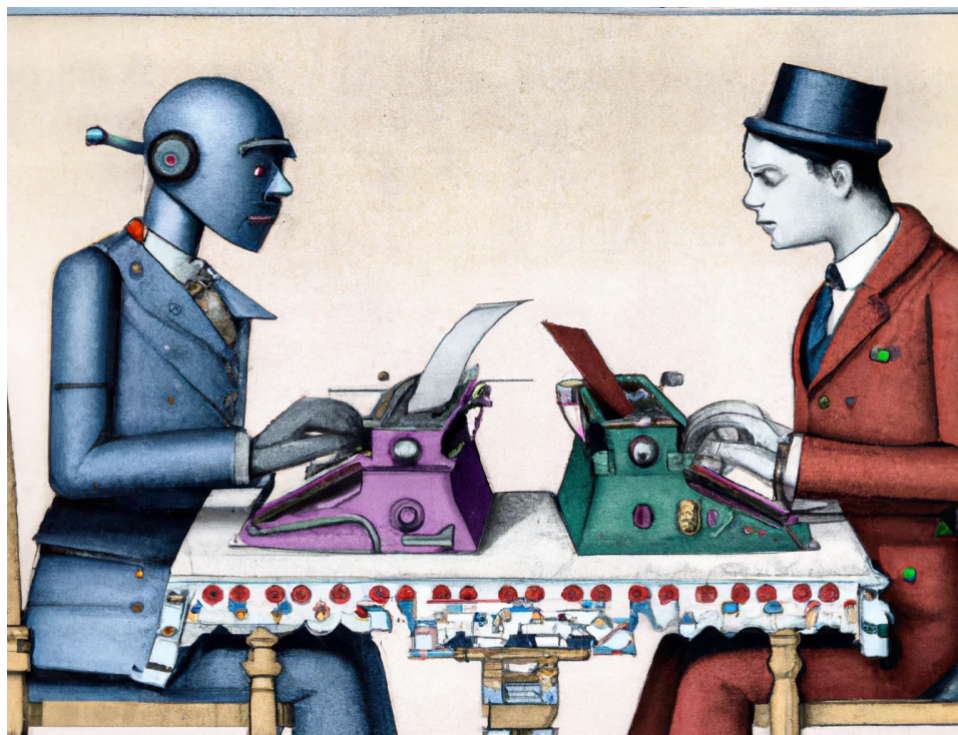




UNIVERSITÀ
DELLA CALABRIA

DIPARTIMENTO DI **MATEMATICA
E INFORMATICA**



Deep Learning Project: Text Classification

Professors:

Prof. Gianlugi Greco

Dott. Carletto Adorcazzo

Students:

Cristian Ciriaco Campagna

Giovanni Iannuzzi

Pierpaolo Sestito

Academic Year 2023/2024

Contents

1	Introduction	2
1.1	Project Overview	2
1.1.1	Objectives	3
2	Detect AI Generated Text	7
2.1	Dataset Description	7
2.2	Methods	9
2.2.1	Preprocessing	12
2.2.2	Architectures	13
3	Modelling	15
3.1	Model description	15
3.2	Model representation	18
4	Results	19
4.1	Final Results	19
4.2	Proof of Challenge Participation	19

Chapter 1

Introduction

In the era of rapidly advancing natural language processing (NLP) technologies, the ability to distinguish between human-generated and language model-generated text has become a pivotal challenge. This competition, titled "LLM Detect AI Generated Text," provides a unique opportunity to explore and address this task.

1.1 Project Overview

The competition dataset encompasses two primary components: `trainessays.csv` and `trainprompts.csv`. The former contains essays with identifiers (`id`), associated prompt identifiers (`promptid`), and the actual essay text (`text`). A crucial attribute, `generated`, indicates whether the essay was composed by a human student (marked as 0) or generated by a Language Model (LM, marked as 1). Notably, the objective is to develop a robust classification model capable of discerning the origin of each essay.

The latter dataset, `trainprompts.csv`, provides essential contextual information for the essays. Each prompt is uniquely identified by `promptid` and features metadata such as `promptname`, `instructions`, and the `sourcetext` (`sourcetext`).

The latter is presented in Markdown format, featuring enumerated paragraphs,

titles, and optional author information. The essays were crafted in response to these prompts, creating a rich training ground for the classification task.

1.1.1 Objectives

The overarching goal of this project is to construct a machine learning model that excels in classifying text as either human-authored or generated by a Language Model. By leveraging the information embedded in the essays and their corresponding prompts, participants are challenged to develop models that generalize well to unseen data and exhibit a nuanced understanding of the distinctive features between human and AI-generated language.

This report delineates the methodology, challenges encountered, and the results achieved during the exploration of this captivating competition. Through a comprehensive analysis, we aim to contribute insights into the evolving landscape of text classification and the intricate interplay between human and artificial intelligence in language generation.

The activities in question were organized as follows:

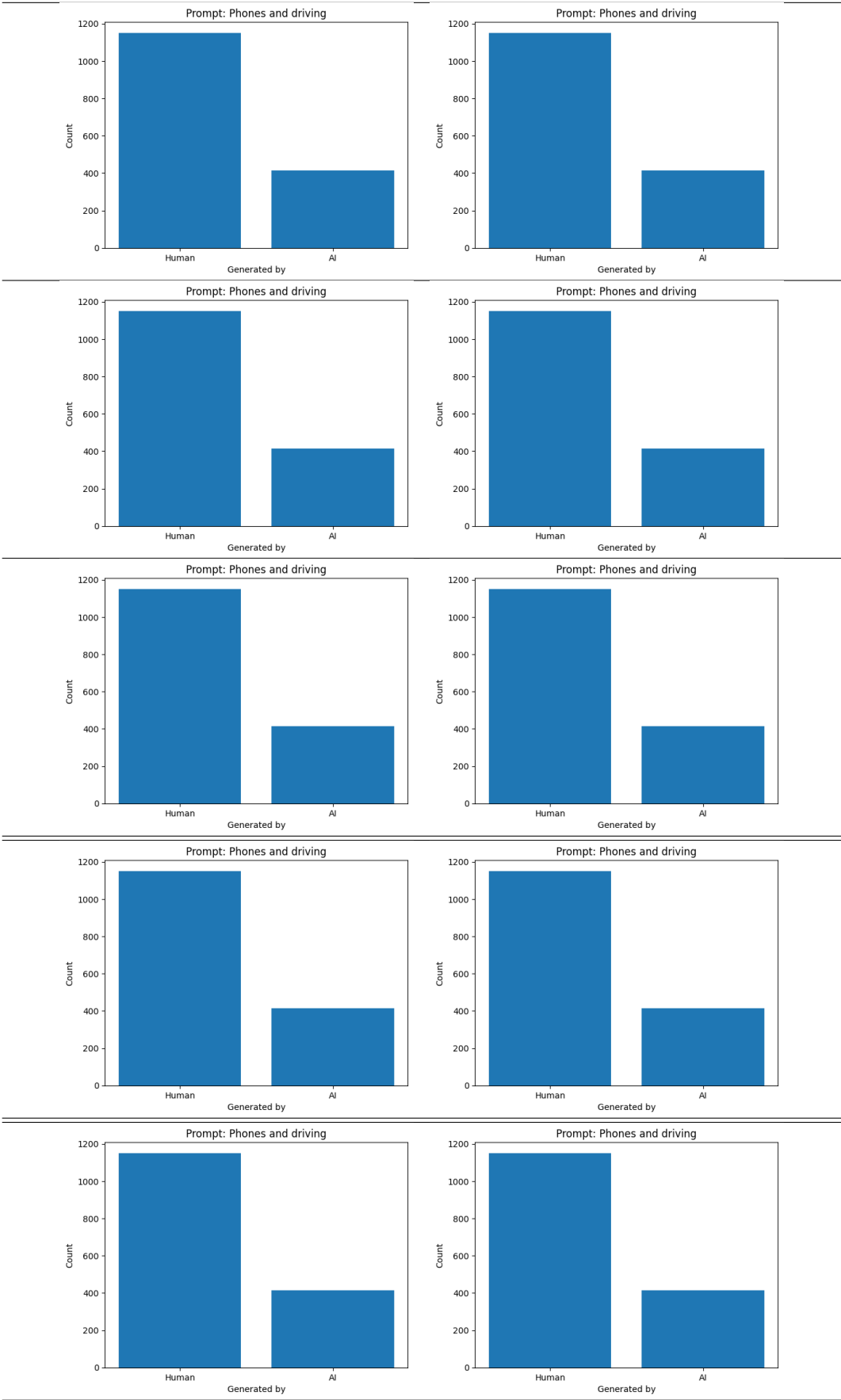


Table 1.1: Una tabella 5x3 con immagini

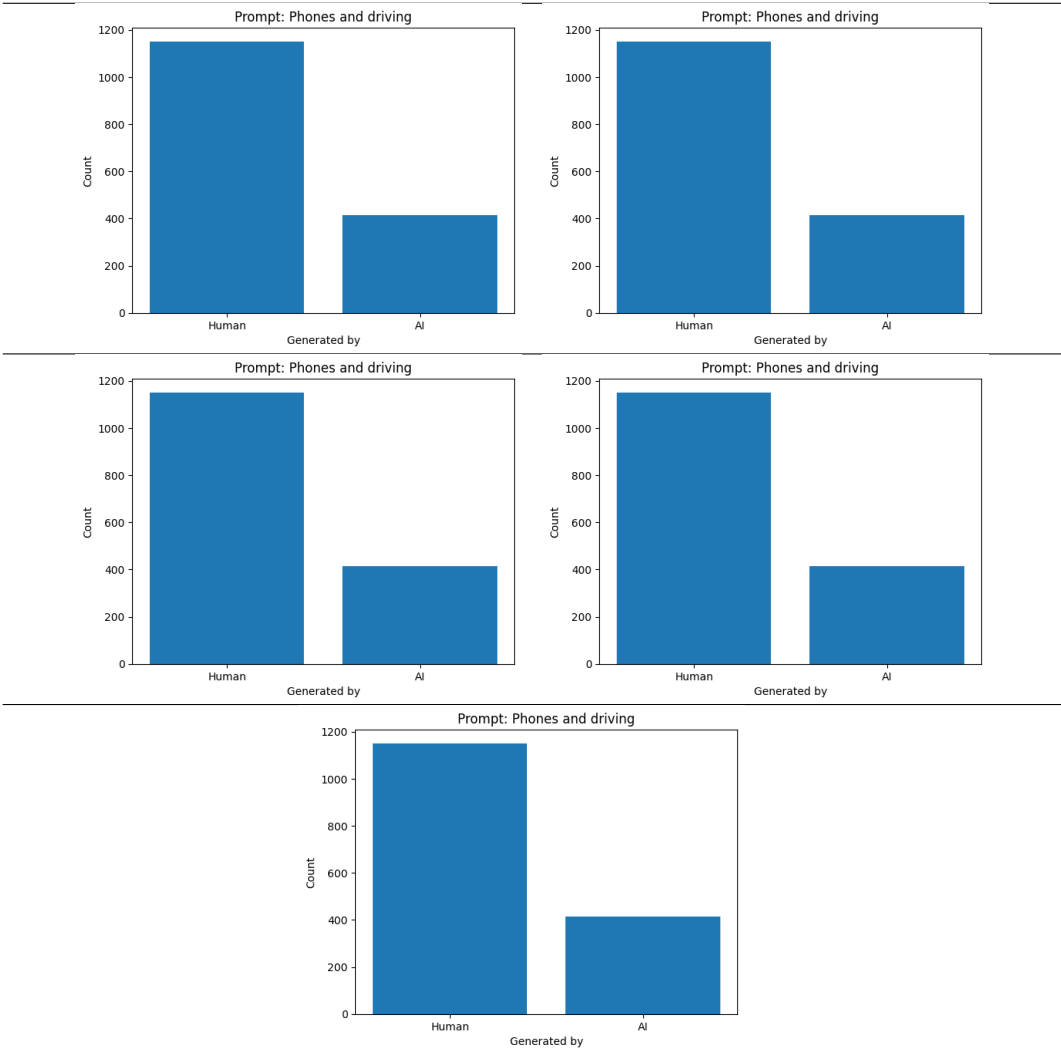


Table 1.2: Una tabella 5x3 con immagini

Chapter 2

Detect AI Generated Text

In this phase we describe the dataset and its attributes and we analyze each individual attribute checking its validity and semantics.

2.1 Dataset Description

During the initial exploration of the competition dataset (`train_essays.csv`), we observed a significant class imbalance. With only three records labeled as 1 (indicating AI-generated text) and 1375 labeled as 0 (indicating human-generated text), the dataset exhibited skewed representation.

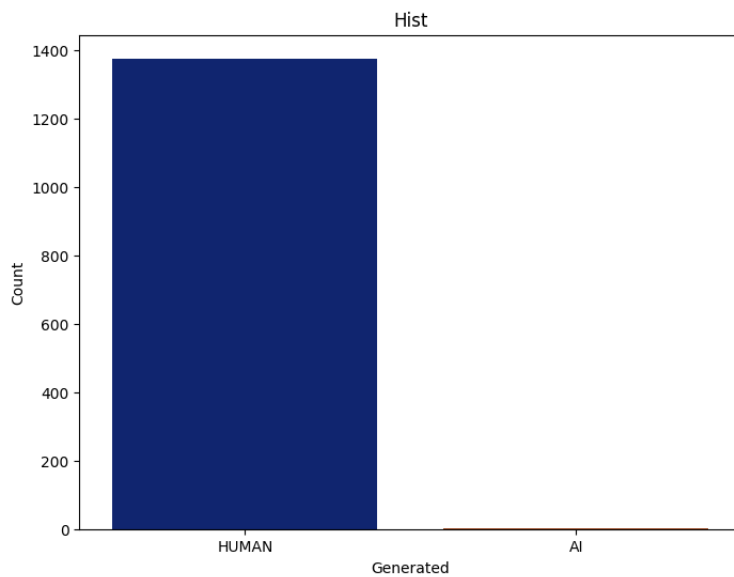


Figure 2.1: train_essays balance

Recognizing the importance of a balanced dataset for model training and evaluation, we embarked on a search for a more comprehensive and balanced alternative.

As a result of our efforts, we identified and start to analyze the "DAIGT-v2" dataset. This new dataset addresses the imbalance concerns present in the initial dataset, providing a more equitable distribution between AI-generated and human-generated texts, but not at all.

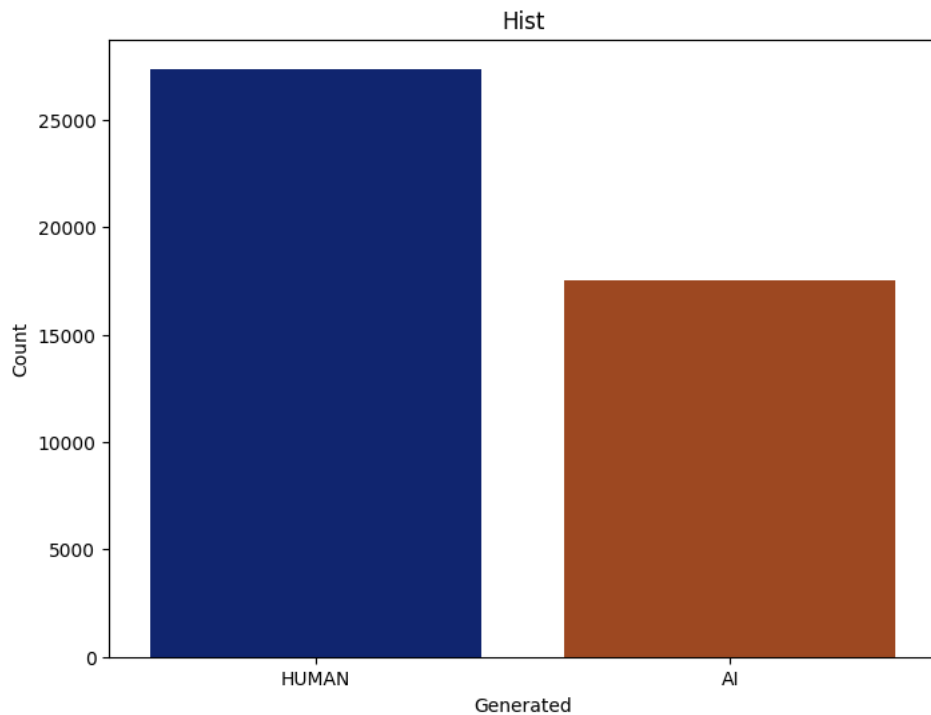


Figure 2.2: DAIGTv2

In the exploration of the DAIGTv2 dataset, our focus shifted towards the textual contributions of human authors. To foster a more balanced representation within the dataset, we employed the "*cosine similarity*" metric as a filtering mechanism. This approach aimed to mitigate redundancy and enhance diversity by removing texts with a cosine similarity greater than or equal to 0.9.

2.2 Methods

Cosine Similarity Filtering

The cosine similarity was calculated for pairs of textual entries attributed to human authors. Entries exhibiting a cosine similarity of 0.9 or higher were identified as closely related and were subsequently pruned from the dataset. This method allowed us to retain a diverse set of human-generated texts while reducing redundancy within the dataset.

Impact on Dataset balance

The application of the cosine similarity filtering mechanism aligns with our broader objective of achieving a balanced distribution of labels within the dataset. By selectively removing highly similar texts, we aimed to diminish potential biases introduced by overrepresented or redundant human-authored samples.

Record Generation for decreasing the unbalance

In response to the persistent label imbalance observed even after cosine similarity filtering, a strategic decision was made to introduce new records through the utilization of language models.

Prompt Selection Criteria

The generation process commenced by narrowing the focus to prompts with

	prompt_name	human	ai	difference
0	"A Cowboy Who Rode the Waves"	1372	524	848
2	Cell phones at school	1656	463	1193
3	Community service	1542	550	992
6	Driverless cars	1886	364	1522
7	Exploring Venus	1862	314	1548
8	Facial action coding system	2167	917	1250
9	Grades for extracurricular activities	1626	490	1136
11	Phones and driving	1151	415	736
13	Summer projects	1750	951	799
14	The Face on Mars	1583	310	1273

Figure 2.3: sources used in DAIGTv2

a substantial difference in the count of records labeled as 0 and 1, exceeding a predefined threshold of 1000. This criterion aimed to target prompts that inherently displayed an imbalance, thus necessitating the introduction of additional instances.

Model for Text Generation Selection and Analysis

Subsequently, an exhaustive analysis of the 'source' attribute within the DAIGTv2 dataset was undertaken. The 'source' attribute denotes the models responsible for generating each record.

source	
persuade_corpus	25996
mistral7binstruct_v1	2421
mistral7binstruct_v2	2421
chat_gpt_moth	2421
llama2_chat	2421
kingki19_palm	1384
train_essays	1378
llama_70b_v1	1172
falcon_180b_v1	1055
darragh_claude_v6	1000
darragh_claude_v7	1000
radek_500	500
NousResearch/Llama-2-7b-chat-hf	400
mistralai/Mistral-7B-Instruct-v0.1	400
cohere-command	350
palm-text-bison1	349
radekgpt4	200

Figure 2.4: sources used in DAIGTv2

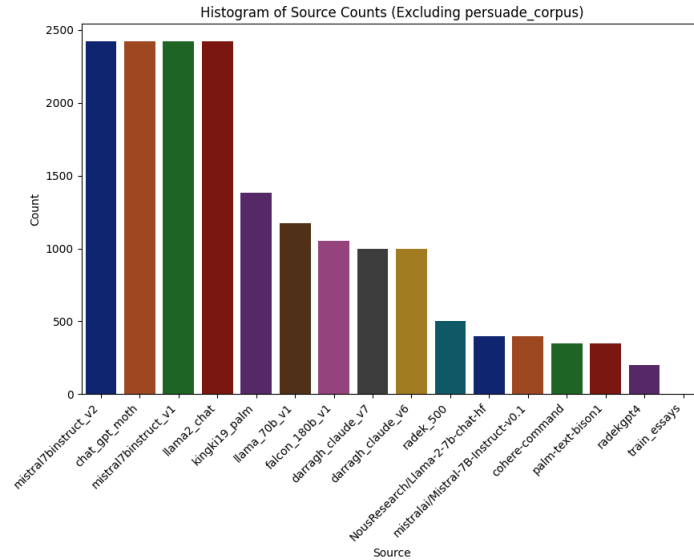


Figure 2.5: sources used in DAIGTv2

In light of resource constraints and a desire for computational efficiency, a meticulous examination led to the selection of *Cohere* and *Mistral* as models for record generation. Both models, chosen for their computational efficiency and ease of use, were deemed suitable alternatives given the limitations in computational resources. Also because they're used less than others.

New dataset : DAIGTv3

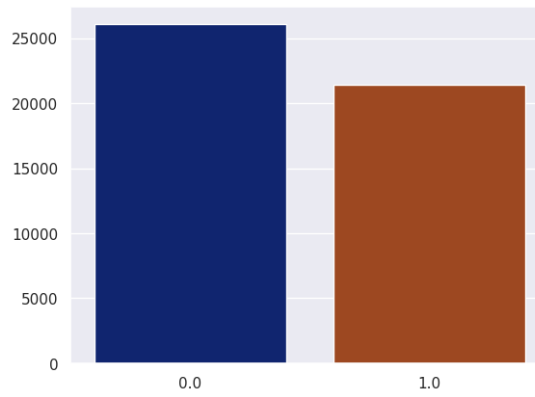


Figure 2.6: DAIGTv2

2.2.1 Preprocessing

In the initial phase of the data preparation process, our focus was on optimizing the data.

Stop Words Removal and Text Normalization

The first operation performed was the removal of "stop words," commonly occurring words that contribute little semantic value to the text. To implement this operation, three different functions were considered: the first based on the Natural Language Toolkit (NLTK), the second utilizing the SpaCy library, and the third leveraging Gensim. Through a series of experiments, it was found that the SpaCy-based approach yielded the most satisfactory results in terms

of preserving the text’s meaning while simultaneously reducing noise caused by low semantic value words. Finally, to standardize the text and reduce the dataset’s dimensionality, all sentences were converted to lowercase. This operation prevents the duplication of features due to different representations of words with uppercase and lowercase letters, simplifying the model’s learning phase.

Other attempts In addition to the previously mentioned methods, other preprocessing techniques were employed. However, during the training phase, they yielded inferior results, leading to their exclusion. Below is a list:

- Strip punctuation
- Lemmatization (using NLTK)

2.2.2 Architectures

In our pursuit of achieving the goal of distinguishing between human and AI-generated text, we embarked on a series of attempts, each aimed at finding the most suitable architecture for our specific dataset characteristics.

- **BERT-model based**

Our initial approach involved leveraging a pre-trained BERT (Bidirectional Encoder Representations from Transformers) model. BERT is renowned for its contextual understanding of language, making it a promising candidate. However, our dataset’s substantial size surpassed the 512-token limit imposed by BERT. Recognizing the inefficiency of discarding such a significant amount of data, we concluded that utilizing a pre-trained BERT model would not be a viable solution for our specific case.

- **LongFormerModel variant**

To overcome the token limit constraint, we explored a variant of BERT known as the LongFormerModel, which provides the capability to handle sequences of up to 4096 tokens. Regrettably, due to computational resource saturation, we had to halt our efforts to use this pre-trained model. The demanding resource requirements rendered it impractical for our current infrastructure.

- **RNN final version**

Faced with the challenges posed by pre-trained models, we turned our attention to building a custom solution. Our third and final attempt involved the implementation of a Recurrent Neural Network (RNN). RNNs are well-suited for sequential data and exhibit the ability to capture dependencies over time. This approach offers more flexibility in handling datasets with varying lengths of text sequences.

Hyperparameter Tuning

Upon transitioning to the development of our final solution using a Recurrent Neural Network (RNN), we recognized the critical importance of fine-tuning the model's hyperparameters to optimize its performance. To address the various trial and error phases in parameter selection during model construction, such as the number of neurons in layers, dropout rates, we relied on the use of Keras Tuner. The goal of this technology is to fine-tune hyperparameters. It was not used for the final model selection but facilitated monitoring the accuracy and loss trends, ultimately providing a baseline model for more meaningful experiments. For our specific use case, we chose to implement the Random Search algorithm, one of the search strategies available within Keras Tuner.

Chapter 3

Modelling

3.1 Model description

The proposed model is designed for the task of natural language processing, specifically sentiment analysis, within the framework of a binary classification problem. The architecture incorporates various layers to effectively capture and represent intricate patterns within sequential textual data.

- **Tokenization Layer**

The initial layer of the model utilizes a text tokenization layer, represented as encoder. This layer preprocesses input text data, converting words into numerical tokens. Although the specific configuration details are not provided, the tokenization step is crucial for translating the textual information into a format suitable for subsequent neural network layers.

```
tf.keras.layers.TextVectorization(  
    output_sequence_length=500, # diverso da cristian  
    standardize=None,
```



```
max_tokens=8000),
```

– **Embedding Layer**

Following the tokenization layer, an embedding layer is employed to transform the discrete tokenized representation into continuous vector space. The Embedding layer has an `input_dim` equal to the vocabulary size obtained from the tokenization layer. Each token is embedded into a vector of size `output_dim`, set to 3000 in this architecture. The `mask_zero=True` parameter is utilized to handle sequences of varying lengths, effectively masking padded values.

```
tf.keras.layers.Embedding(  
    input_dim=len(encoder.get_vocabulary()),  
    output_dim=3000,  
    mask_zero=True),
```

– **Bidirectional LSTM Layers**

Two bidirectional Long Short-Term Memory (LSTM) layers are employed for sequence modeling.

The first LSTM layer, configured with 16 units and `return_sequences=True`, processes input sequences bidirectionally while maintaining the temporal sequence information.

```
tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(  
    units=16,  
    return_sequences=True))
```

Subsequently, a dropout layer with a rate of 0.5 is introduced to mitigate overfitting.

```
tf.keras.layers.Dropout(rate=0.5)
```

The second bidirectional LSTM layer, mirroring the configuration of the first, further captures contextual dependencies within the se-

quence. The absence of `return_sequences` in this layer indicates that it provides a consolidated representation rather than a sequence.

```
tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(  
    units=16))
```

– Dense Layers

A densely connected layer with 8 units and Rectified Linear Unit (ReLU) activation is incorporated for feature extraction and dimensionality reduction. Additionally, a kernel regularization term with L1 penalty (0.1) is applied to the weights of this layer.

```
tf.keras.layers.Dense(  
    units=8,  
    activation = 'relu',  
    kernel_regularizer = regularizers.l1(0.1))
```

The final layer is a dense layer with a single unit and a sigmoid activation function, producing the binary classification output. The sigmoid activation is chosen for its suitability in binary classification tasks.

```
tf.keras.layers.Dense(1, activation='sigmoid')
```

This comprehensive architecture is tailored to address the complexities of sentiment analysis in natural language text, encompassing tokenization, embedding, bidirectional sequence modeling, and dense layers for feature extraction and prediction.

3.2 Model representation

At finally we have:

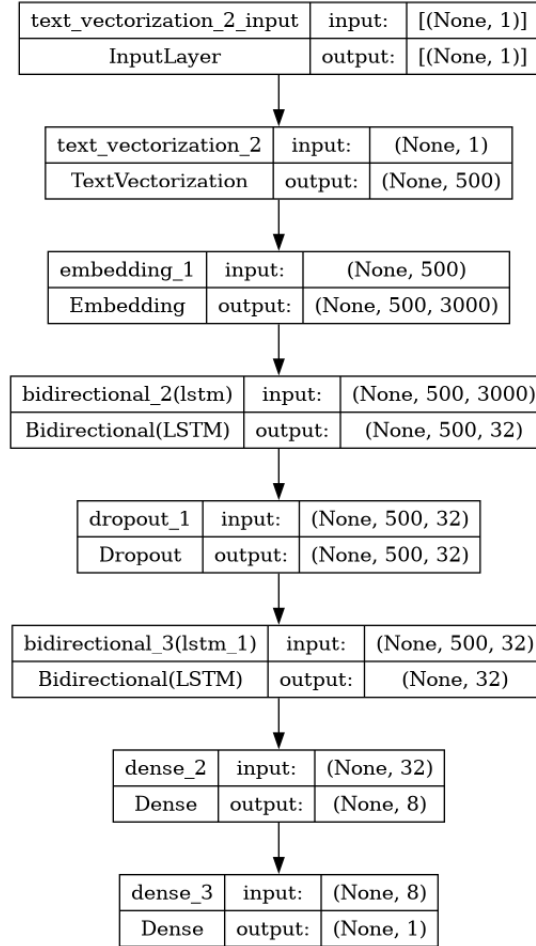


Figure 3.1: Recurrent Neural Network

Chapter 4

Results

4.1 Final Results

	Competition Notebook LLM - Detect AI Generated Text	Run 529.2s - GPU T4 ×2	Public Score 0.855	Best Score 0.855 V2
---	---	----------------------------------	------------------------------	---

Figure 4.1: Accuracy on LLM Detect AI Generated Text Kaggle

4.2 Proof of Challenge Partecipation


#	Team	Members	Score
2927	G15		0.855

Figure 4.2: Kaggle Competition - Leaderboard

- Link to our notebook
- GitHub Repository
- Il nostro inno



Figure 4.3: To Adornetto and Greco