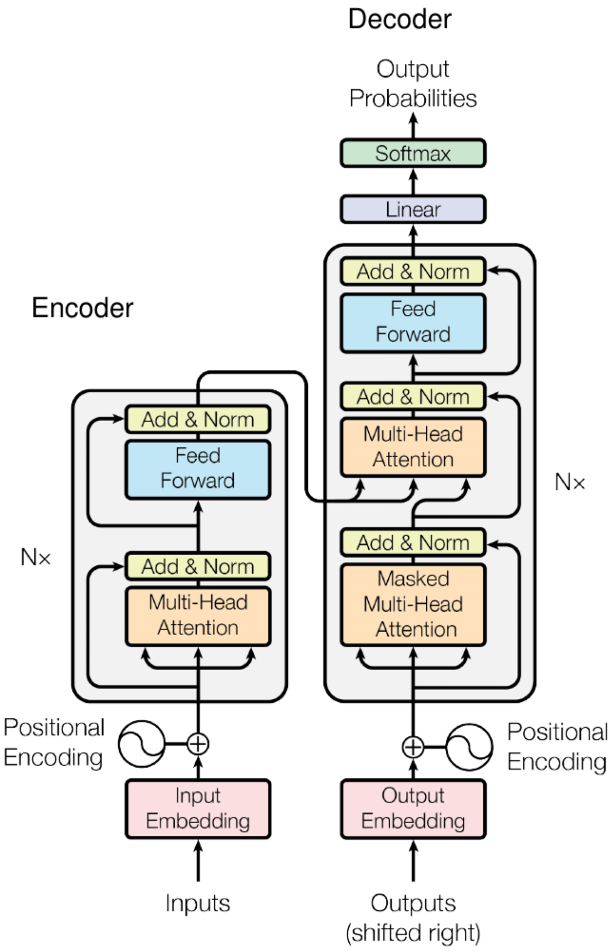


Fig. 2.3. Transformer architecture [6].

The relationship between this work and the Transformer is that both BERT and DistilBERT are just the Transformer's encoder in terms of architecture.

2.2.1. Transformer Encoder

The encoder stacks N encoder blocks which map sets to sets, what means the output size is the same than input: (m x n x d_{model}), where the variables are:

- Number of sentences in a batch: m.
- Number of words per sentence: n.
- Embedding dimension: d_{model}.

In the original implementation, they employ $N = 6$ encoders. Each block is composed by two sub-layers: **Multi-Head Attention**, whose function is applying self-attention several times (depending on the number of heads), and **Feed Forward**, consisting of a multilayer perceptron. Both are followed by an **Add & Norm** layer, which is a residual connection [21] followed by layer normalization [22].

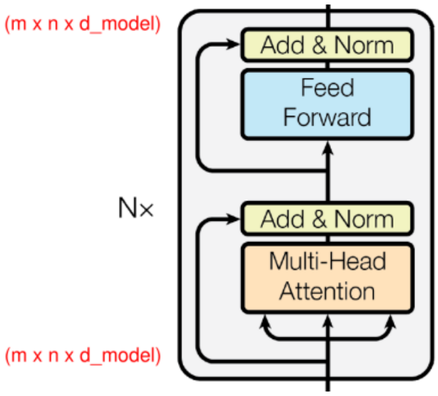


Fig. 2.4. Encoder architecture [6].

2.3. Bidirectional Encoder Representations from Transformers (BERT)

Designed to pretrain deep bidirectional representations from unlabeled text by using self-attention to get the left and right context. This approach overcomes current pre-training techniques, where they use unidirectional language models to learn general language representations. Two examples which were competitive models before BERT are ELMo [23] and GPT [24].

This pretrained model can be fine-tuned adding one output layer and used for a wide range of NLP tasks. The original model developed in TensorFlow is published in GitHub google-research/bert.

2.3.1. Model Architecture

It is a multi-layer bidirectional Transformer encoder, based on [6] and implemented as [25]. The different components with the original values defined for $\text{BERT}_{\text{BASE}}$ (there is a larger version, $\text{BERT}_{\text{LARGE}}$) are:

- Number of layers (12): encoder blocks which compose the whole stack. Defined as N in the Transformer section 2.2.1.
- Hidden size (768): embedding dimension of the model. Previously defined as d_{model}.
- Self-attention heads (12): number of heads in self-attention blocks.
- Inner layer feed-forward (3072): defined as 4 times the hidden size.

2.3.2. Input-Output Representations

In order to handle a variety of tasks, the input can be a single sentence or a pair of sentences, composed by a set of tokens. The first one is always a special classification token [CLS], whose final hidden state layer is used in classification tasks. The input representation of a word is the addition of three different embeddings:

- **Token Embedding**: normal word embedding. It is used WordPiece [26] with 30k token vocabulary.
- **Segment Embedding**: to differentiate the sentence the word belongs to. It is used the special token [SEP] to separate both sentences.
- **Position Embedding**: to indicate the position in the whole sequence.

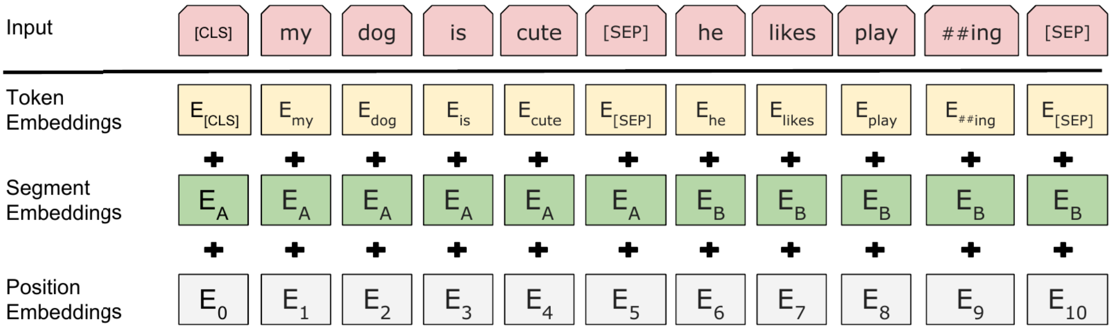

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Fig. 2.5. Input embedding in BERT [2].

As output it is obtained the corresponding sequence embedding and a special token employed in classification. These values are going to be explained better in next section.

2.3.3. Pre-Training

In order to make the model understand the language and its context, two unsupervised tasks are employed. As data, it is used the BooksCorpus [27] and English Wikipedia (2,500M words). Training of BERT (base version) was performed on 4 Cloud TPUs (16 TPU chips total).

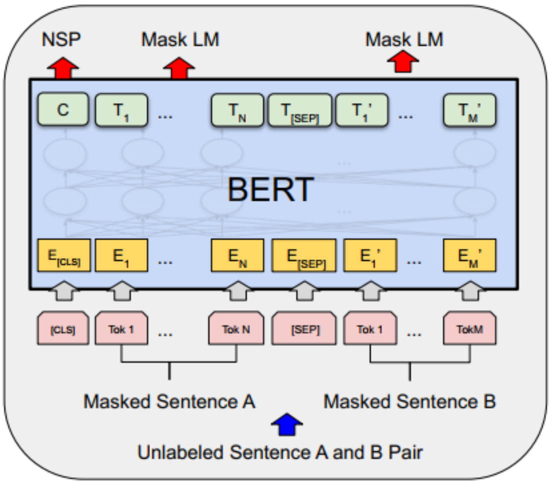


Fig. 2.6. Pre-training process in BERT [2].

First method corresponds with **masked language model**, where some tokens are masked at random and then predicted. The final T_i vectors have the same size and are generated simultaneously. They are fed into an output softmax over the vocabulary (30k) to get a distribution and compare with cross entropy loss in order to train the model.

At the same time, the model is being trained with **next sentence prediction**, where it is used the special token [CLS] in the input and C in the output to indicate if *Sentence B* could follow *Sentence A*. The reason to apply it in addition to masked language modeling is that a lot of tasks as question and answering are based on understanding relationship between two sentences, which is not captured by the previous technique.

2.3.4. Fine-Tuning

Self-attention layers apply bidirectional cross attention between the two sentences, making straightforward to model downstream tasks. Besides that, the fact that two sequences are used as input in pre-training, is equivalent to different NLP tasks as sentence pairs in paraphrasing, hypothesis-premise pairs, question-passage pairs, etc.

Output token representations and C are fed into an output layer. Only new parameters are learned from sratch, while the rest are slightly fine-tuned. That is the reason fine-tune is relatively inexpensive in computation cost compared with pre-training [2].

The original experiments in the paper [2] correspond with text classification on GLUE [28] and question answering on SQuAD [29]. It is also evaluated on SWAG [30], where given a sentence, the task is to choose the most plausible continuation among four choices.

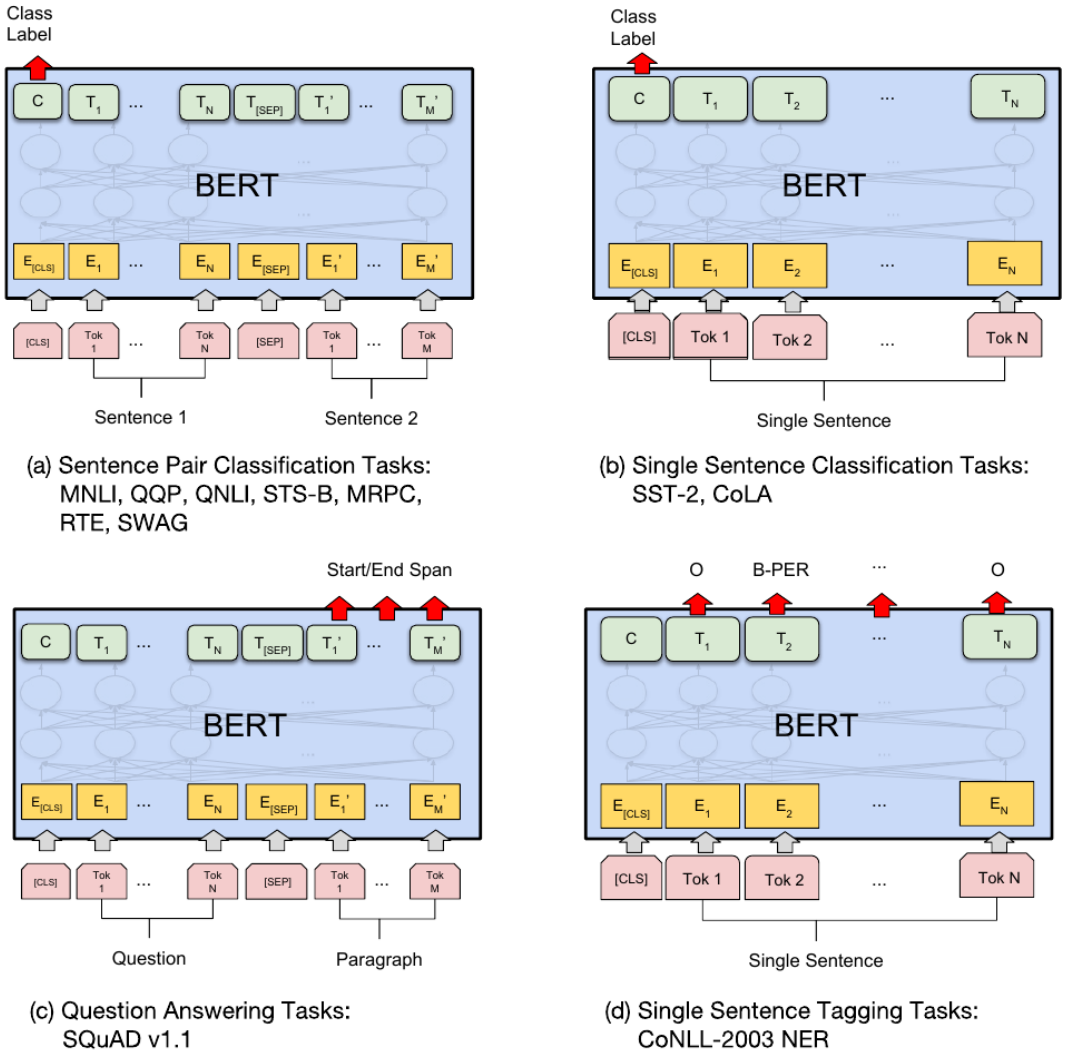

Class Label
C
T1
TN
T[SEP]
T1'
TM'
BERT
E[CLS]
E1
EN
E[SEP]
E1'
EM'
[CLS]
Tok 1
Tok N
[SEP]
Tok M
Sentence 1
Sentence 2
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG
Class Label
T2
E2
Tok 2
Single Sentence
(b) Single Sentence Classification Tasks: SST-2, CoLA
Start/End Span
Question
Paragraph
(c) Question Answering Tasks: SQuAD v1.1
O
B-PER
O
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Fig. 2.7. Different NLP tasks implemented by fine-tuning BERT [2].

Respect this work, both experiments correspond with classification of a single sentence, so the architecture employed is the one plotted in figure 2.7 (b).

2.4. Compression Techniques

There is a trend in the NLP research of continuously growing size of the pre-trained language models in order to get state-of-the-art results just by using more data and computing power [31]. Some of the latest examples are Turing-NLG [32] and GPT-3 [3].

Best performing NLP systems use huge neural networks, usually transformers or similar architectures to BERT, with millions of parameters. However, they are heavy and slow, highlighting two main problems:

1. Environmental cost of exponentially scaling these models' computational

requirements [33], [34].

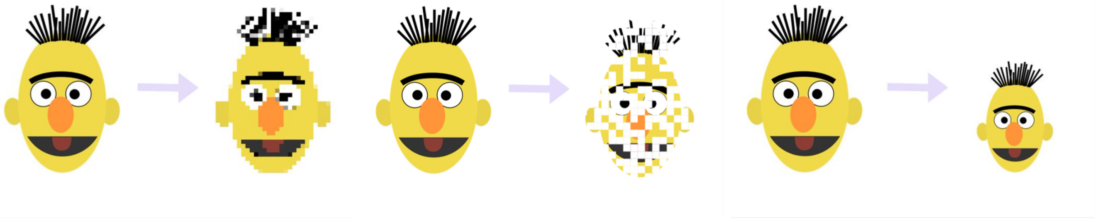
2. The difficulties to use these models on-device in real-time, due to the computational cost and memory requirements.

It is a fact the using much smaller language models can reach similar performances on many downstream-tasks requiring a smaller computational training and keeping the flexibility of larger ones, as the case of DistilBERT [4]. The point is applying the correct compression technique.

Compressing a model means reducing the number of parameters (weights) or their precision. The simplest way is compressing a model post training (before serving it). However, the network does not get a chance to recover after overly aggressive pruning and may significantly under perform. It is better to apply compression in small steps during training, giving the model a chance to recover by further learning from the data. Many compression methods can be applied both post-training and during training; post-training can be faster and often does not require training data, while during-training compression can preserve more accuracy and lead to higher compression rates [35].

There are three main techniques:

1. **Quantization:** decreasing the numerical precision of model's weights, employing less bits. Computer architecture usually work from 8 to 1 bits, although using only one bit (2 possible values) may decrease the precision of model too much. One example of quantization is presented in [36].
2. **Pruning:** removing parts of a model to make it smaller and faster. A popular technique is weight pruning [37] where individual connection weights close to zero are removed. When using self-attention layers, some heads can be dropped without significantly degrading the performance [38].
3. **Distillation:** a smaller model (student) is trained to mimic a larger model (teacher), its output or internal data representations. It leads to very straightforward improvements in both speed and size across different types of networks, being the main characteristic of DistilBERT [4].



(a) Quantization (b) Pruning (c) Distillation

Fig. 2.8. Compression techniques [35].

3. DISTILBERT

Presented in [4] as an alternative to continue training massive models to reach better performances [31]. DistilBERT is **40% smaller** and **60% faster** than BERT, while retaining **97% of its language understanding capabilities**. In addition, it is small enough to run on the edge, for example, on mobile devices.

The implementation is based on a smaller Transformer [6] pre-trained through distillation via the supervision of BERT (uncased version). Authors found beneficial to use a general-purpose pre-training distillation rather than task-specific [39], [40] or multi-distillation [41].

The code is adapted in part from Facebook XLM [42] and the PyTorch version of Google BERT [2] available in the Hugging Face repository [5].

3.1. Knowledge Distillation

Model compression method in which a small model is trained to mimic a pre-trained, larger model (or ensemble of models). This training setting is sometimes referred to as *teacher-student*, where the large model is the **teacher** and the small model is the **student** [43]. Knowledge distillation was first proposed in [44] and generalized by [45].

In supervised learning, a classification model is generally trained to predict a gold class by maximizing its probability (softmax of logits) using the log-likelihood signal. In many cases, a good performance model will predict an output distribution with the correct class having a high probability, leaving other classes with probabilities near zero. However, there are differences in the probabilities close to zero, reflecting how well the model generalizes, **dark knowledge**. Distillation prevents the model to be too sure about its prediction, similarly to label smoothing [46].

In order to explain how BERT is distilled it is followed the explanation provided in [47], [46] and [4].

The loss seeks to achieve two goals: minimize the loss on which the teacher has been pre-trained and mimic the teacher itself, which requires a mix of other two loss functions [47]. At the end, the training objective is a **linear combination of three losses**.

3.1.1. Masked Language Modeling Loss

The teacher, in this case BERT, has been fine-tuned using a determined loss function, for example the cross entropy loss in multiclass classification. As the student, DistilBERT, has the same architecture, the only difference is in the number of layers, it can be easily

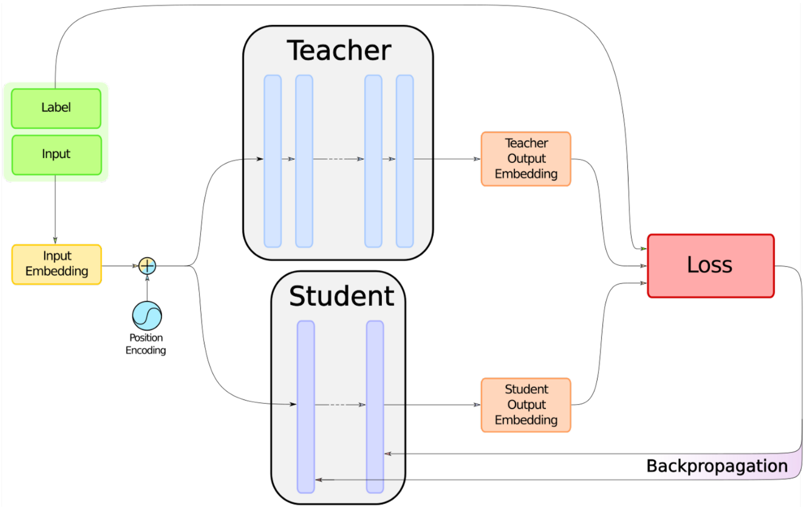


Fig. 3.1. Distillation of models similar to BERT [47].

compute the same loss.

Nevertheless, next sentence prediction is removed in the case of DistilBERT, only masked language modeling is employed with its corresponding loss L_{mlm}.

3.1.2. Teacher-Student Cross Entropy Loss

This one together with the next loss, aim to mimic the teacher. BERT's output are logits, which passed through a softmax layer are converted to a probability distribution. As mentioned, the result of this process is the same both teacher and student, so for an input x, teacher's output is:

$$T(x) = (t_1, \ldots, t_n) = \text{softmax}(\hat{t}_1, \ldots, \hat{t}_n) \tag{3.1}$$

And student's output:

$$S(x) = (s_1, \ldots, s_n) = \text{softmax}(\hat{s}_1, \ldots, \hat{s}_n) \tag{3.2}$$

Instead of training with a cross-entropy over the hard targets (one-hot encoding of the gold class), the knowledge is transferred from the teacher to the student with a cross-entropy over the soft targets (probabilities of the teacher) [46].

$$L_{ce} = -\sum_{i=1}^{n} t_i \cdot \log(s_i) \tag{3.3}$$

A critical point to highlight by the authors is that this loss is a richer training signal since a single example enforces much more constraint than a single hard target [46].

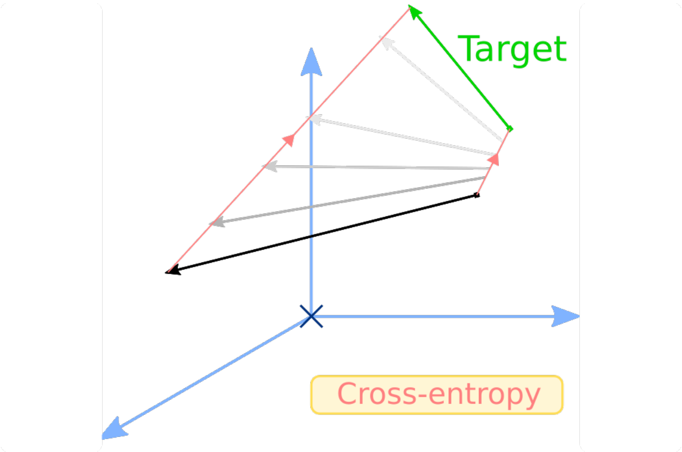


Fig. 3.2. Cross entropy loss effect on two 3D vectors [47].

Temperature

Following [45], it is applied a softmax temperature to control the smoothness of the output distribution [4]:

$$pi = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)} \tag{3.4}$$

Where z_i is the model score for class i. If $T \rightarrow 0$, the distribution becomes a Kronecker, which is equivalent to one-hot target vector, whereas $T \rightarrow \inf$, it becomes an uniform distribution [46].

In DistilBERT, both the student and the teacher's softmax are conditioned by the same temperature T during training, while it is set to 1 during inference [47].

3.1.3. Teacher-Student Cosine Loss

The second loss corresponds with a cosine which tries to align a vector to the target, without taking into account their respective norms or origin in space. It is defined as:

$$L_{cos} = 1 - \cos(T(x), S(x)) \tag{3.5}$$

The authors commented [4], [46] that it was beneficial to include it.

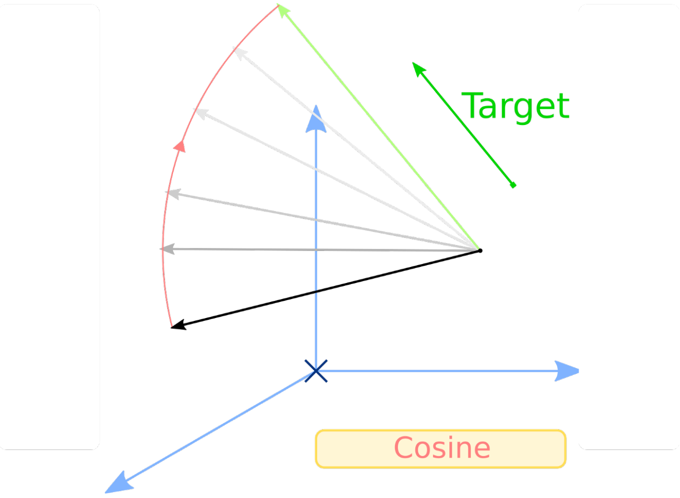


Fig. 3.3. Cosine loss effect on two 3D vectors [47].

Once the training loss is defined as a linear combination between L_{mlm}, L_{ce} and L_{cos}, the model can be trained. The point is that two BERT models have to run in parallel, but with the advantage that only the student applies backpropagation to adjust its weights [47].

3.2. Student Architecture and Initialization

DistilBERT has the same general architecture as BERT with minor changes:

- The token-type embeddings and the pooler at the end of the block are removed.
- The number of layers is reduced by a factor of 2.

In addition to that, the linear layers and layer normalization from the Transformer are highly optimized in modern linear algebra frameworks. The authors claim that variations on the last dimension of the tensor (hidden size) have a smaller impact on computation efficiency respect reducing the number of layers [4].

Another critical element during training is to find the right initialization for the student to converge [48]. DistilBERT takes out one layer out of two from BERT (figure 3.4), which seems to be the best heuristic according to [49].

With the publication of RoBERTa [50] it was shown that the way BERT is trained is crucial for its final performance. For this reason, following RoBERTa, DistilBERT is trained on very large batches leveraging gradient accumulation, with dynamic masking and removed the next sentence prediction objective, as mentioned before [46].

DistilBERT has been pre-trained with the same data as BERT, the BooksCorpus [27] and English Wikipedia (2,500M words), on eight 16GB V100 GPUs for approximately three and a half days [46].

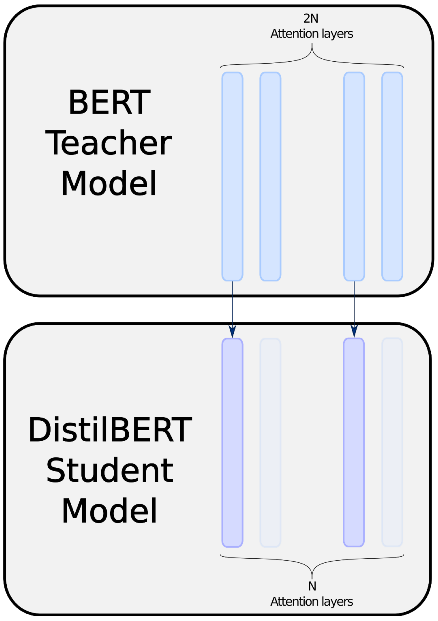


Fig. 3.4. Student model initialization [47].

3.3. Performance

In this section only main and relevant details are provided, if there is a interest in knowing more information, it is advisable to check the original publications [4], [46].

DistilBERT has been compared respect the General Language Understanding Evaluation (GLUE) benchmark in 9 tasks against two baselines: BERT base [2] and a non-transformer baseline composed by two BiLSTMs (bidirectional LSTM) on top of ELMo [23].

ELMo performance is reported by the authors, whereas BERT and DistilBERT results are the medians of 5 runs with different seeds. In table 3.1, the macro score, which is the average of 9 scores, is presented per model.

Model	Macro score
ELMo + BiLSTMs	68.7
BERT	79.5
DistilBERT	77.0

Table 3.1. Comparison of DistilBERT respect BERT and ELMo + BiLSTMs in the GLUE benchmark [4].

DistilBERT performs well respect the other two, improving the results provided by ELMo and retaining **97%** of BERT's performance. In addition to that, DistilBERT is significantly smaller (**40%**) while being much faster (**60%**) respect BERT, as it is shown in table 3.2. Mention that in device computation as a mobile application, DistilBERT becomes **71%** faster than BERT [4].

Model	Parameters (millions)	Inference time (seconds)
ELMo + BiLSTMs	180	895
BERT	110	668
DistilBERT	66	410

Table 3.2. Comparison of size and speed of DistilBERT, BERT and ELMo + BiLSTMs. Inference time of full pass of SST-B on CPU with batch size 1 [4].

An interesting study was done respect two **downstream tasks**: Internet Movie Database (IMDb) sentiment classification [51] and questions answering task with Stanford Question Answering Dataset (SQuAD) [29]. Authors of [4] tried to add another step of distillation by fine-tuning DistilBERT on SQuAD using a BERT previously fine-tuned on this task. There are two distillation steps, first during the pre-training and second in the fine-tuning.

Results are presented in table 3.3, where only BERT and DistilBERT are compared. DistilBERT(D) corresponds with the model which performs a second distillation step. In the IMDb dataset is measured the accuracy while in the case of SQuAD F1-score is compared. As it can be seen, both models perform similarly.

Model	IMDb (accuracy)	SQuAD (F1-score)
BERT	0.9346	88.5
DistilBERT	0.9282	85.8
DistilBERT(D)	-	86.9

Table 3.3. Comparison of two downstream tasks respect BERT and DistilBERT [4].

4. EXPERIMENTS

In order to demonstrate empirically the similar performance between BERT and DistilBERT as well as the elapsed time of both, two common NLP tasks are resolved: classification with **more than one class** and with **multiple labels**. As it has been mentioned in the introduction, these models are more used in language modeling, but due to the available computational power, only simple tasks can be implemented.

To implement the models is used Hugging Face versions [5] of BERT and DistilBERT developed with PyTorch, adding an own defined network in top of them. In both cases the uncased version is employed, what means that no difference is considered between lower and upper letters, since it eases the computation.

In spite of being much faster than pre-training, fine-tune a BERT model for any class could be slow for a CPU. Due to that, it is employed a GPU provided by Google Colaboratory: **Tesla P100-PCIE-16GB** . Mention that is not possible to select which GPU to use, so both experiments were run in the same execution ensuring to employ the same device.

4.1. Multiclass

One of the most common problems in NLP, where a given piece of text or sentence needs to be classified in one category of a given list. Each sample corresponds with only one class. It is used the News aggregator dataset by UCI Machine Learning Repository, composed by news headline which can be grouped into 4 categories:

- Business.
- Science.
- Entertainment.
- Health.

4.1.1. Data Pre-Processing

In addition to the common pre-processing process of cleaning and preparing the data before inputting to the model, BERT and DistilBERT require a specific format, according to the pre-trained vocabulary employed.

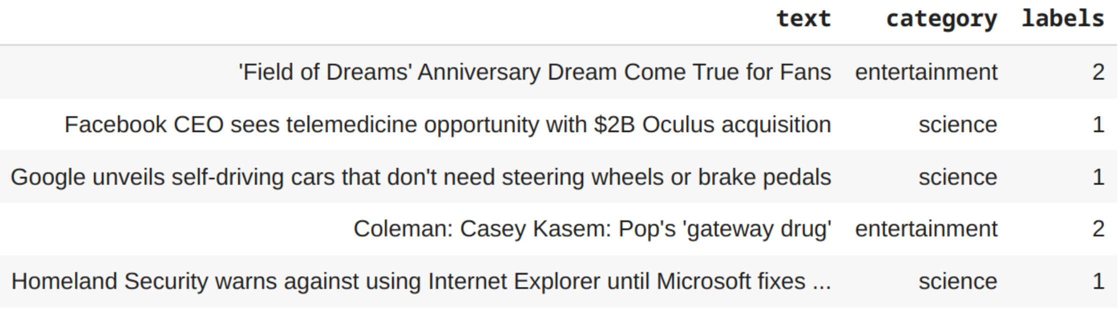
In both tasks the process is exactly the same, with a minor difference in the way of storing the labels.

Cleaning

Several steps are carried out to obtain the data in an useful format:

- Extract only the columns corresponding to title and category.
- Remove titles larger than 512 words (maximum length allowed by BERT and DistilBERT).
- Convert the categories (encoded as letters) to labels, from 0 to 3.
- Rename columns: useful for consistency with both tasks during pre-processing.

Finally, each sample is composed by text and a label, as it can be seen in figure 4.1.



text	category	labels
'Field of Dreams' Anniversary Dream Come True for Fans	entertainment	2
Facebook CEO sees telemedicine opportunity with $2B Oculus acquisition	science	1
Google unveils self-driving cars that don't need steering wheels or brake pedals	science	1
Coleman: Casey Kasem: Pop's 'gateway drug'	entertainment	2
Homeland Security warns against using Internet Explorer until Microsoft fixes ...	science	1

Fig. 4.1. Multiclass data after cleaning process.

Splitting

The data is split in three sets: training, validation and test. It is employed 80% of data as training and 20% of test. A 10% of training data is employed as validation. The total number of samples per set is presented in table 4.1.

Total samples	Training	Validation	Test
422,410	304,135	33,793	84,482

Table 4.1. Data split in multiclass classification.

It is important to know how many samples there are per class in the training set, in case an imbalanced classification problem is presented. The number of samples is going to depend on how the original dataset has been split. The partitions per class are presented in table 4.2.

In spite of having some differences between classes, the problem is not extremely imbalanced and it is not required any other pre-processing step to manage it.

Split	business	science	entertainment	health
training	83,462	77,918	110,014	32,741
validation	9,443	8,674	11,982	3,694
test	23,062	21,750	30,466	9,204

Table 4.2. Data split per class in multiclass classification.

Tokenization

Before input sequences into the models, it is necessary to preprocess data samples following next steps:

- Tokenize sequences, converting text into integers according to embedding space used.
- Pad each sentence to the maximum length which is in your batch.
- Truncate each sentence to the maximum length the model can accept.

BERT and DistilBERT employ WordPiece [26] subword tokenization algorithm. These algorithms rely on the principle that frequently used words should not be split into smaller subwords, but rare words should be decomposed into meaningful subwords. WordPiece is very similar to Byte Pair Encoding (BPE) [52]. Every character present in training data is included to learn the corresponding merge rules. In contrast to BPE that chooses the most frequent symbol pair, this algorithm maximizes the likelihood [53] of the training data when it is added to the vocabulary. This approach evaluates if it is worthy to merge two symbols, instead of separated.

The input in both models follow next scheme for classification:

[CLS] Sentence [SEP]

In order to preprocess according to the model, it is going to be used the BertTokenizerFast from Hugging Face, which ensures getting a tokenizer that corresponds to the desired model architecture. Besides that, it downloads the vocabulary used when pre-training this specific checkpoint. It is important to highlight that same tokenizer can be used in the case of DistilBERT, and it is implemented in that way to save repetitive code.

From tokenizer's instance it can be obtained information about BERT and DistilBERT. These values can be checked in table 4.3.

- Dimension of the vocabulary used
- Maximum allowed length of the model
- In which size is applied padding

- Special used tokens for separation token, padding token, classification token, mask token and other unkown tokens.

Attribute	Value
vocab size	30522
model max length	512
padding side	Right
sep token	[SEP]
pad token	[PAD]
cls token	[CLS]
mask token	[MASK]
unk token	[UNK]

Table 4.3. Parameters of BERT and DistilBERT.

BERT and DistilBERT work with sentences of same dimension. Due to that, it is required to pad (add padding tokens until a specified size) or truncate (separate a sentence if it is longer than the maximum model length). For this reason it is very useful to know which is the maximum length in the sentences, and limit the input size to this value.

Mention that sometimes it is recommended to set a length not very large in order to avoid high computational times. In this case, the maximum length with the provided data was **49 tokens**.

Batch Sizes

Finally, the models are going to be trained using PyTorch dataloaders, to save memory and boost the process, so it is required to specify the batch size in each of the splits. The selected values are presented in table 4.4.

Split	Batch size
Training	64
Validation	32
Test	32

Table 4.4. Selected batch sizes in multiclass classification.

4.1.2. Declaration of the Models

Classification layer weights are introduced $W \in \mathbb{R}^{K\ x\ H}$, where K is the number of labels. Then it is computed a standard classification loss with C and W:

$$\log(\text{softmax}(CW^T)) \tag{4.1}$$

The final models are just combinations of BERT or DistilBERT and a classifier to get the final probabilities, which corresponds with a simple neural network with only one hidden layer.

Classifier's hyper-parameters can be selected, in this case the number of neurons in the hidden layer of the neural network and the dropout probability, table 4.5. Same values are used both in BERT and DistilBERT.

Hyper-parameter	Value
Hidden size	50
Dropout probability	0.2

Table 4.5. Classifier's hyper-parameters in multiclass classification.

The total number of parameters is compared, checking that DistilBERT is **40% smaller** than BERT, fulfilling with the explanations given in the original publication.

4.1.3. Fine-Tuning

As usual, it is iterated through batches trying to reduce the **cross entropy loss** evaluating which is the most likely class between the possible 4. At the end of each epoch the model is tested respect the validation dataset, calculating the **accuracy**.

Mention AdamW optimizer is employed during the process.

Training Hyper-parameters

It can be selected as hyper-parameters the learning rate and epsilon of the optimizer, as well as the number of epochs. Mention than in these models with millions of parameters, these values are not going to be critical, excluding the number of epochs which can lead the model to overfit. Due to that, the models are stored after each epoch.

Selected hyper-parameters are presented in table 4.6. Same values are used both in BERT and DistilBERT.

Hyper-parameter	Value
Learning rate	5e-5
Epsilon	1e-8
Epochs	4

Table 4.6. Classifier's hyper-parameters in multiclass classification.

Training Results

After 4 epochs, it is studied the evolution of training and validation loss, accuracy and elapsed time of both BERT and DistilBERT, in table 4.7 and 4.8 respectively.

Epoch	train loss	val loss	acc	train time	val time	total time
1	0.191402	0.140939	0.953955	0:23:45	0:00:50	0:24:35
2	0.101084	0.128902	0.960376	0:23:43	0:00:50	0:49:08
3	0.064532	0.143761	0.962921	0:23:35	0:00:50	1:13:33
4	0.040264	0.171136	0.962803	0:23:34	0:00:50	1:37:57

Table 4.7. Training results of BERT in multiclass classification.

Epoch	train loss	val loss	acc	train time	val time	total time
1	0.192959	0.134109	0.953807	0:11:56	0:00:25	0:12:22
2	0.099991	0.124602	0.961353	0:11:56	0:00:25	0:24:44
3	0.064531	0.142409	0.962300	0:11:56	0:00:25	0:37:05
4	0.042051	0.158912	0.962063	0:11:56	0:00:25	0:49:27

Table 4.8. Training results of DistilBERT in multiclass classification.

DistilBERT is **50% faster** respect BERT, although the performance is slightly worse. Now the training and validation loss curves are plotted, as well as the validation accuracy, in order to have a better intuition.

First model corresponds with BERT, figures 4.2 and 4.3.

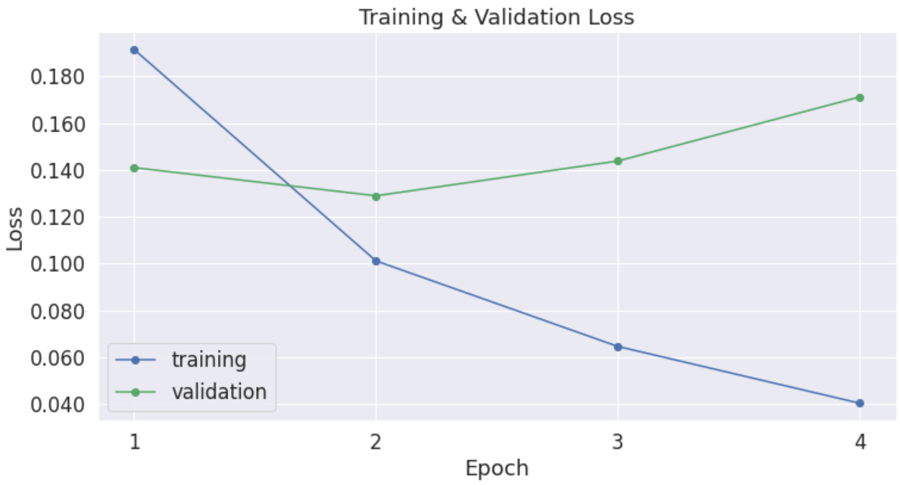


Fig. 4.2. Training and validation loss of BERT in multiclass classification

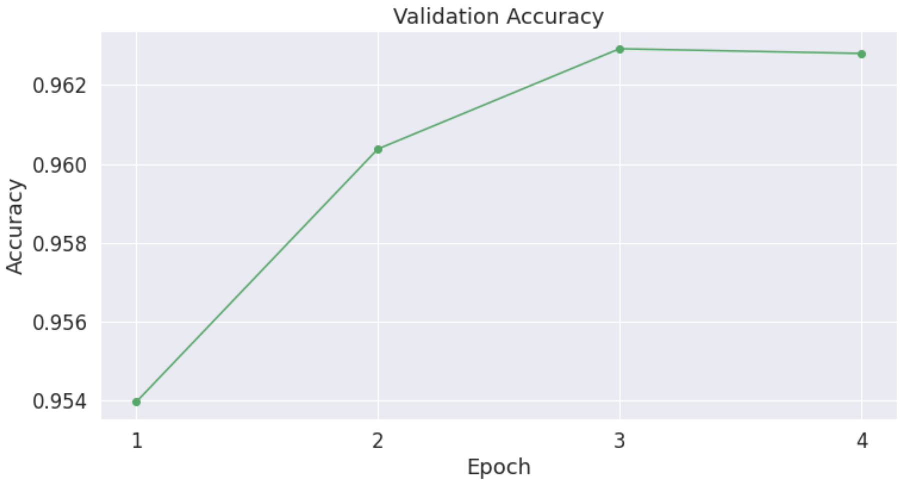


Fig. 4.3. Validation accuracy of BERT in multiclass classification

As it can be seen, the validation loss increases from the second epoch, whereas the training decreases, so the model may have been overfitted. In contrast, the validation accuracy improves until the third epoch, and gets a bit worse in the last one. For these reasons, it is not straightforward to determine which model is going to generalize better. Taking into account that the best accuracy has been obtained in the **third epoch**, and the validation loss is not much larger than in the second, the BERT saved in this epoch may be the best one.

Second model to analyze is DistilBERT, in figures 4.4 and 4.5.

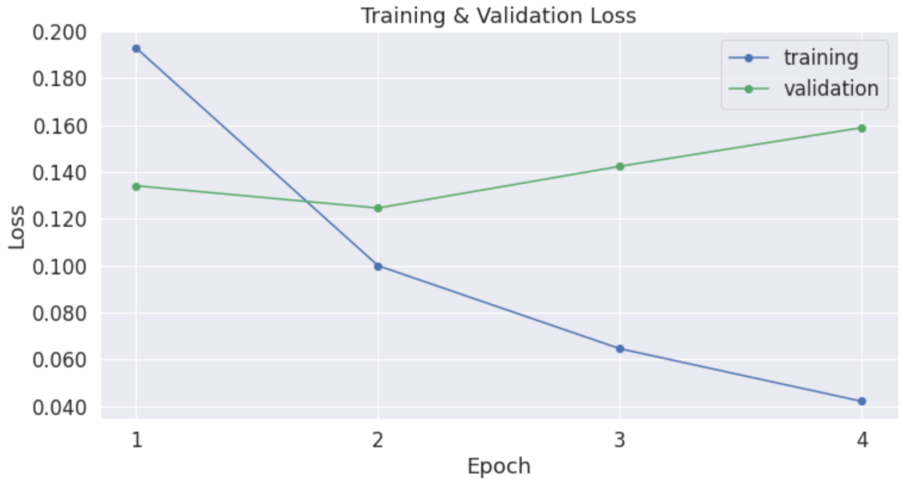


Fig. 4.4. Training and validation loss of DistilBERT in multiclass classification

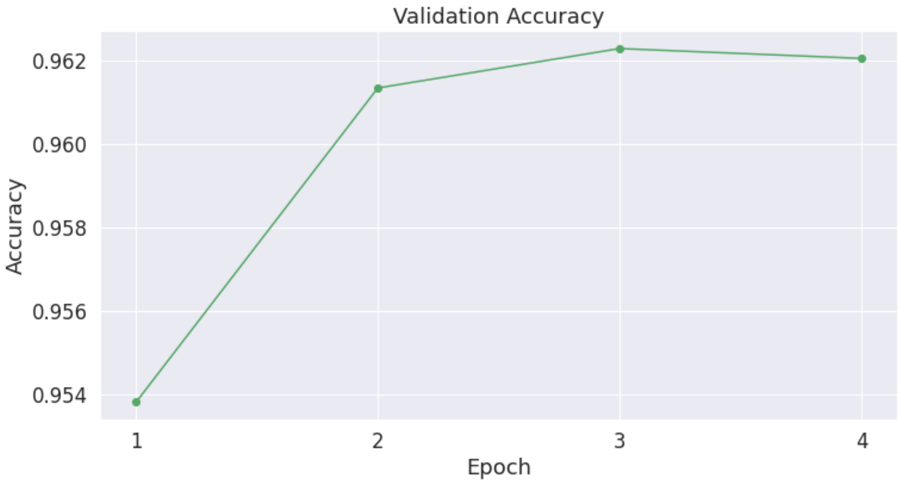


Fig. 4.5. Validation accuracy of DistilBERT in multiclass classification

Same explanation can be applied in this case, what it makes sense since both models behave in the same way, the only difference is in the computational cost, since the final classifier does not change between them. For this reason, the model trained until the **third epoch** may be the best option.

4.1.4. Evaluation

Once the models are fine-tuned, they are evaluated respect the test set. Nevertheless, first it is advisable to select which checkpoint (model trained until certain epoch) may generalise better. In both cases has been explained that the best model corresponds with **epoch 3**. The test results using these two models are presented in table 4.9.

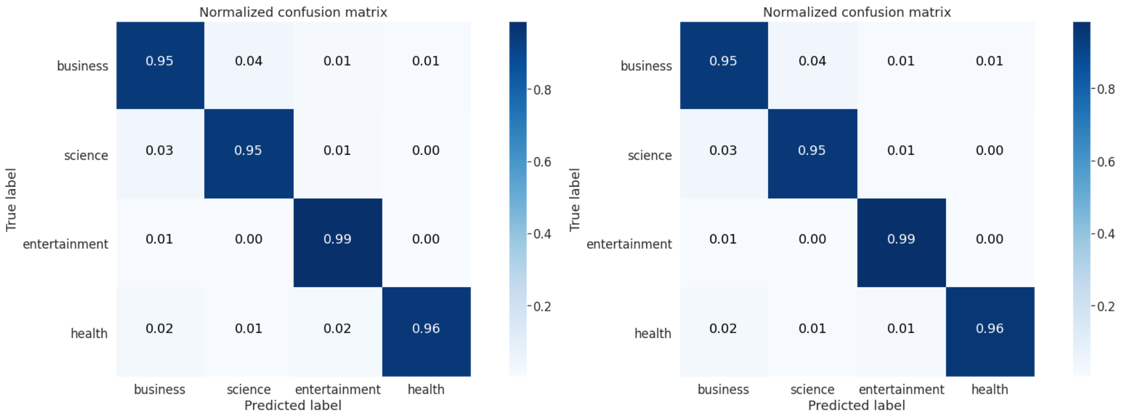
Model	loss	acc	time
BERT	0.132820	0.964205	0:02:05
DistilBERT	0.135481	0.963566	0:01:04

Table 4.9. Test results of BERT and DistilBERT in multiclass classification.

BERT obtains a very high accuracy, taking into account that the problem is multiclass, no binary. In the case of DistilBERT, the performance is also very high, although less accurate than BERT, **0.06%**, a negligible percentage. Mention than the evaluation loss is also higher, but again the difference is insignificant. What is really important is the difference in the computation time, where DistilBERT is again **50% faster** than BERT.

In order to see how good is predicted each category or class, the confusion matrix for both models is plotted.

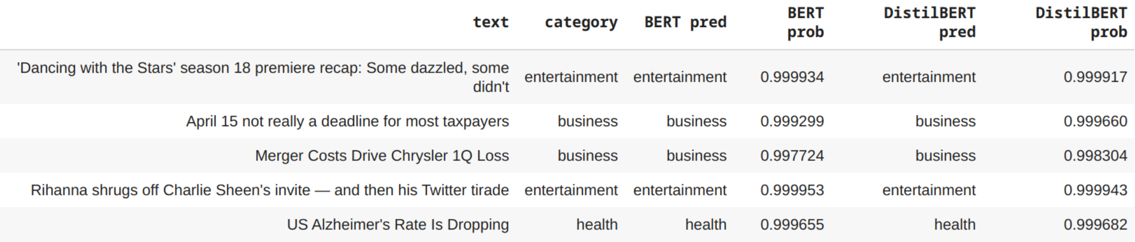
As expected, the performance is great and all the classes are well predicted with minor missclassifications with both models.



(a) BERT (b) DistilBERT

Fig. 4.6. Confusion matrices of BERT and DistilBERT respect the test set in multiclass classification.

Finally, in order to compare how sure both models make the predictions, a table with few test samples is plotted, with the obtained probabilities and predictions by model.



text	category	BERT pred	BERT prob	DistilBERT pred	DistilBERT prob
'Dancing with the Stars' season 18 premiere recap: Some dazzled, some didn't	entertainment	entertainment	0.999934	entertainment	0.999917
April 15 not really a deadline for most taxpayers	business	business	0.999299	business	0.999660
Merger Costs Drive Chrysler 1Q Loss	business	business	0.997724	business	0.998304
Rihanna shrugs off Charlie Sheen's invite — and then his Twitter tirade	entertainment	entertainment	0.999953	entertainment	0.999943
US Alzheimer's Rate Is Dropping	health	health	0.999655	health	0.999682

Fig. 4.7. Predictions on the multiclass test set with BERT and DistilBERT

As it can be seen, both BERT and DistilBERT are very accurate in the predictions, obtaining probabilities close to 1 for the correct category.

4.2. Multilabel

This task is similar to the previous one, but in this case more than one class can be assigned to a single sample. The dataset from Toxic Comment Classification Challenge is employed, which groups comments from Wikipedia's talk page edits. The possible categories are:

- Toxic.
- Severe toxic.
- Obscene.
- Threat.
- Insult.
- Identity hate.

4.2.1. Data Pre-Processing

The dataset is directly downloaded from Kaggle making use of its API, so it is required to use a Kaggle's token.

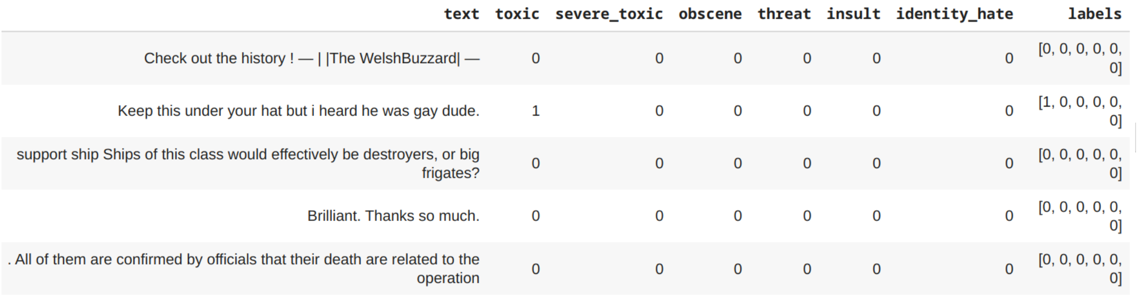
The original data is composed by text and one column per category, indicating if it is presented (1) or not (0) in the sentence.

Cleaning

The cleaning process consist of the following steps:

- Remove the *id* column.
- Remove titles larger than 200 word, because they had a bad format.
- Remove special characters as line break.
- List all the categories.
- Rename columns: as before, useful for consistency with both tasks during pre-processing.

Finally this is the used dataset, in which each sample is composed by text and the corresponding labels.



<table>
<tr><th>text</th><th>toxic</th><th>severe_toxic</th><th>obscene</th><th>threat</th><th>insult</th><th>identity_hate</th><th>labels</th></tr>
<tr><td>Check out the history ! — | |The WelshBuzzard| —</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>[0, 0, 0, 0, 0, 0]</td></tr>
<tr><td>Keep this under your hat but i heard he was gay dude.</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>[1, 0, 0, 0, 0, 0]</td></tr>
<tr><td>support ship Ships of this class would effectively be destroyers, or big frigates?</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>[0, 0, 0, 0, 0, 0]</td></tr>
<tr><td>Brilliant. Thanks so much.</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>[0, 0, 0, 0, 0, 0]</td></tr>
<tr><td>. All of them are confirmed by officials that their death are related to the operation</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>[0, 0, 0, 0, 0, 0]</td></tr>
</table>

Fig. 4.8. Multilabel data after cleaning process.

Splitting

The same strategy is followed to split the data in training, validation and test. 80% of the data is used in the training, where the 10% of it is employed as validation set. The other 20% of total data is set as test. The total number of samples per set is presented in table 4.10.

Total samples	Training	Validation	Test
78,186	56,294	6,255	15,637

Table 4.10. Data split in multilabel classification.

Mention that a considerable less amount of samples are used during this task. The problem of imbalanced classes may also be present, for this reason the partitions per class are presented in table 4.11.

Split	toxic	severe toxic	obscene	threat	insult	identity hate
training	7,297	853	4,209	236	3,943	691
validation	793	98	472	23	428	81
test	2,040	216	1,160	75	1,088	187

Table 4.11. Data split per class in multilabel classification.

As it can be seen, there are some categories as *severe toxic*, th*threat* or *identity hate*, whose representation is minor in the three sets. In spite of that, the imbalance is not strong enough to require some processing. In addition to that, BERT and DistilBERT work so well that they can lead with type of data.

Tokenization and Batch Sizes

The process is exactly the same than before, with the only difference that the labels are stored as *float*, instead of *long* as in the previous case. That is because different loss functions are employed, where in each one a certain type of label is required.

In this case the largest length was **145 tokens** and the same batch sizes were employed, table 4.4.

4.2.2. Declaration of the Models

In this case the models are exactly the same, it is only required changing the number of neurons of the output layer from the neural network employed as classifier. Now there are 6 classes or categories instead of 4. Again, this block is composed by a multilayer perceptron with only one hidden layer, using the same hyper-parameters defined in table 4.5.

The total number of parameters is compared again, checking that DistilBERT is **40% smaller** than BERT.

4.2.3. Fine-Tuning

There are two important differences respect the previous design:

1. Instead of using the cross entropy loss, it is employed the **binary cross entropy**, which evaluates each class independently of the rest.
2. The validation set is not evaluated respect the accuracy, but is used the **hamming score**, which is the fraction of correct predictions compared to the total labels. This is similar to accuracy, and in fact they are interchangeable. An own implemented function is used.

Same values are employed as training hyper-parameters, table 4.6.

Training Results

As before, it is employed 4 epochs in training. The results of BERT and DistilBERT are presented in table 4.12 and 4.13 respectively.

Epoch	train loss	val loss	hamm	train time	val time	total time
1	0.094784	0.048438	0.945196	0:11:46	0:00:25	0:12:11
2	0.045753	0.045861	0.934125	0:11:45	0:00:25	0:24:20
3	0.033957	0.046269	0.939297	0:11:44	0:00:25	0:36:29
4	0.026291	0.047548	0.942118	0:11:44	0:00:25	0:48:38

Table 4.12. Training results of BERT in multilabel classification.

Comparing the total time, DistilBERT is **50% faster** as before, and performs as well as BERT. As in the previous task, the training and validation loss curves are plotted, followed by the hamming score, which can be interpreted as the accuracy.

Epoch	train loss	val loss	hamm	train time	val time	total time
1	0.114079	0.047593	0.944439	0:05:56	0:00:13	0:06:09
2	0.049229	0.045350	0.937831	0:05:56	0:00:13	0:12:18
3	0.036902	0.045316	0.941386	0:05:56	0:00:13	0:18:27
4	0.029288	0.047954	0.941972	0:05:56	0:00:13	0:24:36

Table 4.13. Training results of DistilBERT in multilabel classification.

First model corresponds with BERT, figures 4.9 and 4.10.

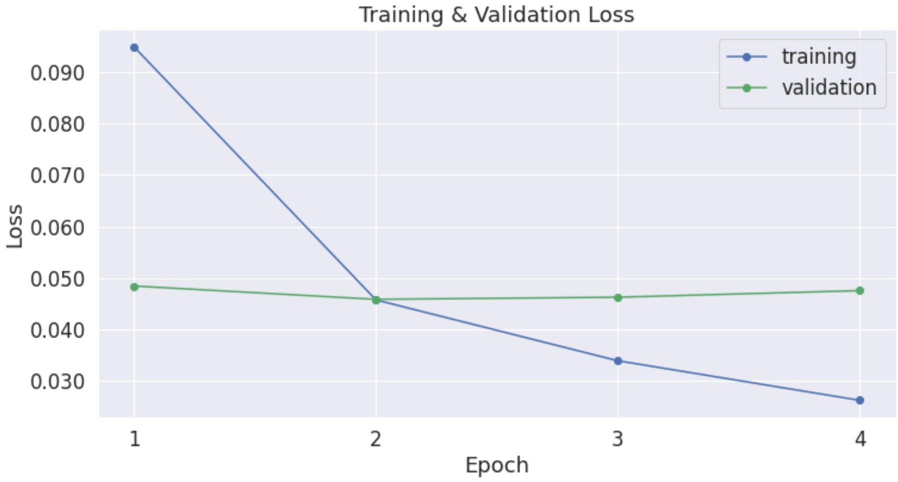


Fig. 4.9. Training and validation loss of BERT in multilabel classification

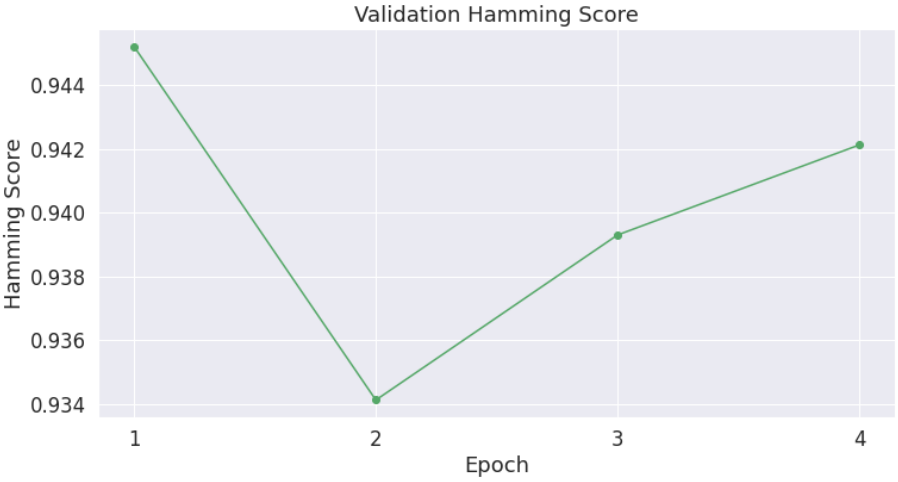


Fig. 4.10. Validation accuracy of BERT in multilabel classification

In this case the validation loss is more constant, leading to think that the model may not overfit. Nevertheless, the hamming score obtained in the last epoch is higher than in the third one, with a minimum increase of the loss. For these reasons, the best considered BERT model is the one of the **fourth epoch**.

Second model to analyze is DistilBERT, in figures 4.11 and 4.12.

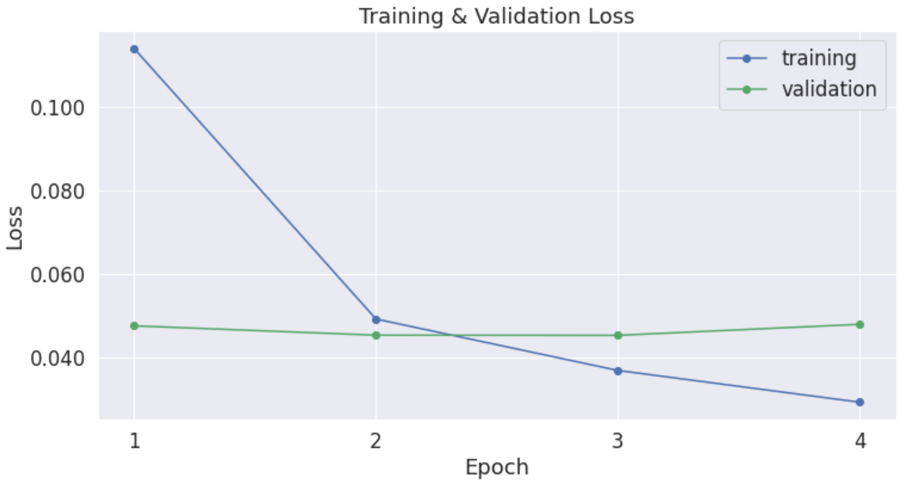


Fig. 4.11. Training and validation loss of DistilBERT in multilabel classification

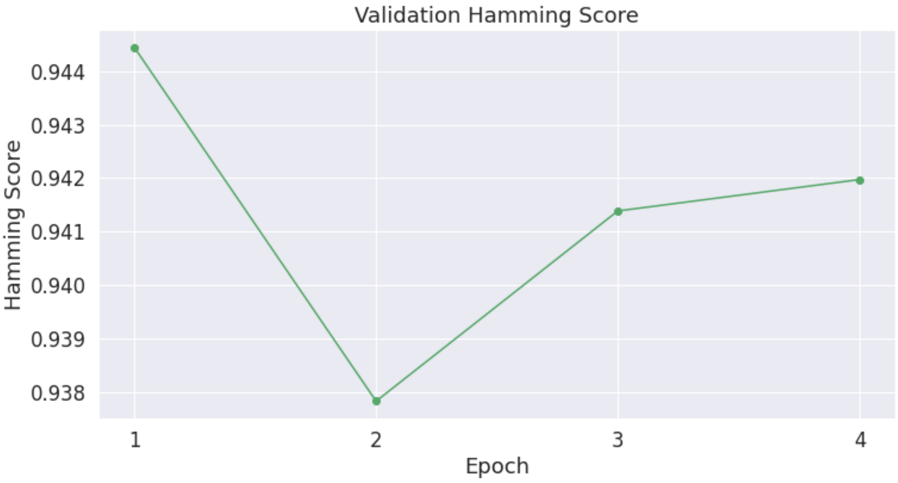


Fig. 4.12. Validation accuracy of DistilBERT in multilabel classification

Same situation than in the multiclass classification is presented here, the training results follow the same pattern in both models, what makes sense as explained before. Again, the best model corresponds with the **fourth epoch** in the case of DistilBERT.

4.2.4. Evaluation

As in the previous case, once the models are fine-tuned, they are evaluated respect the test set. In both cases has been explained that the best model corresponds with **epoch 4**. The test results using these two models are presented in table 4.14.

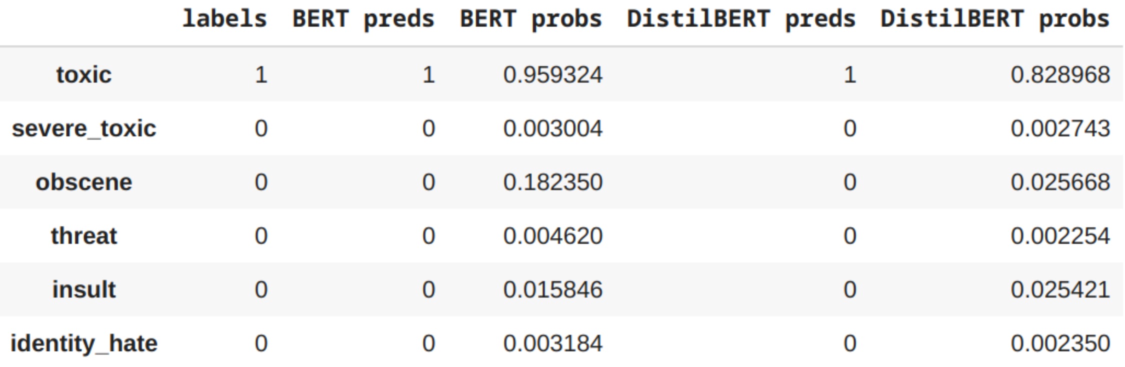
As in multiclass problem, both models perform very well respect the hamming score. Again, DistilBERT is significantly faster, but even a bit more accurate, **0.0007%**, than BERT. The corresponding loss is also smaller.

Model	loss	hamm	time
BERT	0.054059	0.939548	0:01:02
DistilBERT	0.052840	0.939555	0:00:32

Table 4.14. Test results of BERT and DistilBERT in multilabel classification.

Finally, in order to compare how sure both models make the predictions, a table with a test sample is printed, with the obtained probabilities and predictions by model. The sentence is the following:

```
, darn, darn.  keep this here and OFF my page or i’ll file another
complaint.  IM DONE!!!!  HE GONE!!!!  KA-BOOOOOOOM!!!!
```



	labels	BERT preds	BERT probs	DistilBERT preds	DistilBERT probs
toxic	1	1	0.959324	1	0.828968
severe_toxic	0	0	0.003004	0	0.002743
obscene	0	0	0.182350	0	0.025668
threat	0	0	0.004620	0	0.002254
insult	0	0	0.015846	0	0.025421
identity_hate	0	0	0.003184	0	0.002350

Fig. 4.13. Predictions on a multilabel test sample with BERT and DistilBERT.

As it can be seen, both BERT and DistilBERT are very accurate in the predictions, obtaining probabilities close to 1 or 0, when the category is attributed to sentence or not, respectively.

4.2.5. Optimal Threshold per Class

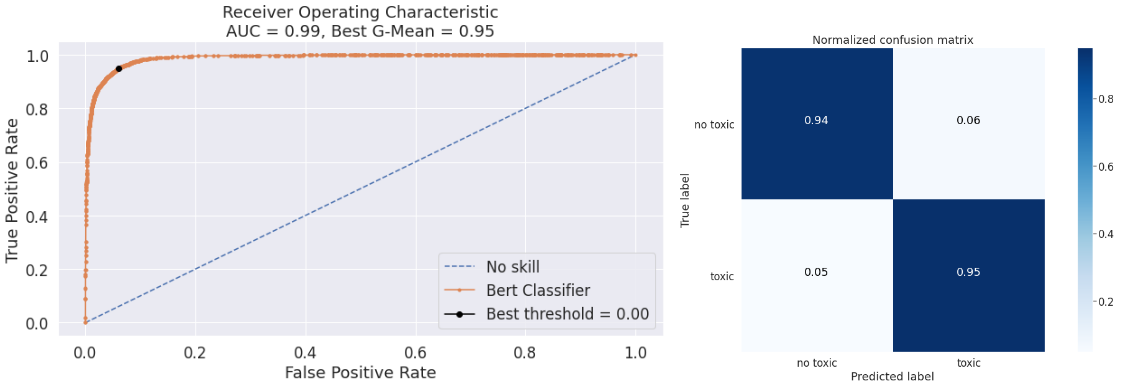
A multilabel problem can be seen as multiple binary classifications, one per category, where there may be an optimal threshold for each class.

The objective is to determine which is the optimal threshold in the 6 classes by model, BERT and DistilBERT. Nevertheless, mention this study is done with the test set and their corresponding labels, an information which is not available in a real situation. The approach will be analyzing the training data and hope that the found thresholds generalise well with new samples.

In order to determine which is the optimal threshold, the ROC curve is employed, calculating the **geometric mean** or **G-mean**, which will seek a balance between the sensitivity (true positive rate) and the specificity (the complement of the false positive rate).

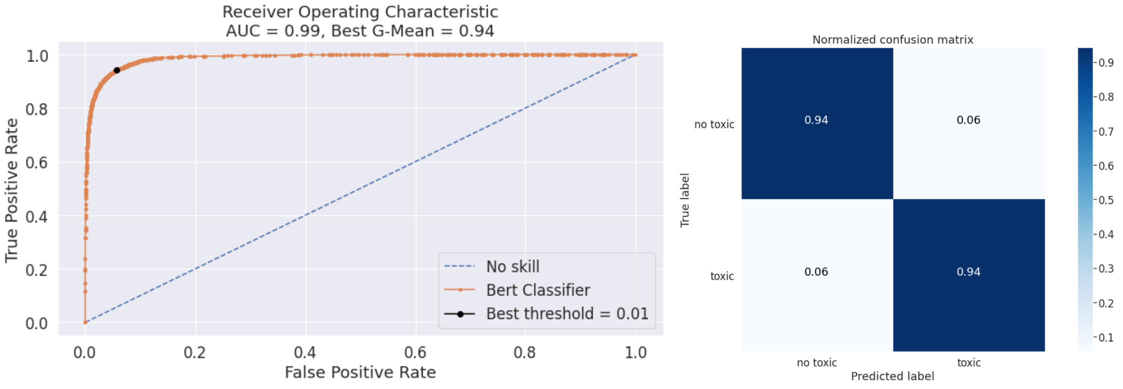
$$\text{G-Mean} = \sqrt{\text{Sensitivity} \cdot \text{Specificity}} = \sqrt{\text{TPR} \cdot (1 - \text{FPR})} \tag{4.2}$$

As there are 6 categories and the process is the same, only the plots corresponding to class *toxic* are plotted here, the rest can be found in appendix B.



(a) ROC curve (b) Confusion matrix

Fig. 4.14. ROC curve and confusion matrix respect class *toxic* using BERT.



(a) ROC curve (b) Confusion matrix

Fig. 4.15. ROC curve and confusion matrix respect class *toxic* using DistilBERT.

The optimal thresholds per class and model are the following:.

Model	toxic	severe toxic	obscene	threat	insult	identity hate
BERT	0.003730	0.012142	0.028351	0.009285	0.018270	0.011171
DistilBERT	0.013721	0.015630	0.040182	0.012213	0.007833	0.016469

Table 4.15. Optimal thresholds of all classes and by model based on test set.

As it can be seen, the results are similar in the 6 categories, in spite of the significant differences in the amount of samples. Both models perform very well predicting each of the classes, with a minor number of missclassifications. As it can be seen in table 4.15, the optimal thresholds are really close to 0. This may be due to the fact that it is better for these data to have a false positive instead of a false negative, in terms of maximizing the AUC.

Finally both models are evaluated again but employing the found thresholds to make the predictions.

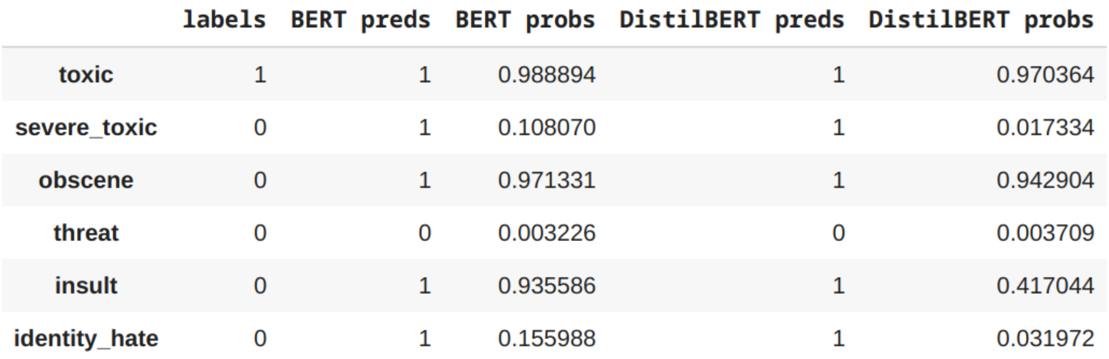
Model	loss	hamm	time
BERT	0.054059	0.891633	0:01:02
DistilBERT	0.052840	0.893641	0:00:32

Table 4.16. Test results of BERT and DistilBERT in multilabel classification employing optimal thresholds.

As it can be seen in table 4.16, the hamming score decreases in both cases. Consequently, it can be said that these thresholds improve the individual accuracy, but the general result gets worse, so it should be considered which is the main requirement in the implementation. Mention that the evaluation losses remain unchanged because they do not depend on the raw predictions.

To conclude, the new predictions are displayed for a test sample as before. The sentence is:

```
what ever you fuggin fag Question how did you know they were not
mine
```



	labels	BERT preds	BERT probs	DistilBERT preds	DistilBERT probs
toxic	1	1	0.988894	1	0.970364
severe_toxic	0	1	0.108070	1	0.017334
obscene	0	1	0.971331	1	0.942904
threat	0	0	0.003226	0	0.003709
insult	0	1	0.935586	1	0.417044
identity_hate	0	1	0.155988	1	0.031972

Fig. 4.16. Predictions on a multilabel test sample with BERT and DistilBERT employing optimal thresholds.

It can be seen how both models produce some false positives, due to the new thresholds, as with *severe toxic* and *identity hate*. Also mention that multilabel classification is more complex than multiclass, and more errors are expected, especially with categories with few samples.

5. CONCLUSION

5.1. Results

BERT is a powerful model able to reach high performances in a wide range of tasks, but it may be slow and inefficient in some contexts. DistilBERT provides the solution, a 40% smaller model which is able to obtain a performance very close respect its larger version.

As it has been shown, DistilBERT was 50% faster than BERT in both experiments. In the original work [4], authors claim that the difference in velocity raises to 60% and even 71% in edge applications.

The point is that the performance's difference in multiclass classification was minor, 0.06% less accurate on the part of DistilBERT, while in multilabel classification this one was even better, with a difference of 0.0007%. This improvements is negligible, the real success is being close to BERT's skills. In [4] DistilBERT retained 97% of the language understanding capabilities, respect an average on GLUE [28], which includes more complex tasks, so the obtained results are in line with the study.

In addition to the performance, another critical point to highlight is the use of knowledge distillation in BERT. This technique can be applied to similar models which have the same problems respect computation and size. It is one alternative to consider in the face of increasing the number of parameters to obtain new models [31], or at least, to reduce the size of massive implementations employed in the industry.

5.2. Future Work

Respect this work, more complex task may be implemented, although it is difficult due to the computational resources. For example, question answering was developed in the bachelor thesis [1] using BERT (base version), so it could be compared with a DistilBERT version.

Due to the possibility of applying distillation in other models, it may be worthwhile perform a similar study with other huge architectures as GPT-3 [3] or Turing-NLG [32]. It is known [47] that there are already some cases [49], [54].

BIBLIOGRAPHY

[1] I. Bueno, “Inside a NLP multitasking neural network. BERT, one model to rule them all,” Master’s thesis, University Carlos III of Madrid, Madrid, Spain, Jun. 2021. [Online]. Available: https://github.com/ion-bueno/bert-from-inside.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” *Computing Research Repository*, vol. abs/1810.04805, arXiv:1810.04805, May 2019. [Online]. Available: https://arxiv.org/abs/1810.04805 (visited on 06/22/2022).

[3] T. B. Brown *et al.*, “Language models are few-shot learners,” *Computing Research Repository*, vol. abs/2005.14165, arXiv:2005.14165, Jul. 2020. [Online]. Available: https://arxiv.org/abs/2005.14165 (visited on 06/22/2022).

[4] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter,” *Computing Research Repository*, vol. abs/1910.01108, arXiv:1910.01108, Mar. 2020. [Online]. Available: https://arxiv.org/abs/1910.01108 (visited on 06/22/2022).

[5] T. Wolf *et al.*, “Huggingface’s transformers: State-of-the-art natural language processing,” *Computing Research Repository*, vol. abs/1910.03771, arXiv:1910.03771, Jul. 2020. [Online]. Available: https://arxiv.org/abs/1910.03771 (visited on 06/22/2022).

[6] A. Vaswani *et al.*, “Attention is all you need,” *Computing Research Repository*, vol. abs/1706.03762, arXiv:1706.03762, Dec. 2017. [Online]. Available: https://arxiv.org/abs/1706.03762 (visited on 06/22/2022).

[7] G. Chowdhury, “Natural language processing,” *Annual Review of Information Science and Technology*, vol. 37, no. 1, pp. 51–89, Jan. 2005. [Online]. Available: https://doi.org/10.1002/aris.1440370103 (visited on 06/22/2022).

[8] L. Wu, S. C. H. Hoi, and N. Yu, “Semantics-preserving bag-of-words models and applications,” *IEEE Transactions on Image Processing*, vol. 19, no. 7, pp. 1908–1920, Oct. 2009. [Online]. Available: https://doi.org/10.1145/1631058.1631064 (visited on 06/22/2022).

[9] H. Christian, M. P. Agus, and D. Suhartono, “Single document automatic text summarization using term frequency-inverse document frequency (tf-idf),” *ComTech: Computer, Mathematics and Engineering Applications*, vol. 7, no. 4, pp. 285–294, Dec. 2016. [Online]. Available: https://journal.binus.ac.id/index.php/comtech/article/view/3746 (visited on 06/22/2022).

[10] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, “On the properties of neural machine translation: Encoder-decoder approaches,” *Computing Research Repository*, vol. abs/1409.1259, arXiv:1409.1259, Oct. 2014. [Online]. Available: https://arxiv.org/abs/1409.1259 (visited on 06/22/2022).

[11] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, pp. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735 (visited on 06/22/2022).

[12] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, Mar. 1994. [Online]. Available: https://doi.org/10.1109/72.279181 (visited on 06/22/2022).

[13] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” May 7, 2015. [Online]. Available: http://arxiv.org/abs/1409.0473.

[14] Y. Kim, C. Denton, L. Hoang, and A. M. Rush, “Structured attention networks,” *Computing Research Repository*, vol. abs/1702.00887, arXiv:1702.00887, Feb. 2017. [Online]. Available: https://arxiv.org/abs/1702.00887 (visited on 06/22/2022).

[15] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*, 1st ed. Kitchener, Canada: D2L, 2020. [Online]. Available: https://d2l.ai.

[16] B. Hoover, H. Strobelt, and S. Gehrmann, “ExBERT: A visual analysis tool to explore learned representations in transformer models,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Online, Jul. 5, 2020. [Online]. Available: https://www.aclweb.org/anthology/2020.acl-demos.22.

[17] R. Paulus, C. Xiong, and R. Socher, “A deep reinforced model for abstractive summarization,” *Computing Research Repository*, vol. abs/1705.04304, arXiv:1705.04304, May 2017. [Online]. Available: https://arxiv.org/abs/1705.04304 (visited on 06/22/2022).

[18] J. Cheng, L. Dong, and M. Lapata, “Long short-term memory-networks for machine reading,” in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Austin, United States, Nov. 1, 2016. [Online]. Available: http://dx.doi.org/10.18653/v1/D16-1053.

[19] Z. Lin *et al.*, *A structured self-attentive sentence embedding*, Mar. 2017. [Online]. Available: https://arxiv.org/abs/1703.03130 (visited on 06/22/2022).

[20] A. Parikh, O. Täckström, D. Das, and J. Uszkoreit, “A decomposable attention model for natural language inference,” in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Austin, United States, Nov. 1, 2016. [Online]. Available: http://dx.doi.org/10.18653/v1/D16-1244.

[21] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Las Vegas, United States, Jun. 27, 2016. [Online]. Available: https://doi.org/10.1109/CVPR.2016.90.

[22] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *Computing Research Repository*, vol. abs/1607.06450, arXiv:1607.06450, Jul. 2016. [Online]. Available: https://arxiv.org/abs/1607.06450 (visited on 06/19/2021).

[23] M. E. Peters *et al.*, “Deep contextualized word representations,” in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, New Orleans, United States, Jun. 1, 2018. [Online]. Available: http://dx.doi.org/10.18653/v1/N18-1202.

[24] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. (Jun. 11, 2018). Improving language understanding with unsupervised learning, OpenAI, [Online]. Available: https://openai.com/blog/language-unsupervised/ (visited on 06/22/2022).

[25] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush, “OpenNMT: Open-source toolkit for neural machine translation,” in *Proceedings of ACL 2017, System Demonstrations*, Vancouver, Canada, Jul. 1, 2017. [Online]. Available: https://www.aclweb.org/anthology/P17-4012.

[26] M. Schuster and K. Nakajima, “Japanese and korean voice search,” in *International Conference on Acoustics, Speech and Signal Processing*, Kyoto, Japan, Mar. 25, 2012. [Online]. Available: https://doi.org/10.1109/ICASSP.2012.6289079.

[27] Y. Zhu *et al.*, “Aligning books and movies: Towards story-like visual explanations by watching movies and reading books,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, Dec. 7, 2015. [Online]. Available: https://doi.org/10.1109/ICCV.2015.11.

[28] A. Wang *et al.*, “GLUE: A multi-task benchmark and analysis platform for natural language understanding,” in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, Brussels, Belgium, Nov. 1, 2018. [Online]. Available: http://dx.doi.org/10.18653/v1/W18-5446.

[29] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, “SQuAD: 100,000+ questions for machine comprehension of text,” in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Austin, United States, Nov. 1, 2016. [Online]. Available: http://dx.doi.org/10.18653/v1/D16-1264.

[30] R. Zellers, Y. Bisk, R. Schwartz, and Y. Choi, "SWAG: A large-scale adversarial dataset for grounded commonsense inference," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, Oct. 1, 2018. [Online]. Available: http://dx.doi.org/10.18653/v1/D18-1009.

[31] A. Rogers. (Jul. 22, 2019). How the transformers broke NLP leaderboards, Hacking Semantics, [Online]. Available: https://hackingsemantics.xyz/2019/leaderboards/ (visited on 06/20/2021).

[32] C. Rosset. (Feb. 13, 2020). Turing-NLG: A 17-billion-parameter language model by microsoft, Microsoft Research Blog, [Online]. Available: https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/ (visited on 06/20/2021).

[33] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," *Commun. ACM*, vol. 63, no. 12, pp. 54–63, Nov. 2020. [Online]. Available: https://doi.org/10.1145/3381831 (visited on 06/22/2022).

[34] A. M. Emma Strubell Ananya Ganesh, "Energy and policy considerations for deep learning in NLP," *Computing Research Repository*, vol. abs/1906.02243, arXiv:1906.02243, Jun. 2019. [Online]. Available: http://arxiv.org/abs/1906.02243 (visited on 06/22/2022).

[35] S. Sucik. (Aug. 8, 2019). Compressing bert for faster prediction, RASA, [Online]. Available: https://rasa.com/blog/compressing-bert-for-faster-prediction-2/#motivation (visited on 06/22/2022).

[36] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on Machine Learning*, Lille, France, Jul. 6, 2015. [Online]. Available: https://proceedings.mlr.press/v37/gupta15.html.

[37] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," *Computing Research Repository*, vol. abs/1902.09574, arXiv:1902.09574, Feb. 2019. [Online]. Available: https://arxiv.org/abs/1902.09574 (visited on 06/22/2022).

[38] P. Michel, O. Levy, and G. Neubig, "Are sixteen heads really better than one?" In *Advances in Neural Information Processing Systems*, Vancouver, Canada, Dec. 8, 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/2c601ad9d2ff9bc8b282670cdd54f69f-Paper.pdf.

[39] R. Tang *et al.*, "Distilling task-specific knowledge from bert into simple neural networks," *Computing Research Repository*, vol. abs/1903.12136, arXiv:1903.12136, Mar. 2019. [Online]. Available: https://arxiv.org/abs/1903.12136 (visited on 06/22/2022).

[40] D. Chatterjee, “Making neural machine reading comprehension faster,” *Computing Research Repository*, vol. abs/1904.00796, arXiv1904.00796, Mar. 2019. [Online]. Available: https://arxiv.org/abs/1904.00796 (visited on 06/22/2022).

[41] Z. Yang, L. Shou, M. Gong, W. Lin, and D. Jiang, “Model compression with multi-task knowledge distillation for web-scale question answering system,” *Computing Research Repository*, vol. abs/1904.09636, arXiv:1904.09636, Apr. 2019. [Online]. Available: https://arxiv.org/abs/1904.09636 (visited on 06/22/2022).

[42] A. Conneau *et al.*, “Unsupervised cross-lingual representation learning at scale,” *Computing Research Repository*, vol. abs/1911.02116, no. arXiv:1911.02116, Jan. 2019. [Online]. Available: https://github.com/facebookresearch/XLM (visited on 06/22/2022).

[43] N. Zmora, G. Jacob, L. Zlotnik, B. Elharar, and G. Novik, “Neural network distiller: A python package for dnn compression research,” *Computing Research Repository*, vol. abs/1910.12232, arXiv:1910.12232, Oct. 2019. [Online]. Available: https://arxiv.org/abs/1910.12232 (visited on 06/22/2022).

[44] C. Buciluundefined, R. Caruana, and A. Niculescu-Mizil, “Model compression,” in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Philadelphia, United States, Aug. 1, 2006. [Online]. Available: https://doi.org/10.1145/1150402.1150464.

[45] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *Computing Research Repository*, vol. abs/1503.02531, arXiv:1503.02531, Mar. 2015. [Online]. Available: https://arxiv.org/abs/1503.02531 (visited on 06/22/2022).

[46] V. Sanh. (Aug. 1, 2019). Smaller, faster, cheaper, lighter: Introducing distilBERT, a distilled version of BERT, Medium, [Online]. Available: https://medium.com/huggingface/distilbert-8cf3380435b5 (visited on 06/22/2022).

[47] R. Ouazan. (Dec. 10, 2021). Distillation of bert-like models: The theory, Medium, [Online]. Available: https://towardsdatascience.com/distillation-of-bert-like-models-the-theory-32e19a02641f (visited on 06/22/2022).

[48] J. Frankle and M. Carbin, “The lottery ticket hypothesis: Finding sparse, trainable neural networks,” *Computing Research Repository*, vol. abs/1803.03635, arXiv:1803.03635, Mar. 2019. [Online]. Available: https://arxiv.org/abs/1803.03635 (visited on 06/22/2022).

[49] X. Jiao *et al.*, “TinyBERT: Distilling BERT for natural language understanding,” *Computing Research Repository*, vol. abs/1909.10351, arXiv:1909.10351, Oct. 2020. [Online]. Available: https://arxiv.org/abs/1909.10351 (visited on 06/22/2022).

[50] Y. Liu *et al.*, “RoBERTa: A robustly optimized BERT pretraining approach,” *Computing Research Repository*, vol. abs/1907.11692, arXiv:1907.11692, Jul. 2019. [Online]. Available: https://arxiv.org/abs/1907.11692 (visited on 06/22/2022).

[51] A. Maas *et al.*, “Learning word vectors for sentiment analysis,” in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, Portland, United States, Jun. 19, 2011. [Online]. Available: https://aclanthology.org/P11-1015.pdf.

[52] R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with subword units,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, Berlin, Germany, Aug. 1, 2016. [Online]. Available: http://dx.doi.org/10.18653/v1/P16-1162.

[53] L. Le Cam, “Maximum likelihood: An introduction,” *International Statistical Review/Revue Internationale de Statistique*, vol. 58, no. 2, pp. 153–171, Aug. 1990. [Online]. Available: https://doi.org/10.2307/1403464 (visited on 06/20/2021).

[54] Z. Sun *et al.*, “MobileBERT: A compact task-agnostic BERT for resource-limited devices,” *Computing Research Repository*, vol. abs/2004.02984, arXiv:2004.02984, Apr. 2020. [Online]. Available: https://arxiv.org/abs/2004.02984 (visited on 06/22/2022).

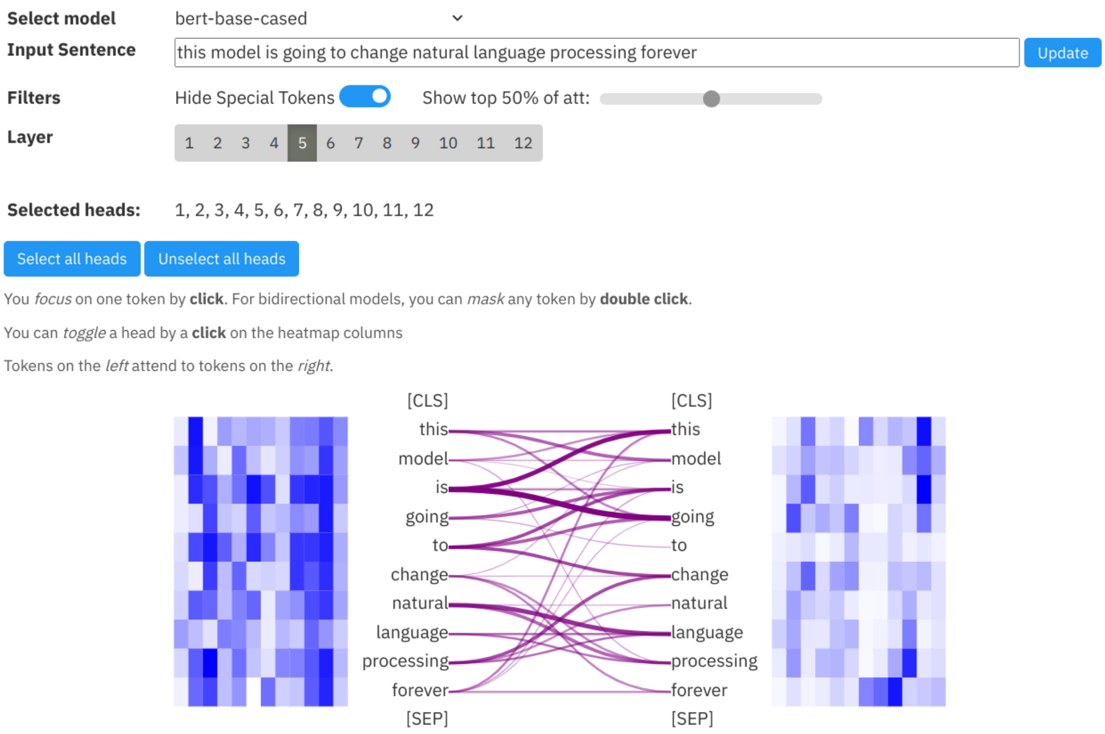
A. SELF-ATTENTION VISUALIZATION

As it has been explained with BERT and consequently DistilBERT in sections 2.3 and 3, the models contains a number of blocks which compose the encoder as attention heads in each of them.

Self-attention operations are not straightforward, even further if we have different blocks and many heads into them. To get an intuition about BERT and DistilBERT (and even other BERT-like models) respect the attention weights between elements, exBERT [16] can be used following next steps:

1. Select a model and introduce the input sentence.
2. Choose the percentage of attention shown. Special tokens can be hidden.
3. Select an encoder block (as *Layer*) and the attention heads that want to be visualized.

One example of the application is shown in figures A.1 and A.2, with BERT and DistilBERT respectively.



Select model bert-base-cased

Input Sentence this model is going to change natural language processing forever

Filters Hide Special Tokens ● Show top 50% of att:

Layer 5

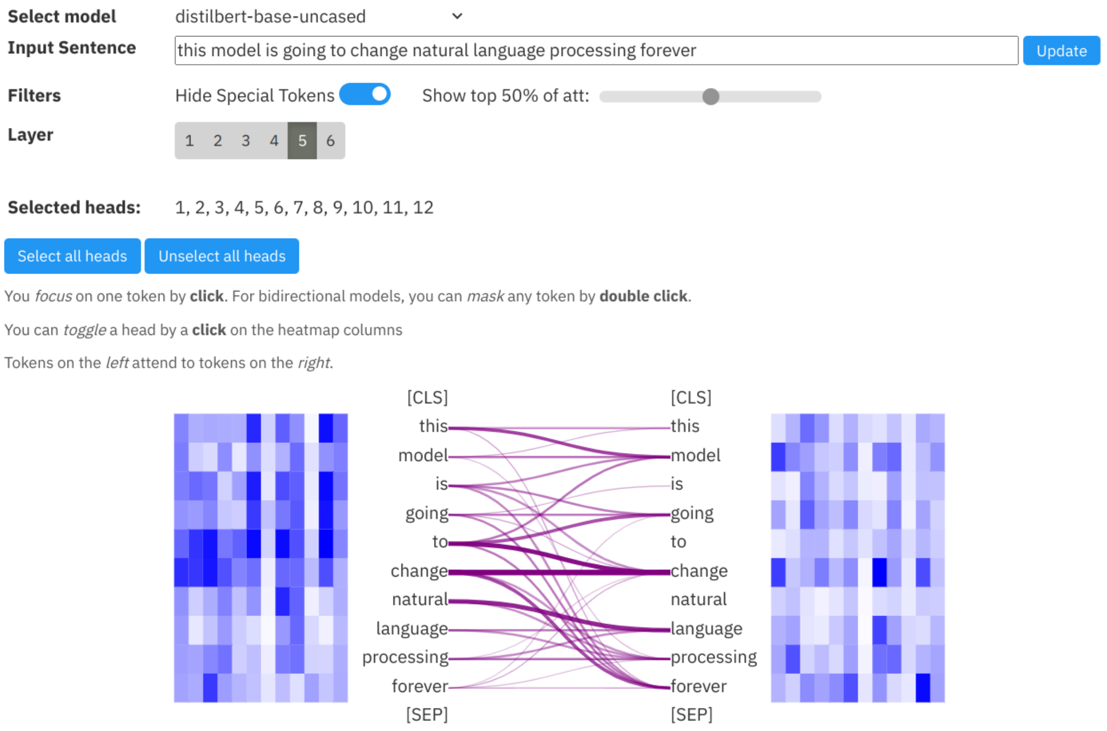
Selected heads: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

You *focus* on one token by **click**. For bidirectional models, you can *mask* any token by **double click**.

You can *toggle* a head by a **click** on the heatmap columns

Tokens on the *left* attend to tokens on the *right*.

Fig. A.1. Example of exBERT visualization with BERT.



Select model distilbert-base-uncased

Input Sentence this model is going to change natural language processing forever

Filters Hide Special Tokens Show top 50% of att:

Layer 1 2 3 4 5 6

Selected heads: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

You *focus* on one token by **click**. For bidirectional models, you can *mask* any token by **double click**.

You can *toggle* a head by a **click** on the heatmap columns

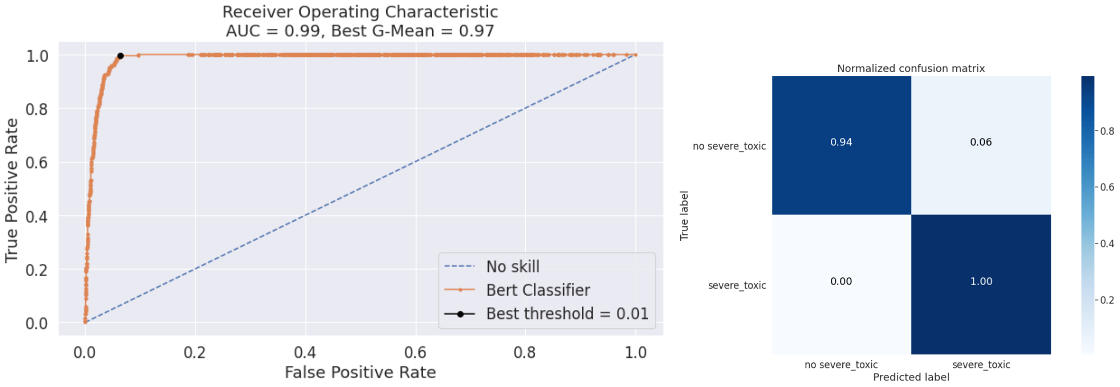
Tokens on the *left* attend to tokens on the *right*.

Fig. A.2. Example of exBERT visualization with DistilBERT.

B. OPTIMAL THRESHOLDS IN MULTILABEL CLASSIFICATION

Severe toxic

BERT.

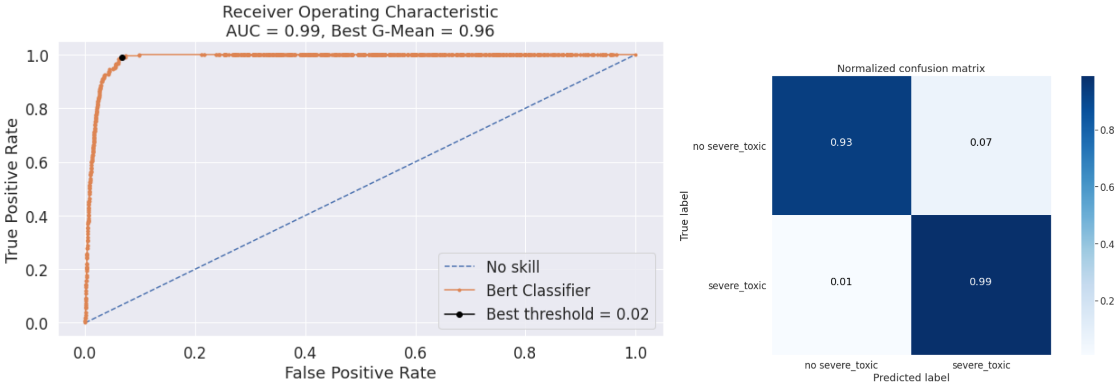


(a) ROC curve

(b) Confusion matrix

Fig. B.1. ROC curve and confusion matrix respect class *severe toxic* using BERT.

DistilBERT.

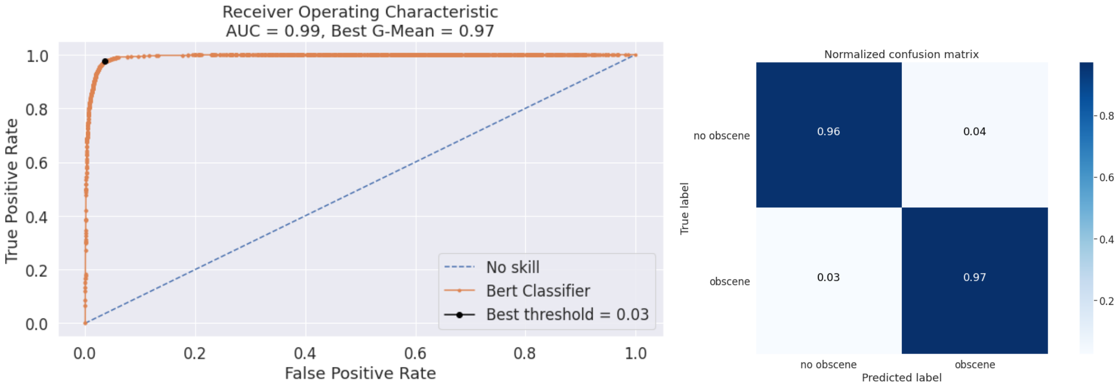


(a) ROC curve

(b) Confusion matrix

Fig. B.2. ROC curve and confusion matrix respect class *severe toxic* using DistilBERT.

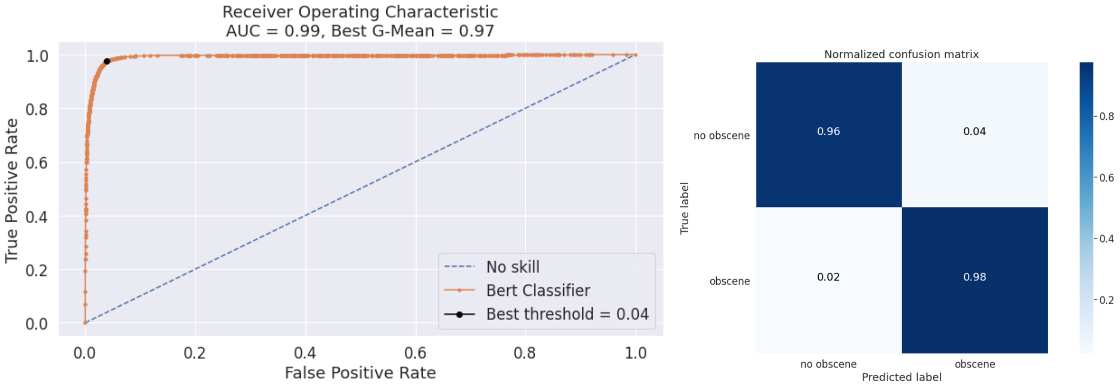
Obscene



(a) ROC curve (b) Confusion matrix

Fig. B.3. ROC curve and confusion matrix respect class *obscene* using BERT.

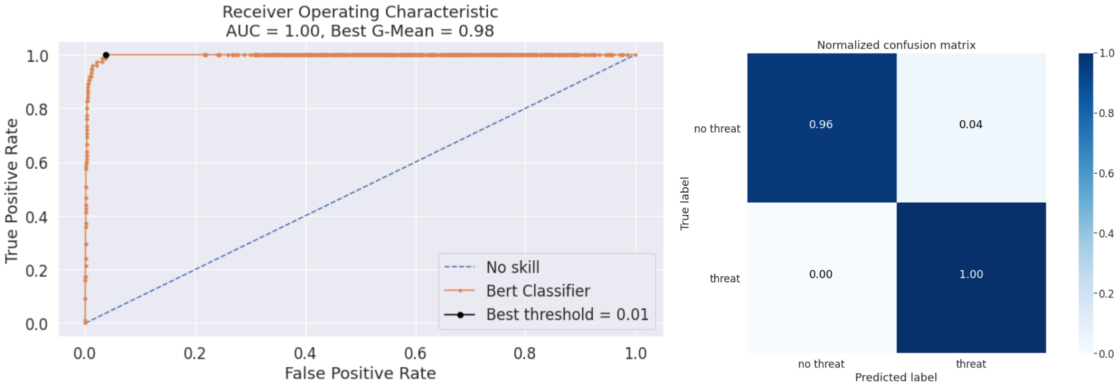
DistilBERT.



(a) ROC curve (b) Confusion matrix

Fig. B.4. ROC curve and confusion matrix respect class *obscene* using DistilBERT.

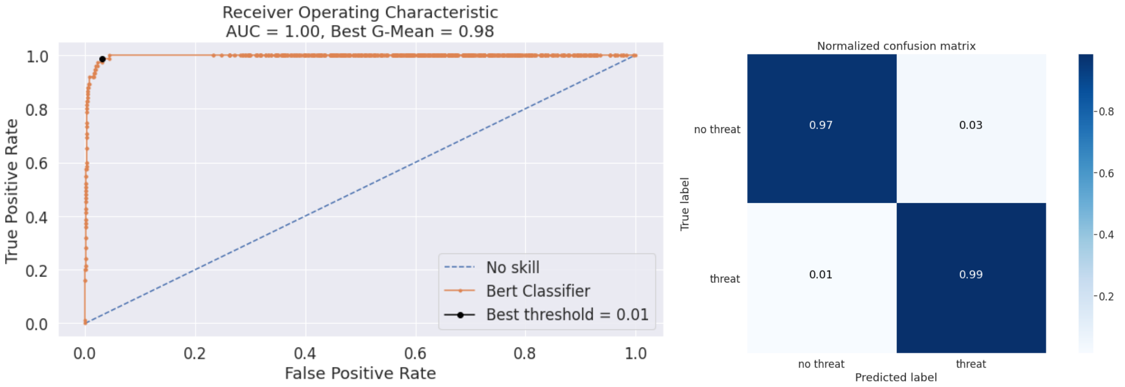
Threat



(a) ROC curve (b) Confusion matrix

Fig. B.5. ROC curve and confusion matrix respect class *threat* using BERT.

DistilBERT.

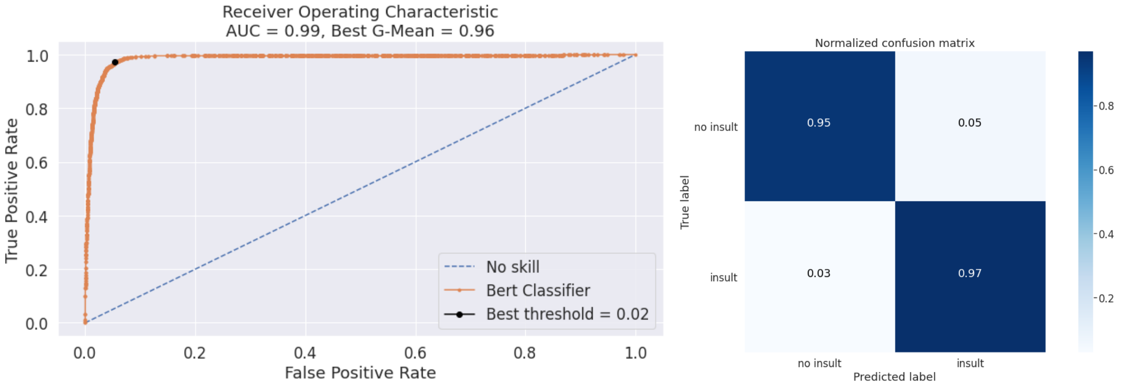


(a) ROC curve

(b) Confusion matrix

Fig. B.6. ROC curve and confusion matrix respect class *threat* using DistilBERT.

Insult

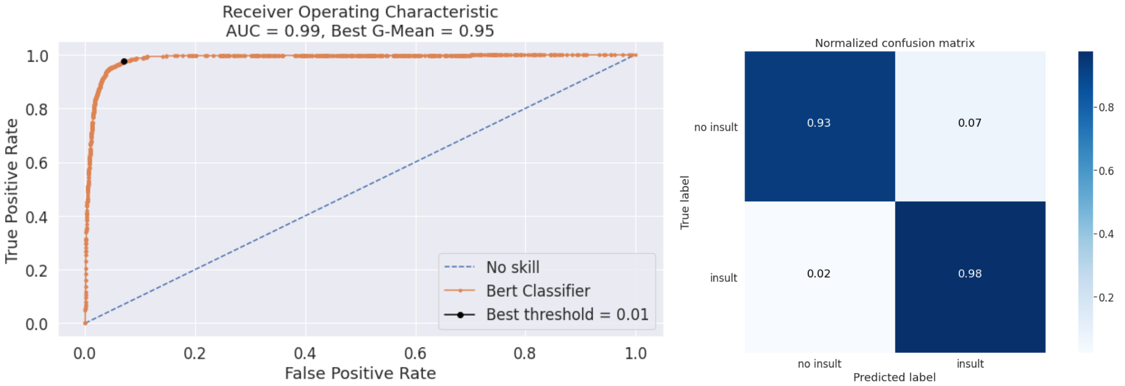


(a) ROC curve

(b) Confusion matrix

Fig. B.7. ROC curve and confusion matrix respect class *insult* using BERT.

DistilBERT.

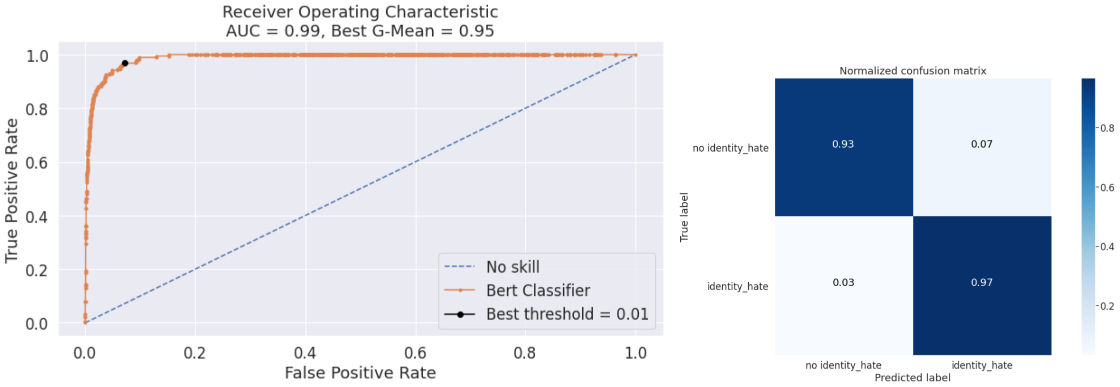


(a) ROC curve

(b) Confusion matrix

Fig. B.8. ROC curve and confusion matrix respect class *insult* using DistilBERT.

Identity hate

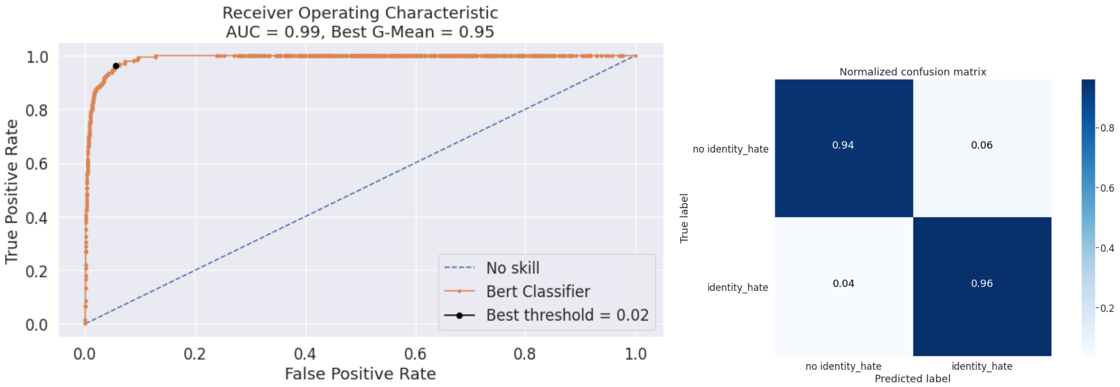


(a) ROC curve

(b) Confusion matrix

Fig. B.9. ROC curve and confusion matrix respect class *hate* using BERT.

DistilBERT.



(a) ROC curve

(b) Confusion matrix

Fig. B.10. ROC curve and confusion matrix respect class *hate* using DistilBERT.