University of Milan Bicocca

School of Science

Department of Computer Science, Systems and Communication

Bachelor of Science in Computer Science

DATA ANALYTICS

Exam Project

Luca Poli 852027 l.poli6@campus.unimib.it

Summary

Introduction

For this Data Analytics project, I will address the problem expressed in Track 3, namely sarcasm detection. This problem is one of the most interesting ones in the field of NLP (Natural Language Processing), due to its complexity; caused by its subjective nature and the linguistic nuances involved; in fact, it is often not so simple even for a human being. In particular, the system will have to examine a text, and helping itself with external elements (the context), it will have to figure out whether it is sarcastic or not.

Several strategies were adopted in the research for this task, which can be divided into two categories:

- Based on linguistic features: in which texts are analyzed for ligustic patterns related to sarcasm; such as the use of multiple contradictory words in the same sentence, or with respect to context, the use of contrasts between words with positive and negative connotation, or the use of exaggeration.
- Based on Machine Learning or Deep Learning models: in which supervised (or semi-supervised) learning techniques are exploited to train statistical models with labeled data to detect sarcastic texts; generally the models can be more "simple" such as SVM, Random Forest Naive Bayes, or small neural networks, up to Deep Learning models, such as RNN (Recurrent Neural Network) or the Transfomers that can reach and exceed millions of parameters.

In this project, the problem-solving strategy adopted will be based on supervised training with Transformers; as they are a model that is revolutionizing the field, and it will be interesting to apply them in the foreground (albeit on a small scale) to this difficult task.

The report will be divided into three main chapters: in the first I will describe the data analysis carried out, from which the choices made are based; in the second I will discuss the model used and its training; and in the third I will discuss the results produced.

Chapter 1: Data Analysis

In this chapter, I will describe the analysis of the training data, examining the elements of the dataset, the classification target, the texts, and the context elements. With which I can draw conclusions in order to process the data and train the model more carefully.

1.1 General description of the data

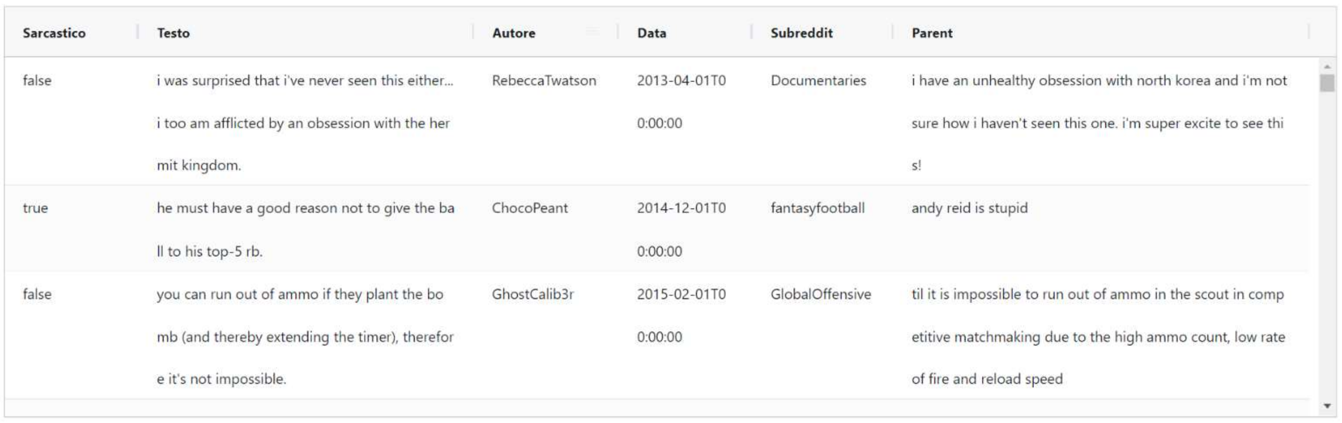
1.1.1 Introduction to datasets and Splitting

Associated with the trace, there are two datasets, one with training data (train) and one for testing. They are two sets of Reddit comments from 2009 to 2016; specifically, for each instance we have:

- Text: Text of the comment to be classified;
- Sarcastic: Boolean label indicating whether the comment is sarcastic (True) or not (False);
- Author: Name of the author of the comment;
- Subreddit: Name of the subreddit in which the comment was posted;
- Date: Date the comment was published;
- Parent: Text of the discussion to which the comment refers;

The original training dataset, with one million rows (precisely 1010822), will be randomly divided into two parts: the final training dataset (with 95%), on which all analyses and assumptions will be made; the validation dataset (with the remaining 5%), used during the training process to measure model performance.

The training dataset is very complete, because the number of duplicate and invalid rows is relatively insignificant (200 and 53, respectively); therefore, no special strategies need to be implemented to recover missing or null data (such as imputation), and removal of them can be done. In addition, both text and date encoding comply with standards, and thus no conversion errors are present.



Sarcastico	Testo	Autore	Data	Subreddit	Parent
false	i was surprised that i've never seen this either... i too am afflicted by an obsession with the her mit kingdom.	RebeccaTwatson	2013-04-01T0 0:00:00	Documentaries	i have an unhealthy obsession with north korea and i'm not sure how i haven't seen this one. i'm super excite to see thi s!
true	he must have a good reason not to give the ba ll to his top-5 rb.	ChocoPeant	2014-12-01T0 0:00:00	fantasyfootball	andy reid is stupid
false	you can run out of ammo if they plant the bo mb (and thereby extending the timer), therefor e it's not impossible.	GhostCalib3r	2015-02-01T0 0:00:00	GlobalOffensive	til it is impossible to run out of ammo in the scout in comp etitive matchmaking due to the high ammo count, low rate of fire and reload speed

Figure 1.1: Some sample training lines

1.1.2 Description of analysis methodology

Each row consists of a label and its instance, whose attributes can be divided into the fundamental text to be classified (the Text attribute), and a set of contextual attributes that add information for the text (the remaining attributes).

Given this division, I structured the analysis process in two stages:

- In the first step I will examine context attributes, calculating for each unique element of each attribute what is the rate at which it appears in sarcastic and non-sarcastic texts (called the Informative Rate), so as to identify the elements (and thus their attributes) most relevant to a model.
- In the second phase I will examine the different ways to process text, calculating which system produces the text with a higher information rate content.

In detail, the information rate of a single element ($e \in Feature$) will be calculated as:

$$IR_e = \left| \frac{n_{sc}}{n_{sc} + n_{nsc}} - S_{IR} \right| * 100$$

Where: n_{sc} e n_{nsc} are the number of sarcastic and nonsarcastic comments associated with e, respectively; while, S_{IR} is the probability of the most likely class.

1.1.3 Target analysis

The target, i.e., the Sarcastic attribute, is the Label that identifies the nature of the comment, and thus will be the target of the NLP model.

For the training to be successful, it is important that its distribution be balanced, that is, that the number of sarcastic and non-sarcastic instances be approximately similar. As argued in the article "A Large Self-Annotated Corpus for Sarcasm" (attached to the track), and after a verification phase, I can state that the dataset exhibits this characteristic and therefore there is no need to apply techniques to improve the balance.

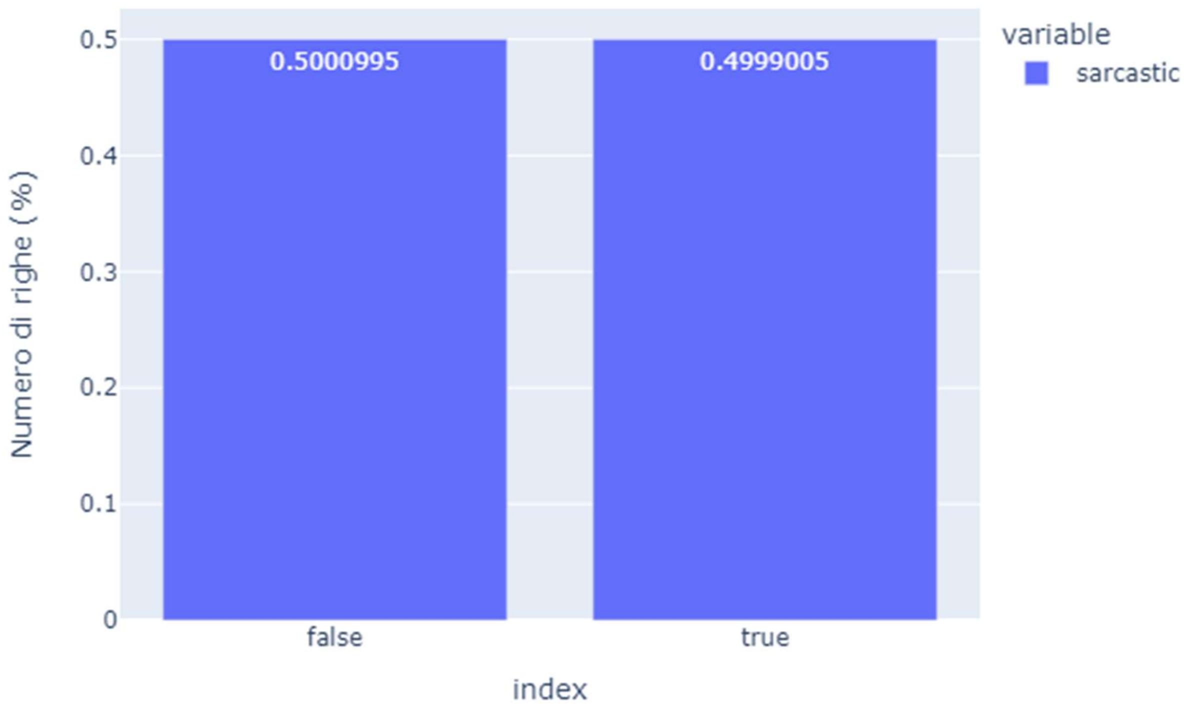


Figure 1.1: Target distribution obtained after the verification phase, as can be seen the balance is almost perfect

In addition, this analysis was used for me to calculate precisely. $S_{IR} \simeq 0.5001$.

1.2 Contextual feature analysis

As stated earlier such are defined as: Parent, Date, Subreddit and Author. They could be useful for prediction by adding context information to the comment. In detail:

- The Parent: coded as a multi-word string, it represents the text of the discussion to which the comment was left; it could be very useful for detecting conflicting meanings with the text (the main feature of irony), or for detecting polarized discussions sarcastic or not.
- Date: coded as a "year-month" string, may have relevance in finding highly sarcastic (or not) dates.
- Subreddit: coded as a one-word string, it represents the name of the subreddit in which the comment was posted; it could be useful for finding subreddits that tend to be more or less sarcastic.
- Author: coded as a one-word string, represents the name of the author of the comment; could be useful for finding more or less sarcastic authors.

From these premises, I performed an analysis to get real feedback on the data.

1.2.1 Analysis of the context versus the number of comments

At this stage I analyzed for each feature how many elements are repeated (taking them only once) and how many comments are associated with each.

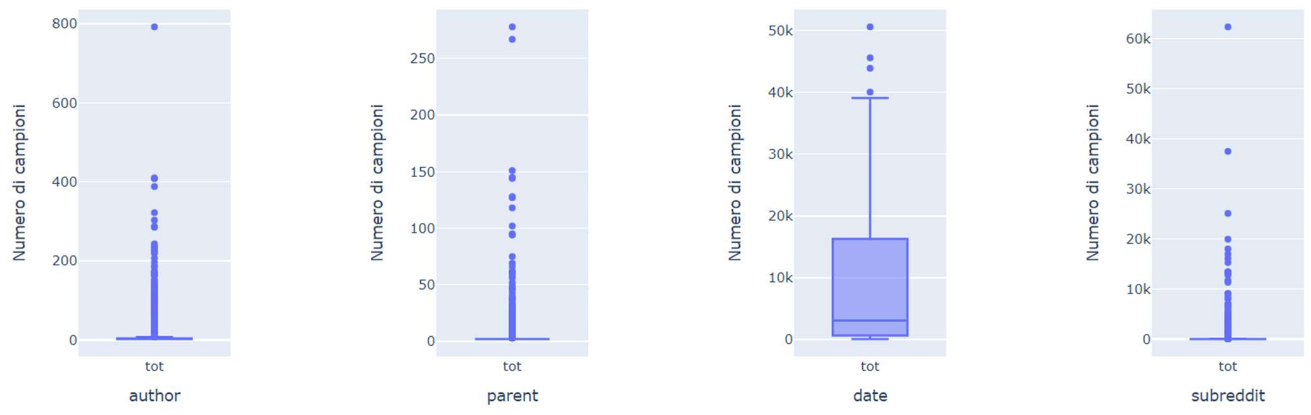
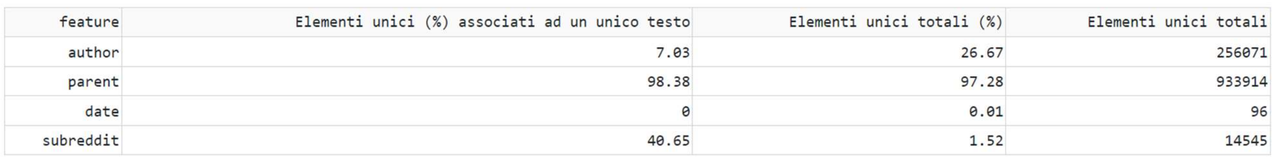


Figure 1.2: Distribution, via boxplot, of unique context elements versus the number of associated comments



feature	Elementi unici (%) associati ad un unico testo	Elementi unici totali (%)	Elementi unici totali
author	7.03	26.67	256071
parent	98.38	97.28	933914
date	0	0.01	96
subreddit	40.65	1.52	14545

Figure 1.3: Table showing the number of elements that are unique (and thus repeat in multiple comments), and the number of associated texts

From Figure 1.2 e Figure 1.3, it is particularly noticeable that:

- Almost every comment refers to a unique Parent; in fact, in the boxplot the distribution is "squeezed" toward 1 (the 1st, 2nd and 3rd quartiles are at 1), and in the table 97.3% of the Parents repeat at least once, of which 98.4% repeat exactly once.
- The dates, as expected being only 96, are repeated with a high frequency in the comments; in fact, in the boxplot the values are mainly distributed in a range from 668 to 16000, with median at 3108.
- The author feature could be relevant; as the number of unique authors compared to comments is 26.7% of which only 7% have only one comment, as shown in the table; the boxplot confirms the thesis, in fact, the distribution is placed in the range 1, 4
- The feature subreddit, with a distribution in the range 1, 9 and a number of unique items at 1.5% (14545 compared to 960000), is even more interesting; however, it is worth noting that 40.6% of them have only one comment.

1.2.2 Information rate analysis of the context individually

After analyzing how unique context features are distributed, I will proceed with the information content analysis of those features that have at least more than one associated comment. I will delve into each feature separately through histograms that present, for a range of associated comments, the average rate.

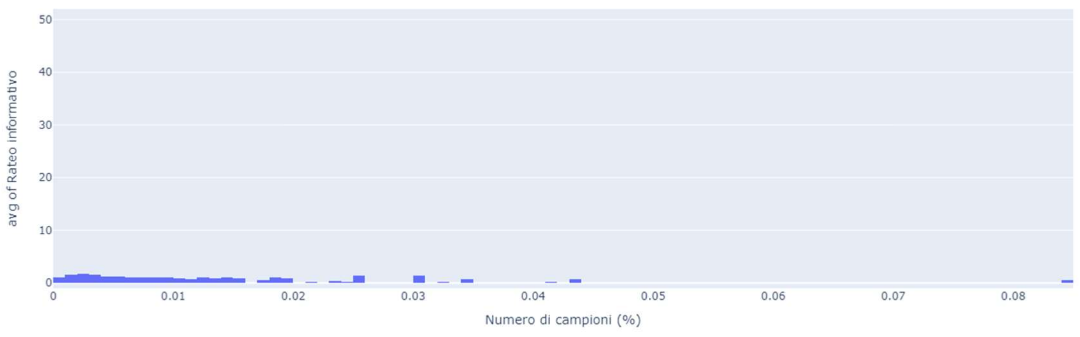


Figure 1.4: Information rate for Author elements.

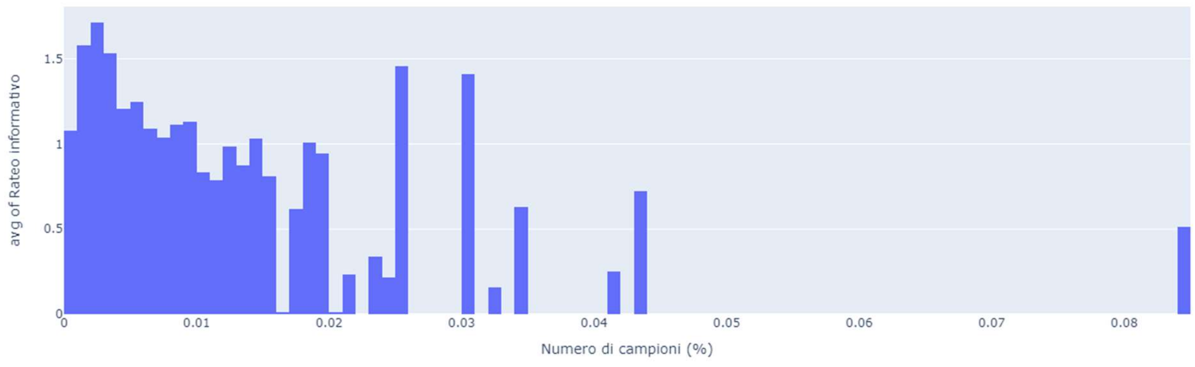


Figure 1.5: Information rate for Author elements. With y scaled

As can be seen in Figure 1.4 e Figure 1.5 the author information rate is very low; in fact, it is necessary to scale the y-axis between 0 and 1.8 in order to distinguish histograms. It follows that we cannot distinguish many sarcastic-majority authors or not, and thus this feature may be irrelevant.

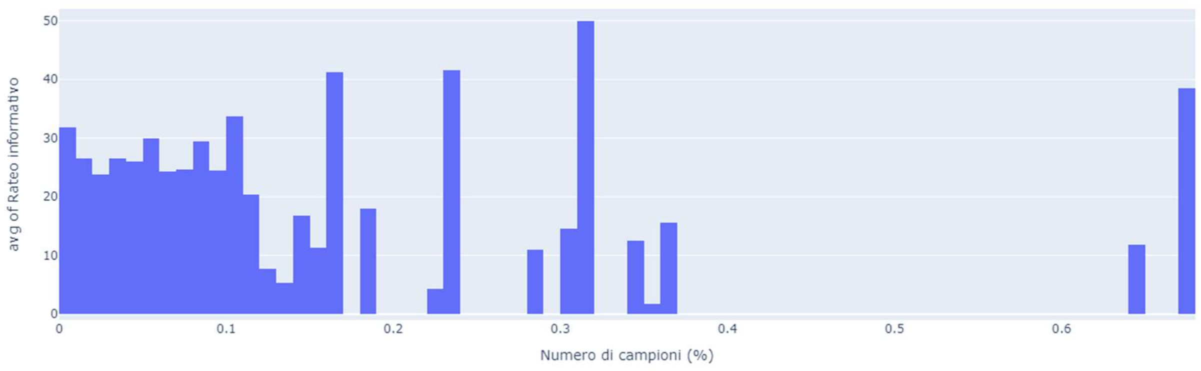


Figure 1.6: Information rate for Parent elements.

From Figure 1.6: We note that the information rate for Parent elements is high; however, as seen in the x-axis (and as confirmed by the Figure 1.3) the associated number of comments is low.

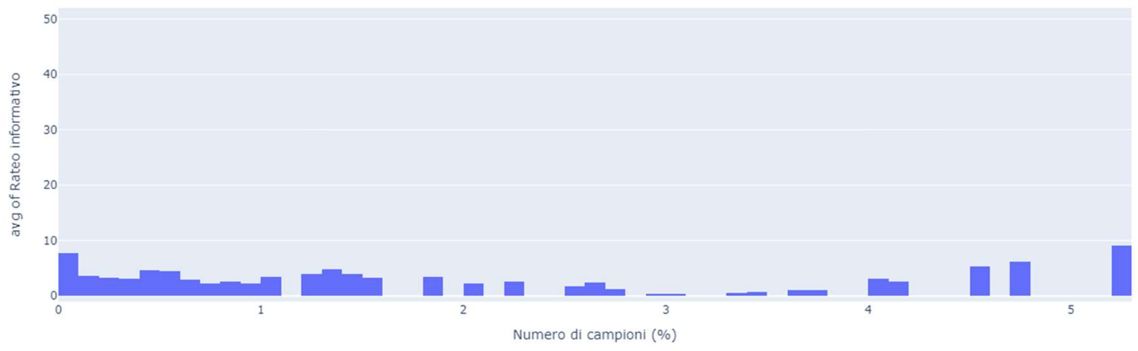


Figure 1.7: Information rate for date elements.

As can be seen from Figure 1.7 the information rate is quite poor, although the number of samples per element is quite high.

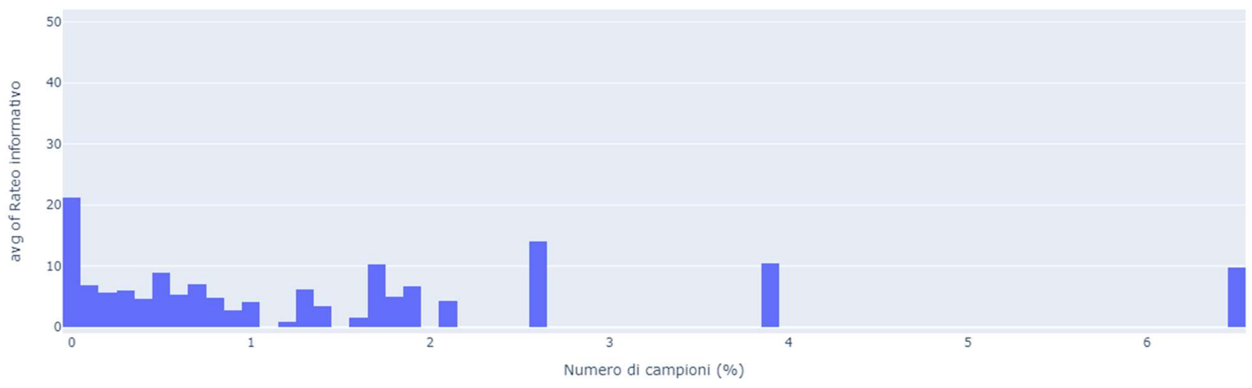


Figure 1.8: Information rate for subreddit elements.

From Figure 1.8 we can see that the information rate for Subreddit is significantly better than for dates, reaching and even exceeding 10% in many cases; as expected, however, we note that the rate drops as the number of comments increases.

1.2.2 Aggregate analysis of information accrual

After looking at the distributions individually of the information rate, it is useful to aggregate the results so that they can be compared efficiently. To do this, I followed two approaches:

- Threshold: in which I calculated the percentage of unique items exceeding thresholds (10%, 20%, 30%, 40%) of information rate.
- Mean: in which I calculated the mean and standard deviation weighted on the number of associated comments.

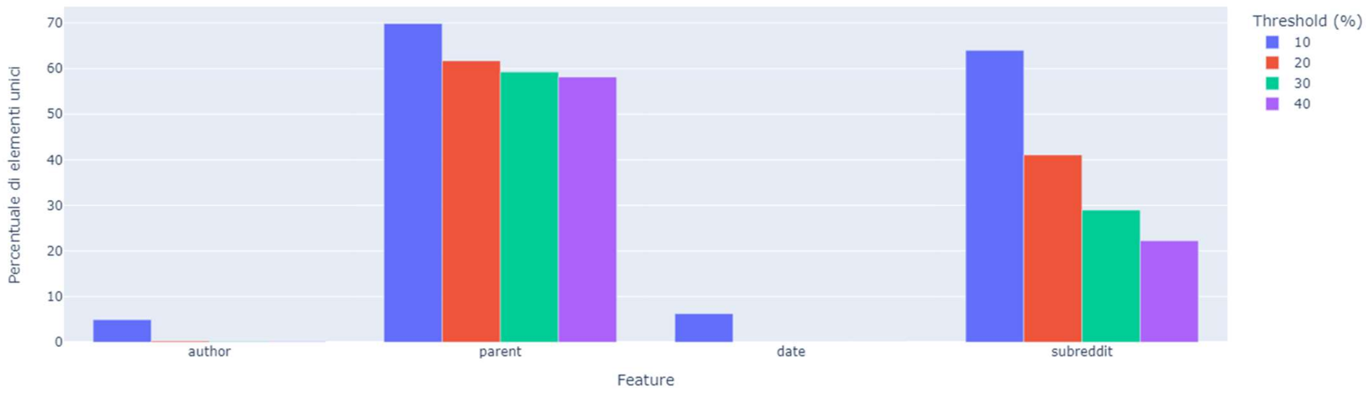
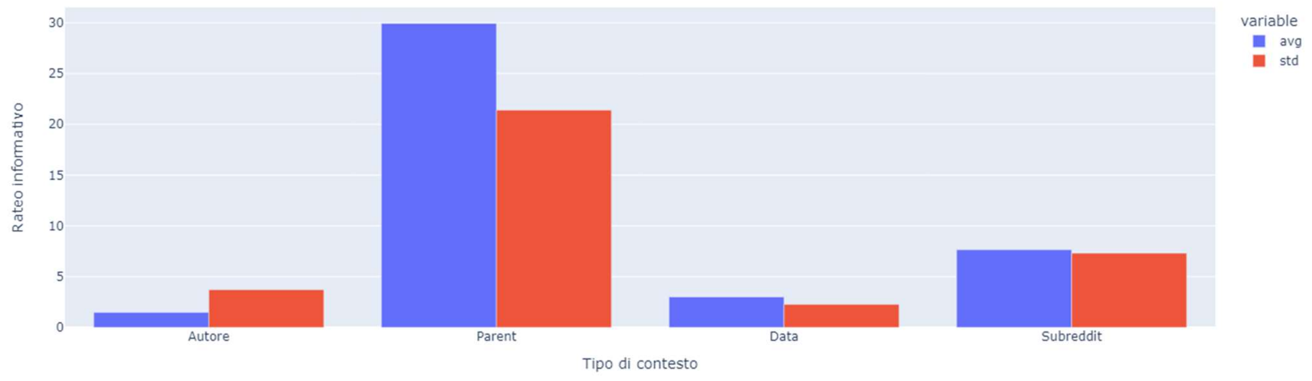


Figure 1.9: Threshold: Percentage of unique elements exceeding certain thresholds, per context feature

Figure 1.10: Weighted mean and STD of information rate for context types.



We can see from Figure 1.9 e **Error! Reference source not found.**:

- Date and Author have a low average information rate, and few exceed the 10% threshold.
- Parent has the highest average information rate, and almost all those that exceed the 10 percent threshold also exceed the 40 percent threshold; this, means that the few repeated items are very significant.
- Subreddit has an acceptable information rate, with as many as 64% of items exceeding 10% and 22% exceeding the 40% threshold.

1.2.3 Conclusions of context analysis

By combining the analyses done previously, we can come to useful conclusions for the model:

- Author presents not too many unique elements (thus associated with enough comments), however, they do not have enough information rate; therefore, they would introduce too much noise in model training.
- Given as an author, he has very few unique elements but these do not have a meaningful information rate; therefore, he would also introduce too much noise, and would be totally useless with comments in a later time period.
- Parent on the contrary has a high information rate, but the number of unique elements is almost equal to the number of comments; therefore, using the whole sentence statistically would not make any sense, however, associations of the words with each other or with the comment text could be useful.
- Subreddit has a good information rate and an adequate number of unique elements; therefore, it could be a useful feature for the model.

In conclusion, the Subreddit feature will be used by the model, Date and Author will not be used, and Parent will be used only in text binding.

1.3 Text analysis

For the analysis of the commentary text, I adopted a logic similar to that for context, but instead of comparing different features, I compared different ways of processing the text.

I did such analysis, with four graphs taking each type of text individually:

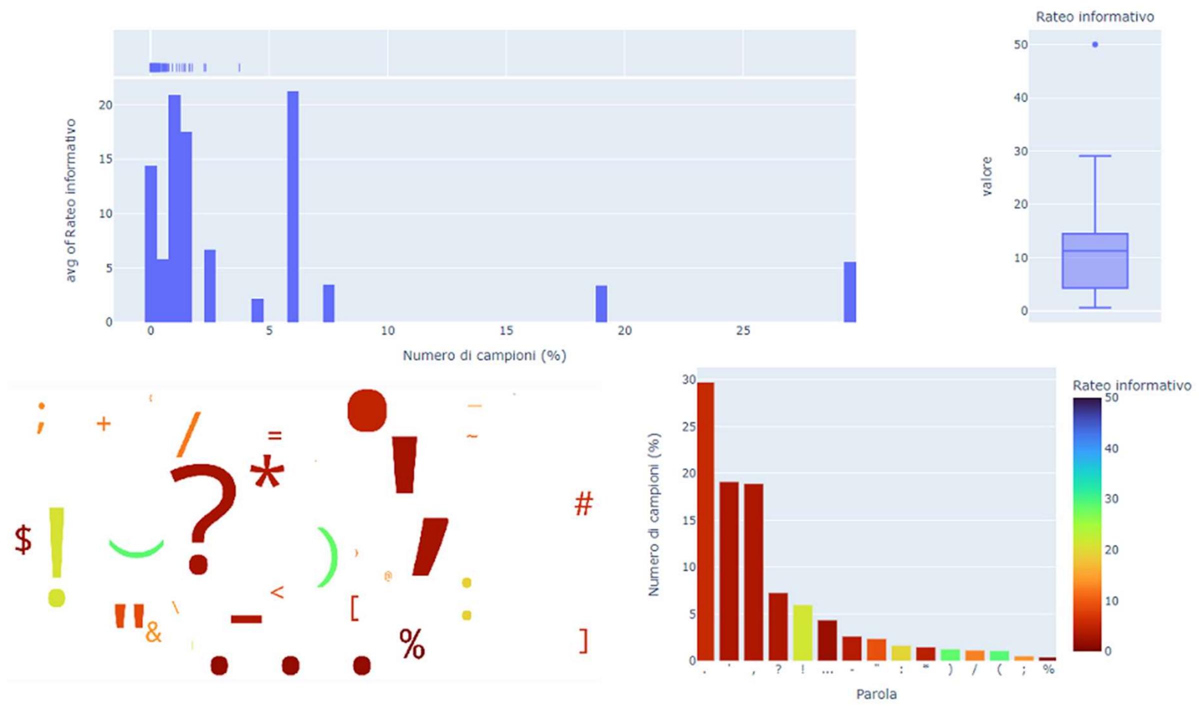
- Histogram: like the one used in context analysis, it shows the information rate associated with a range of token frequencies in comments.
- Boxplot: shows the information rate distribution in summary.
- Wordcloud: Graph showing the 200 most frequent tokens; size indicates frequency, while color indicates information rate.
- Barplot: shows in order the most frequent tokens and their information rate.

The various types of text, processed and analyzed, are:

- Normal
- Without stopwords: to which non-keywords have been removed; such as articles, conjunctions, etc....
- With stemming: in which derived and inflected words are reduced to their root form.
- Without stopwords and with stemming: in which the previous two approaches are applied simultaneously.

However, before the analysis, it is necessary to tokenize the text, that is, to convert it from a unique string into a list of words (tokens); and then remove the punctuation, as it is often not meaningful and only results in noise. However, in lieu of removing all punctuation I performed information rate analysis of the points to keep those with significant rate, so that there would not be excessive loss of information.

Figure 1.11: Rate distribution for punctuation symbols.



As can be seen from the **Error! Reference source not found.** the only significant punctuation mark is the "!"; because it has a rate of 21 (in fact sentences with at least one "!" are sarcastic 71% of the time), with a frequency of about 6%. So, I decided to keep this point and remove the others, and then proceed to the individual analysis of text types.

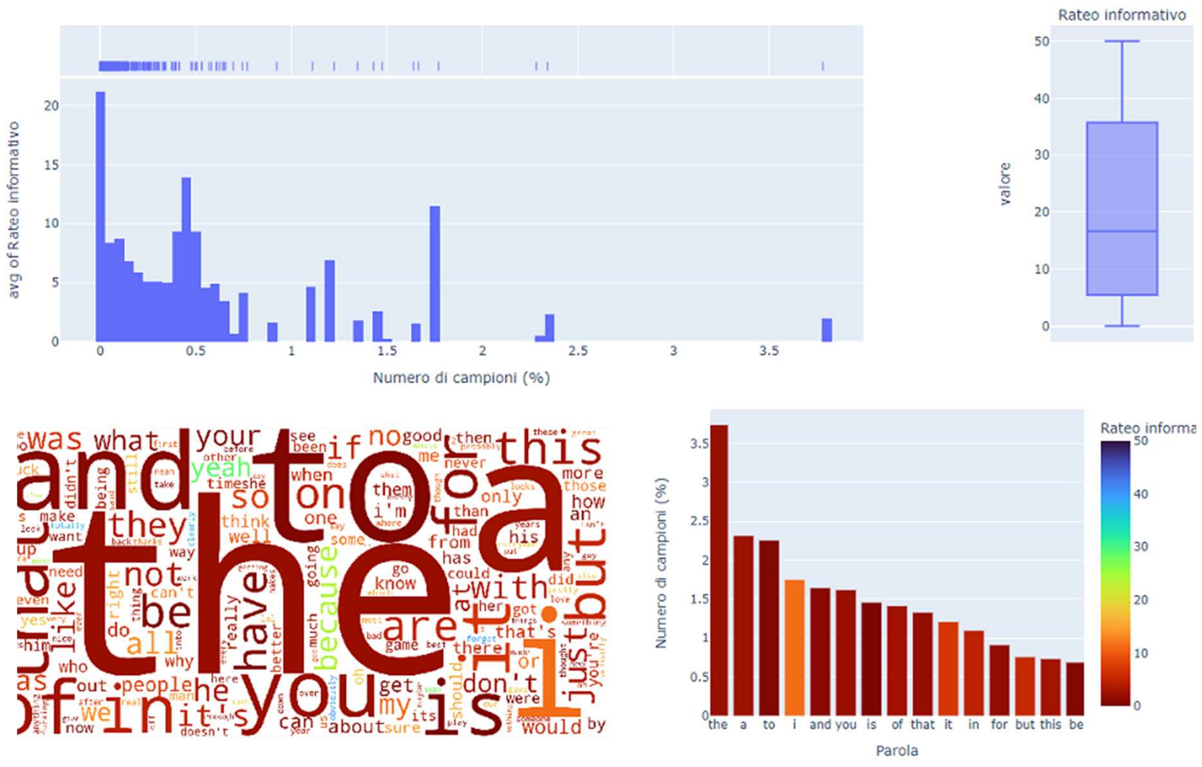


Figure 1.12: Rate distribution for plain text tokens.

As can be seen from Figure 1.12, most of the information accrual is distributed on the low-frequency elements, and therefore not very usable; in contrast, the more frequent elements, which are mainly stopwords do not carry much information, although they are distinguishable in the background of the wordcloud of useful tokens.

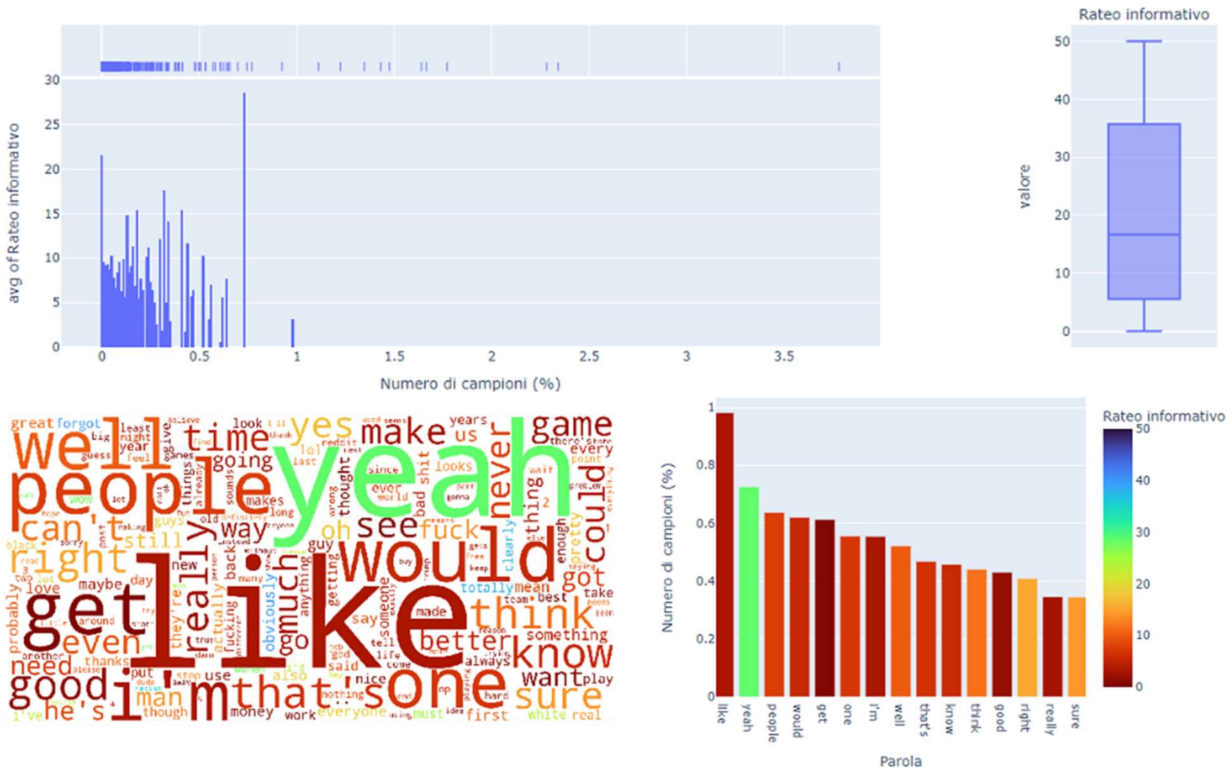
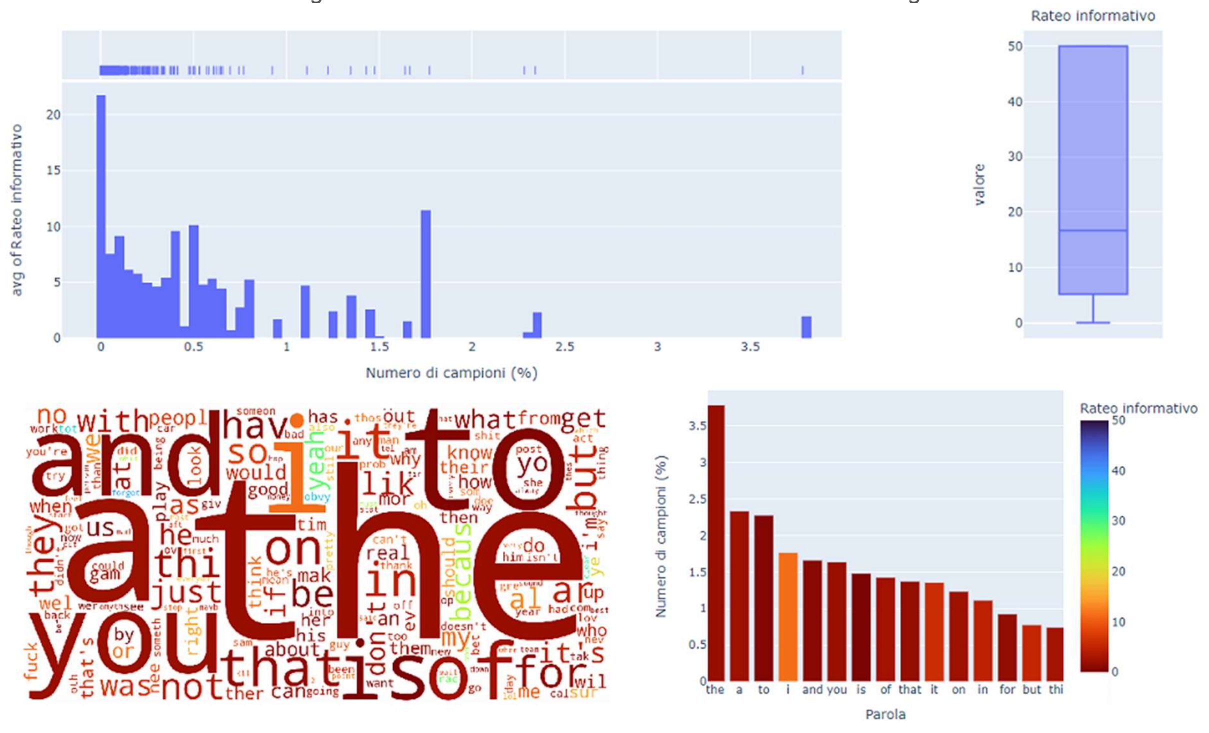


Figure 1.13: Rate distribution for text tokens without stopwords.

We observe that in Figure 1.13 the frequency of the most common tokens has dropped significantly, from more than 3.5 percent to almost 1 percent, but the rate of these tokens is significantly higher; in fact, we notice in both the wordcloud and the barplot that the colors are less prone to dark red (and thus have a higher rate).

Figure 1.14: Rate distribution for text tokens after stemming



Taking a closer look at the Figure 1.14 it can be seen that stemming has not had a good impact, and in fact the information rate in the most frequent elements has dropped significantly; instead, the boxplot shows that the range has widened, this due to the fact that the total number of elements has decreased.

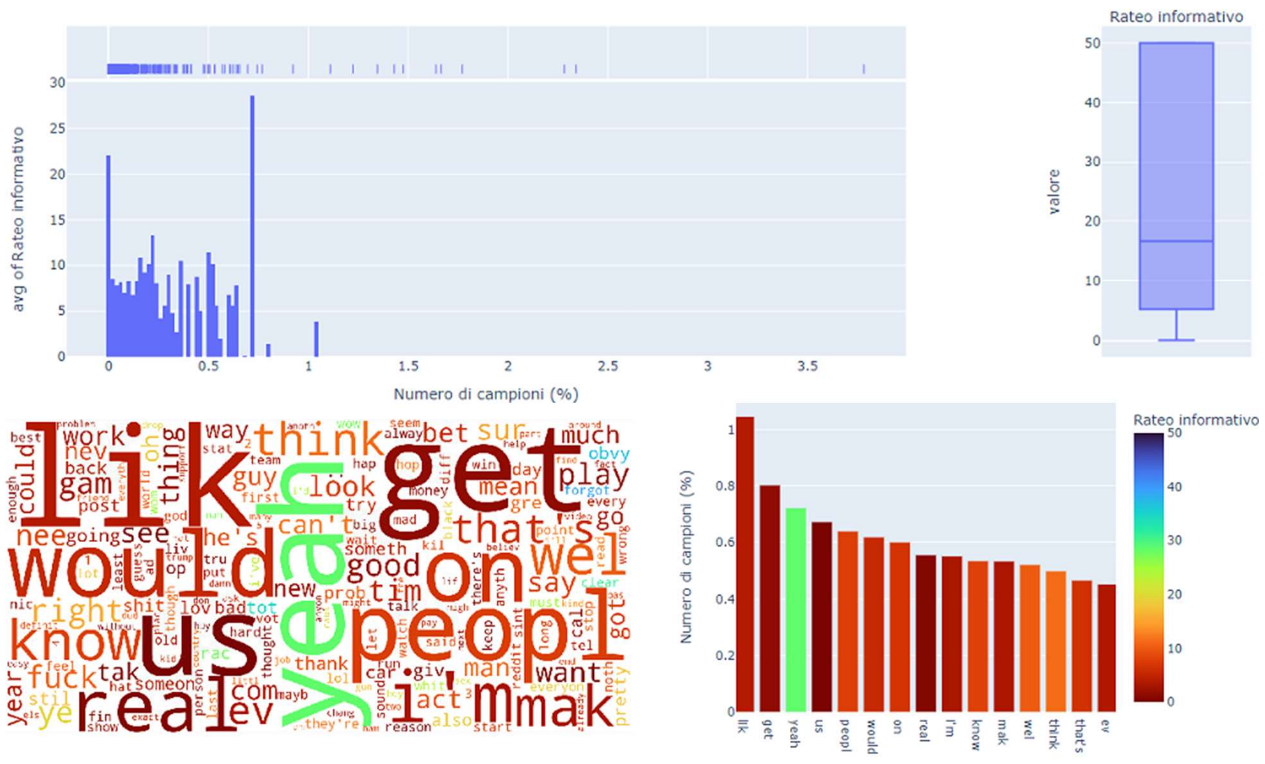


Figure 1.15: Rate distribution for text tokens without stopwords and with stemming

At Figure 1.15 we finally note that applying stemming to the text without a stopword does not produce a noticeably different result; however, in the wordcloud it appears that the rate has dropped in the higher frequency tokens; on the other hand, in the boxplot it appears to have increased overall (because the third quartile is higher).

After analyzing text types individually, it is useful to perform an aggregate analysis, calculating weighted mean and standard deviation (as previously done for context).

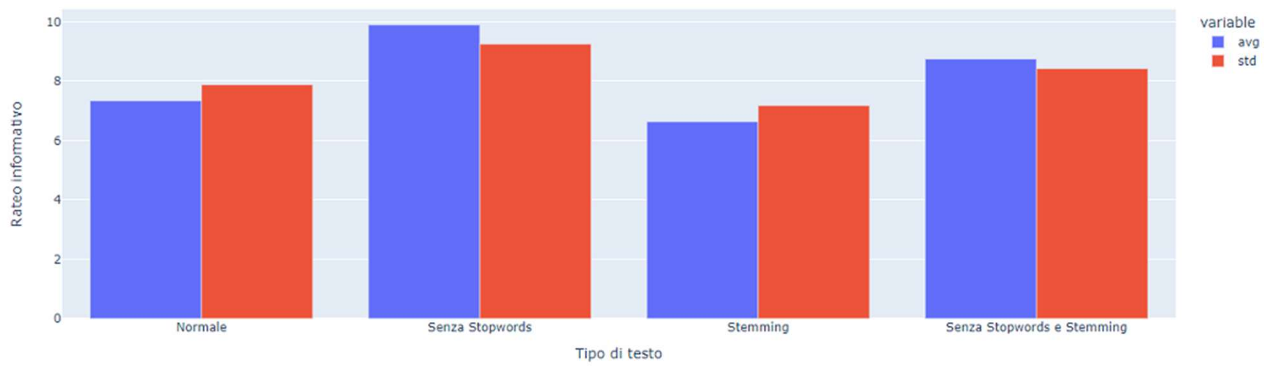


Figure 1.16: Mean and weighted STD of the information rate of text types.

The Figure 1.16 provides us with the overview useful for deciding which type of text to use. As seen in the individual analyses the removal of stopwords increases the quality of the text, whereas, the use of stemming is more relative; in fact, the types "Without stopwords" and "Without stopwords and Stemming" have averages of 9.89% and 8.74%, so given the small difference, the choice of text type (between the two) is more arbitrary. I chose the "Without stopwords" type for two reasons: the average is higher; and, The adopted model applies a pre-trained preprocessor and embeddings, which should also be able to take advantage of the information that stemming would remove.

After analyzing the various types of comment text, we can process parent text equally; since, since they are also comments we assume that they are very similar.

1.4 Analysis of Text and Parent lengths.

Another possible useful piece of information could be the length (in the sense of number of words) of the text and parent. In fact, this information is lost due to preprocessing and padding operations (an operation, required by the model, that brings all sentences to the same length). It could be useful, in that, they do not represent a large amount of data and would serve to highlight contrasts and not between text and parent.

Before such a feature can be included in the model, a brief analysis phase is required, which I performed by examining the distributions of the lengths, the distributions of the information rate at different lengths; then ending with the calculation of the information rate at the combinations of the lengths of the two features.

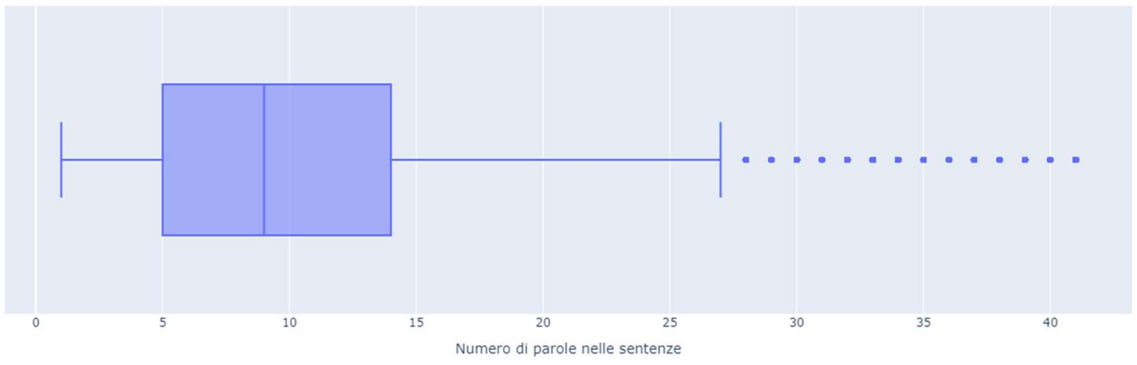


Figure 1.17: Boxplot representing the distribution of parent length.

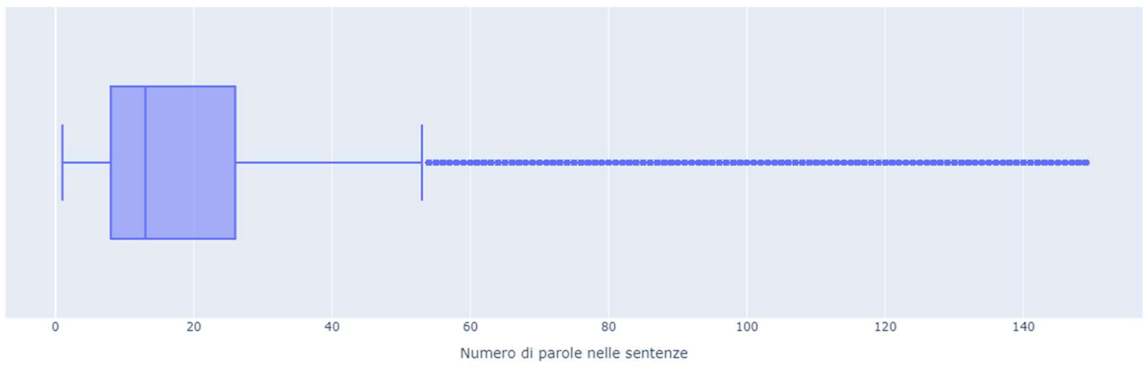


Figure 1.18: Boxplot represent the distribution of parent length.

As can be seen from Figure 1.17 e Figure 1.18, parent tends to be longer than text; in fact, parent has as its first quartile, median and second quartile respectively: 8, 13, 26; while, text has: 5, 9, 14. Moreover, if we go to consider the extremes of the distribution and outliers the difference is even more striking.

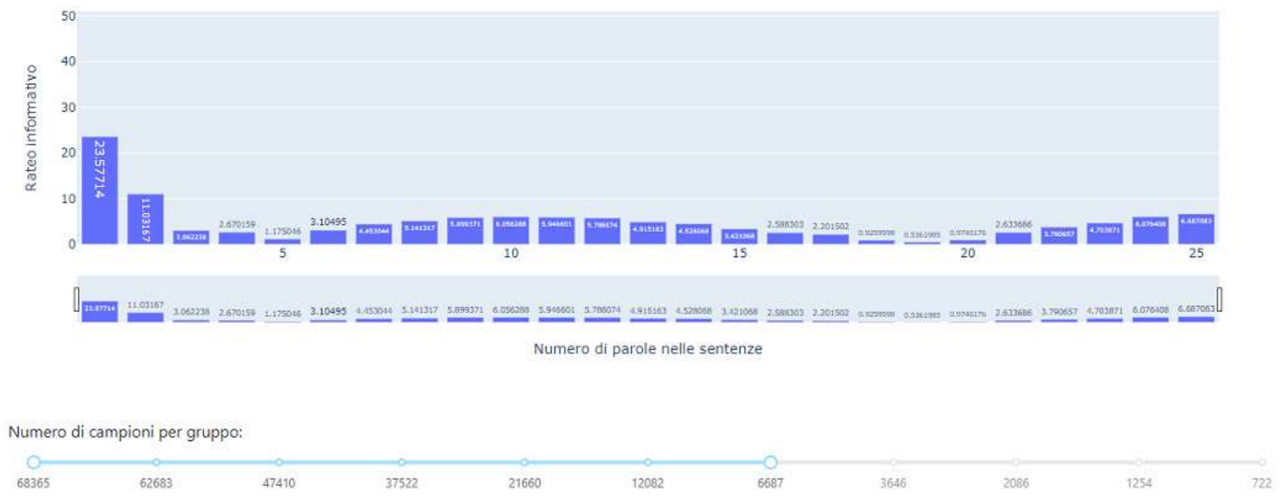


Figure 1.19: Histogram represents the distribution of information rate for text length, note: only lengths with a significant frequency (>40% of the distribution) are filtered out

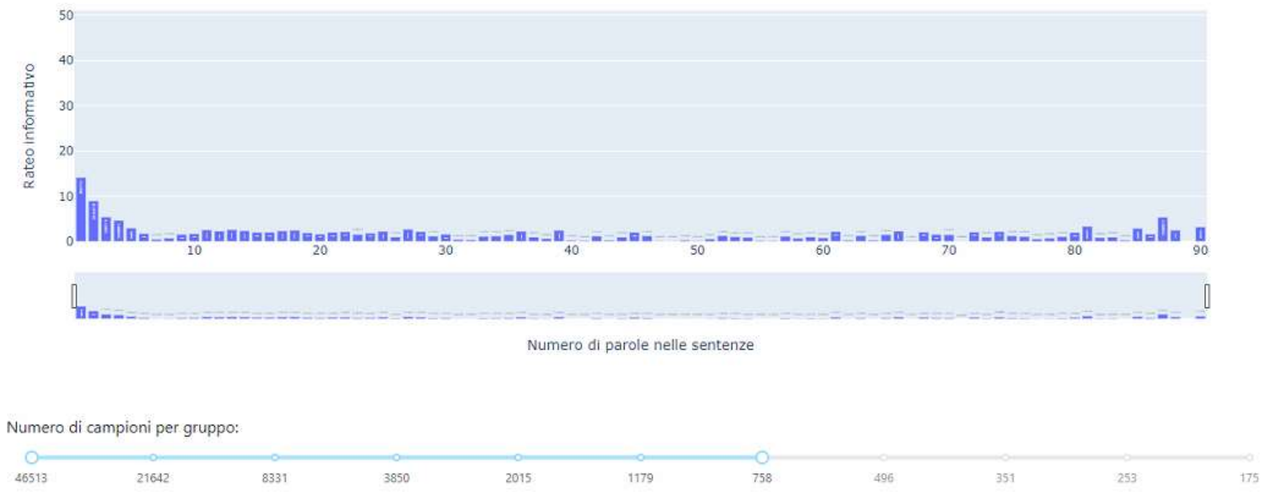


Figure 1.20: Histogram representing the distribution of information rate for parent length. Note: only lengths with a significant frequency (>40% of the distribution) are filtered out.

Comparing the Figure 1.19 and Figure 1.20, we notice that: the parent has a much more variable length (as seen previously in Figure 1.18); the length of the text generally has a higher information rate; in both cases we have a higher information rate with shorter lengths.

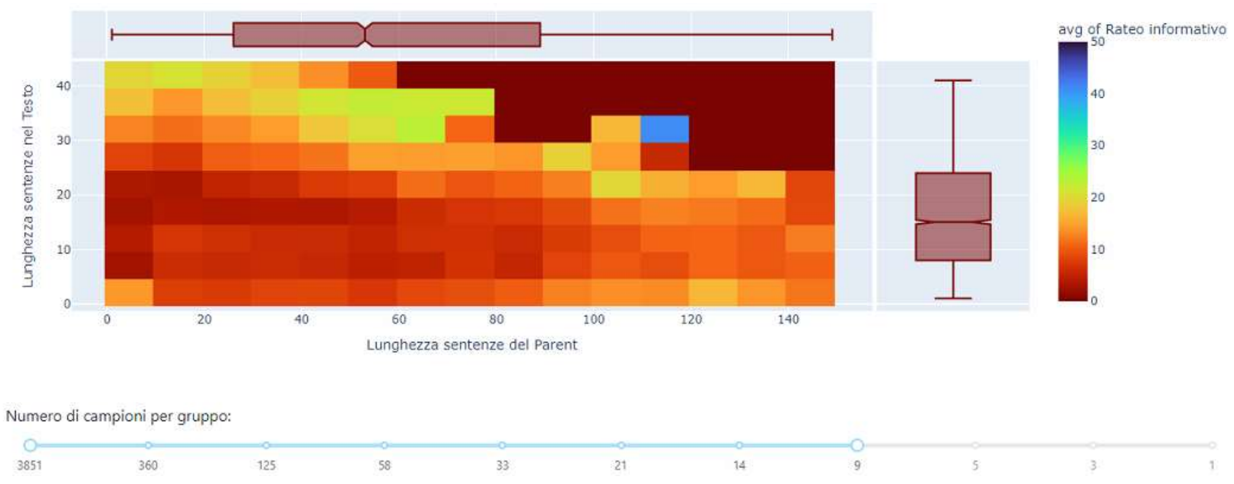


Figure 1.21: Heatmap representing on the cells the average information rate of the associated lengths of text and parent

From Figure 1.21 we note that:

- The heatmap was filtered to eliminate lengths with nonsignificant frequencies (>40% of the distribution).
- There is (as expected), in the upper right part an area with information rate 0, because at those values the samples were filtered.
- The average information rate is not very high; but, one can identify an area with lengths between 40 and 80 for parent and 30 and 40 for text where the rate exceeds 20%.
- A high rate area (40%) is present, however, which is insignificant because it has a low number of samples (increased filtering in fact disappears).

So as a result of the analysis, the two lengths do not have a very high rate; however, since they do not require many parameters from the model, and thus are not likely to add too much noise, they could still be useful to it.

Chapter 2: Model

In this chapter I will describe the model used, first illustrating, on a theoretical level, the model and the reasons behind my choice; then I will elaborate on the neural architecture employed; and then present the results obtained.

2.1 Short presentation on Transformers

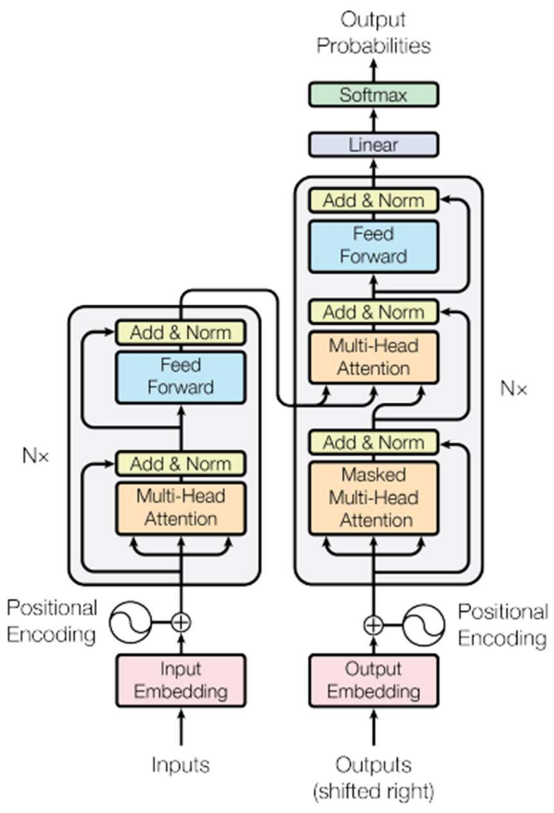


Figure 2.1: Generic architecture of a transformer

Transformer models were introduced in 2017 with the paper "Attention Is All You Need" by Vaswani et al. They represent a neural network architecture that relies heavily on the use of attention mechanisms. Unlike RNN and LSTM models (which were the previous standard), Transformers do not depend on sequential structure, but use attention to capture long-range relationships between words within text; this makes them particularly suitable for processing text sequences of varying length.

It consists of two basic components: the encoder, which processes the input text and creates a contextual representation of the words; the decoder, on the other hand, takes that representation as input and generates a translated or generated sequence of words.

Transformer-based models are promising for tasks such as sarcasm because of their ability to capture and model contextual relationships between words, even in very long texts.

2.2 BERT and TransformerEncoder

In my model I adopted two models of the transformer class (BERT and TransformerEncoder), they are used to process text and parent simultaneously, so as to detect ties and contrasts, and are placed one after the other.

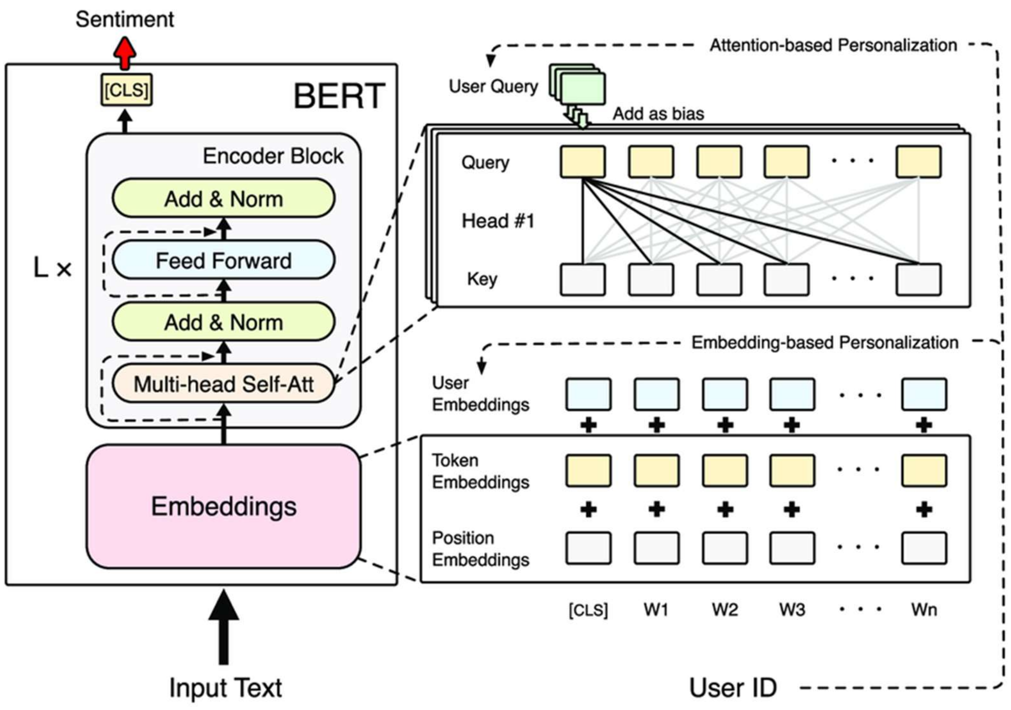


Figure 2.2: Illustrative diagram of BERT

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained Transformer-based model that uses a series of encoders to extract the text representation. Specifically, it takes as input one or two sentences (in this case text and parent), and after embedding (distributed representation of words) processes the representations using multiple sequential blocks of encoders (such as those in the transformer). This model is trained in two stages:

- Pre-training: in this first stage you have to pre-train the model on a large amount of data, without labels, in two modes:
    - Masked Language Model (MLM): in which random words are masked and the model will have to reconstruct them from the context
    - Next Sentence Prediction (NSP): in which the model must predict whether two sentences are consecutive or not in the original text.
- Fine-tuning: after pre-training, the parameters of the intial model are made static (untrainable), and layers are added that are trained to use the output of BERT for the purpose of the task at hand.

I used BERT because, being pre-trained, one can exploit its capabilities, derived from huge amounts of data and computational resources, without training it from scratch. Moreover, you can use it by exploiting the output of either or both modes; in my case, extracting the size of the representation corresponding to the CLS token (visible in Figure 2.1), I exploit the NSP that is more suitable for the sarcasm task, since, it encapsulates the semantic links between the two sentences.

TransformerEncoder is also an equal model to the encoder in the Transformer architecture, which I used for fine-tuning. I trained it to take as input the output representation of BERT, and reprocess it to extract useful information for the model to predict sarcasm.

2.1 The model used

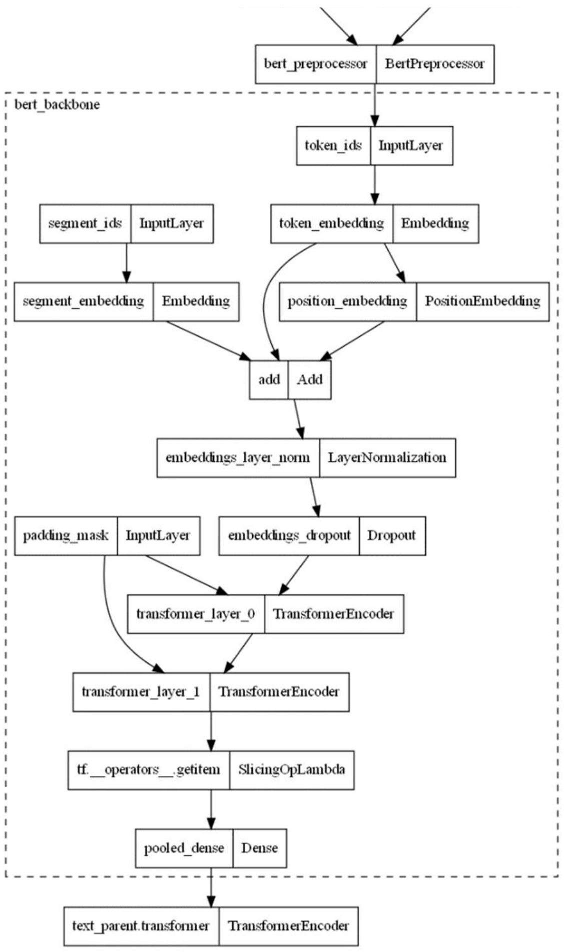


Figure 2.3: BERT used

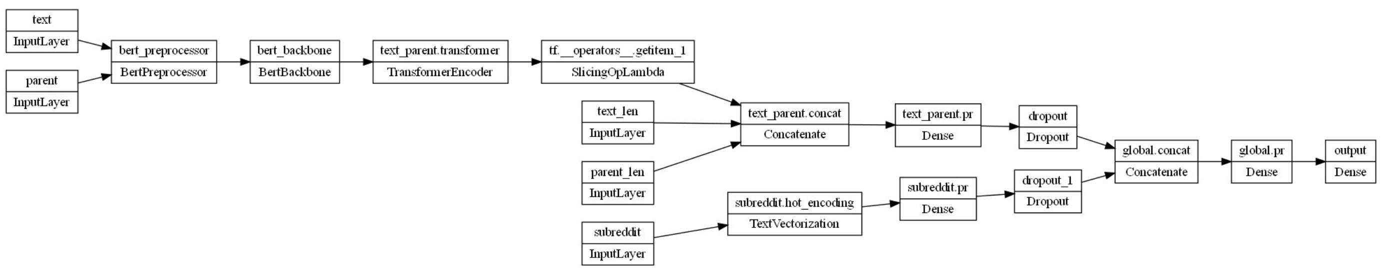
The model used is divisible into two parts:

- The first is to exploit BERT
- The second concatenates the result of the first to the subbreddit, text_len, and parent_len features

The first part is based on using BERT to process the two most important features simultaneously: text and parent. It consists of: a Preprocessor, which takes text as input, and tokenizes it so that it is compatible with the model; and the Backbone, which takes the tokens of the two sentences and extracts the representations.

Such a component can be downloaded from the keras library, in my case, I used the "bert_tiny_en_uncased" which does not support uppercase characters, and was trained on English wikipedia and BookCorpus. Of the possible ones it is the smallest, in fact it has "only" 4.39M parameters, while the largest has as many as 335.14M parameters.

Figure 2. 4: Representation of the model used; in which you can see the various layers with their names



Finally, the first part returns the representations to the TransformerEncoder called text_parent.transformer which, consisting of 4 heads with a total of 131K parameters performs fine-tuning. The encoder result will then be passed to the second part.

The second part processes the subreddit separately, first performing multi-hot encoding and then processing the result with a layer of 25 neurons (100K parameters), resulting in a 25-length vector. In particular, hot-econdig is performed as follows:

- In the adapt phase (which precedes training) create a vocabulary of the most frequent names;
- The size is set to 4000, which is two-thirds of the number of unique elements with more than one text (as can be calculated from Figure 1.22: 14545 * 0.4)
- For each input name it returns a vector of 4001 elements, with only one 1 whose position is the same as it has in the dictionary (there is an additional dimension for external values)

In addition, the second part concatenates the result of the first part with the lengths of the text and parent, resulting in a vector of 130 values (2 of which are the lengths); and then a layer of 100 neurons (13K parameters) is applied, with which a 100-length vector is obtained. Then that result is concatenated with the subreddit result, and then passed to a first layer with 20 neurons (2520 parameters). That finally reaches the last neuron, which with the sigmoid function, returns the probability of belonging to the sarcastic class.

2.3 Problems faced during the training

This model, as much as it leverages BERT, because of the total number of 246K parameters and nearly a million rows was eternal to train. Therefore, I purposely decided to focus on only one model, without tuning the hyperparameters and choosing them as best as possible from the analyses performed and some tests performed.

The training was only possible because of Colab (Google's cloud), which provides Tesla T4 GPUs that are dozens of times more performant than Nvidia's common top-of-the-line (RX and GTX). With the T4s a single epoch took just over an hour and 30 minutes; while with the Intel I7-6700k processor, on my desktop computer, the times were several days.

Chapter 3: Results

In this chapter we will look at the results I obtained in training the model; first we will look at the training history, then at the results on the validation dataset, and finally at the testing dataset (so far untouched).

3.1 History of training

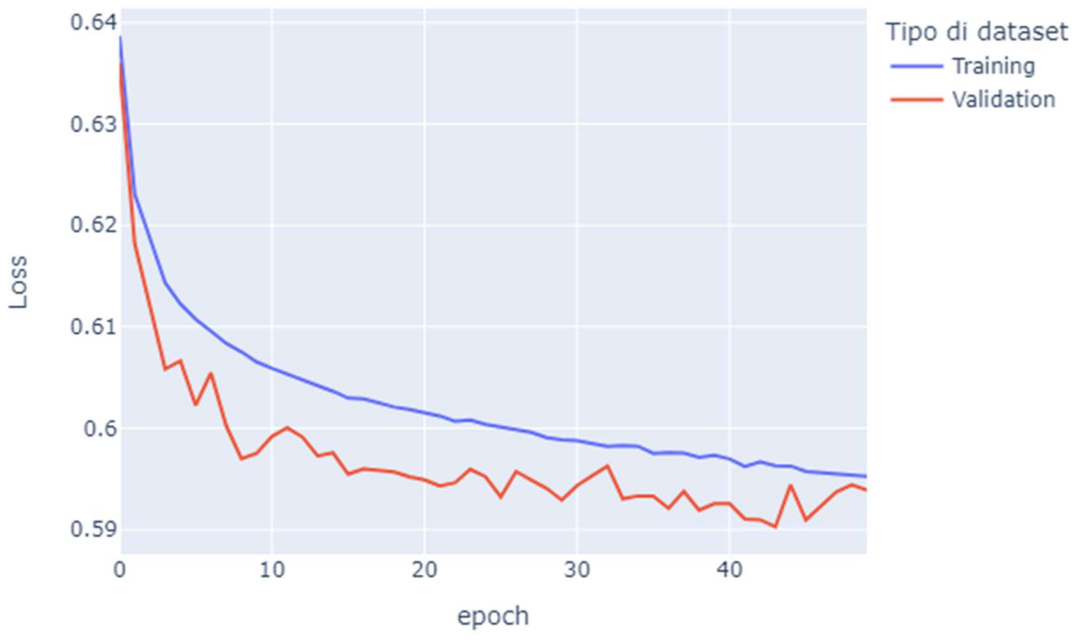


Figure 3.1: Loss function during training, comparing values on the training and validation datasets.

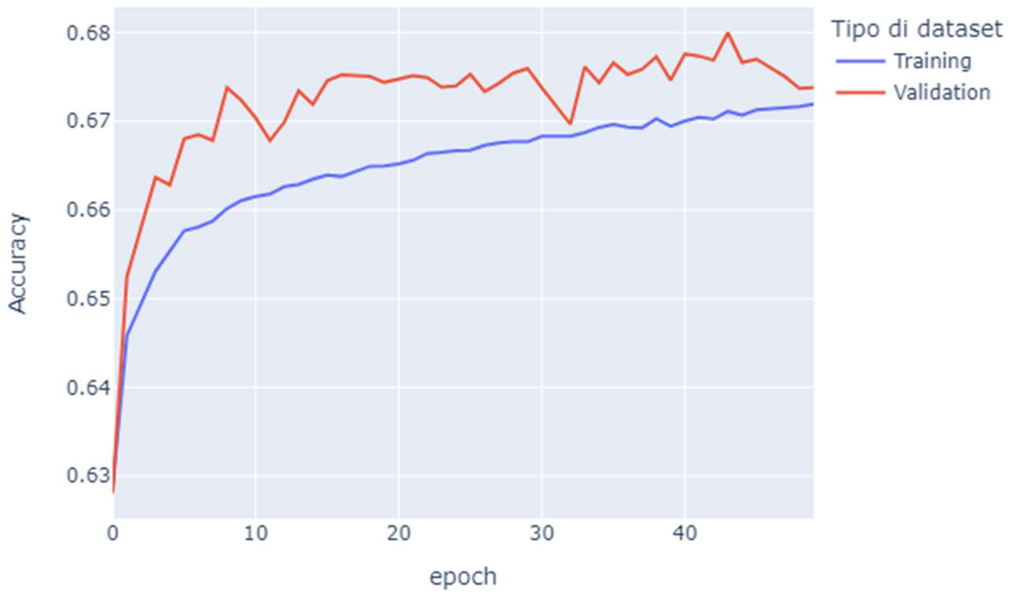


Figure 3.2: Accuracy metric during training, comparing values on the training and validation datasets.

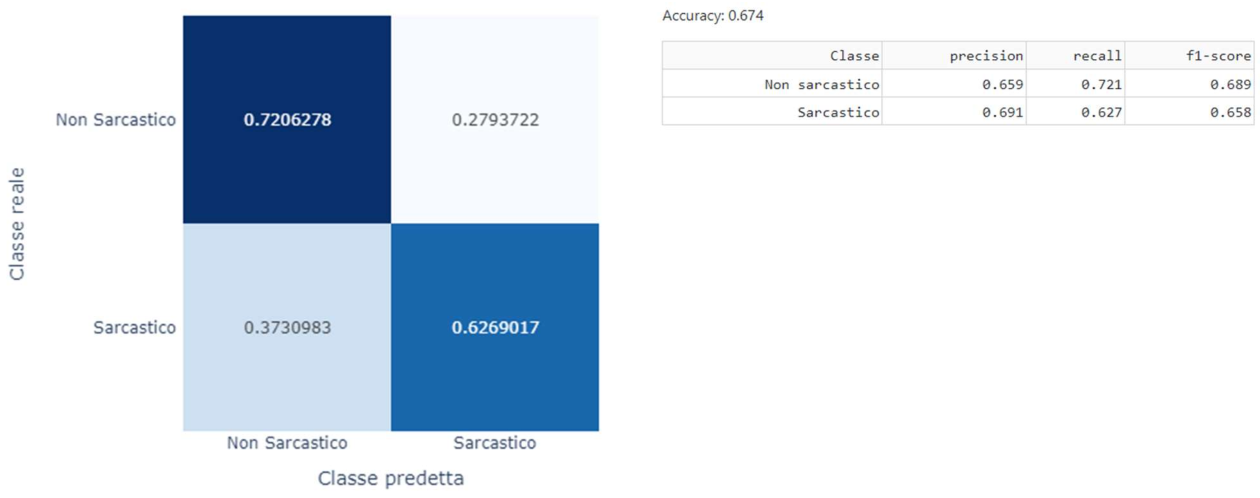
Looking at Figure 3.1 e Figure 3.2, we can observe that:

- The model after training, which lasted a total of 50 epochs, achieves an accuracy of 67.2% (from 62.8%) on training, and 68.0% (from 62.8%) on validation.
- Both Loss functions drop gradually:
  - In fact, the Earlystopping callback set, which stops training after 5 consecutive epochs without an improvement, never triggered;
  - So the model is definitely not overfitting.
- Given the slow descent it is likely that the model is going underfitting, this is solvable:
  - Eliminating Dropout layers, which introduce a random component useful for generalization.
  - Increasing the model parameters, specifically those placed in the text_parent.transformer or those in the last layers.
  - Continuing the training for multiple eras.

3.2 Results on the Validation dataset

I elaborated on the results obtained on the validation dataset (exactly as I will later perform for the Test), through the following graphs and calculations:

- Confusion Matrix: to get an overview of predicted versus actual labels; obtaining the values of true and false positive and true and false negative.
- Accuracy, precision, recall, and f1-score by label:
  - Accuracy: number of correct predictions/total predictions;
  - Precision: True Positives / (True Positives + False Positives), measures the correct positive ratings compared to the total positive ratings obtained;
  - Recall: True Positives / (True Positives + False Negatives), measures correct positive classifications against all true positive instances.
  - F1-score: takes into account the previous two metrics to provide an overall measure;
- Histogram: represent the probabilities obtained with respect to the actual classes.
- ROC curve: which examines the relationship between the True Positive Rate (TPR) and False Positive Rate (FPR) as the classification threshold changes;



Accuracy: 0.674

Classe	precision	recall	f1-score
Non sarcastico	0.659	0.721	0.689
Sarcastico	0.691	0.627	0.658

Figure 3.3: Confusion Matrix and statistics for the validation dataset.

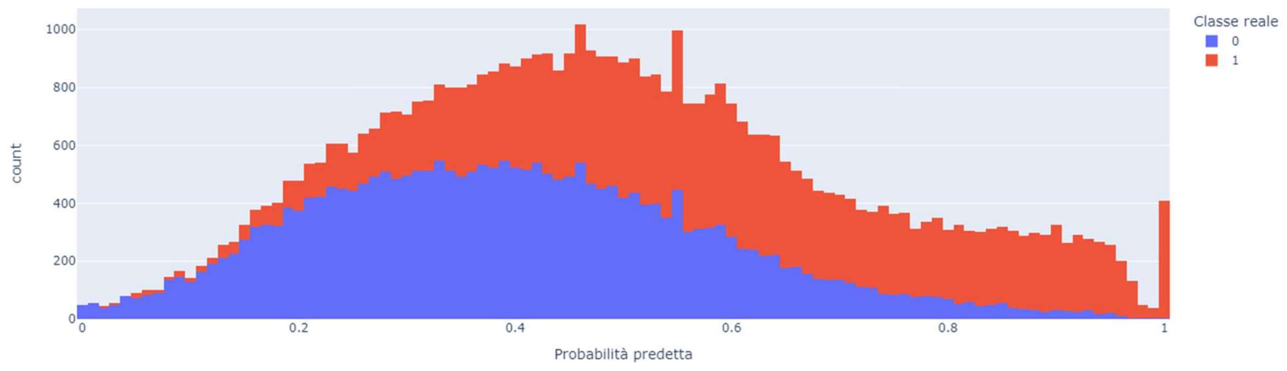


Figure 3.4: Histogram comparing the predicted probabilities against the actual classes; for the validation dataset.

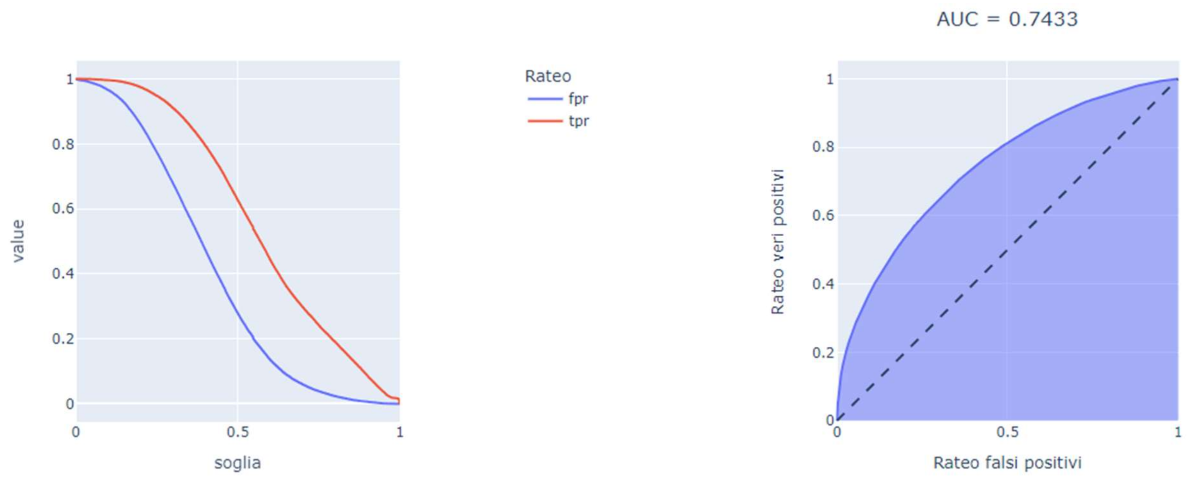


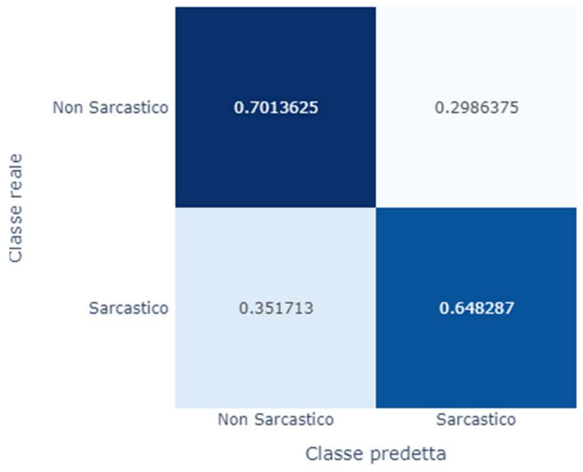
Figure 3.5: ROC, AUC, and FPR and TPR ratios at different thresholds; for the validation dataset.

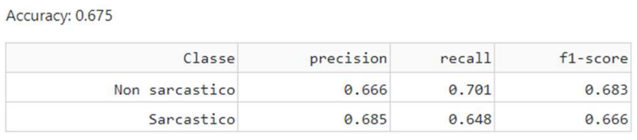
Analyzing the results obtained; viewable in Figure 3.3, Figure 3.4 e Figure 3.5, we can see that:

- The model significantly better predicts negative instances; in fact, the rate of true negatives is 72% versus 62.7% of true positives; this is also confirmed by the F1-score and recall for the negative label.
- The model has a high rate of false positives, compared to false negatives (37.3% vs. 27%); that is, it tends more to classify positive instances as negative than the other way around.
- The AUC of 0.743 indicates that the model has good discriminating power between classes, but not excessive; in fact, it is quite far from 0.5 but not very close to the value of 1.

3.3 Results on the Test dataset.

Proceeding now with the analysis on the Test dataset, I obtained:





Accuracy: 0.675

Classe	precision	recall	f1-score
Non sarcastico	0.666	0.701	0.683
Sarcastico	0.685	0.648	0.666

Figure 3.6: Confusion Matrix and statistics for the test dataset.

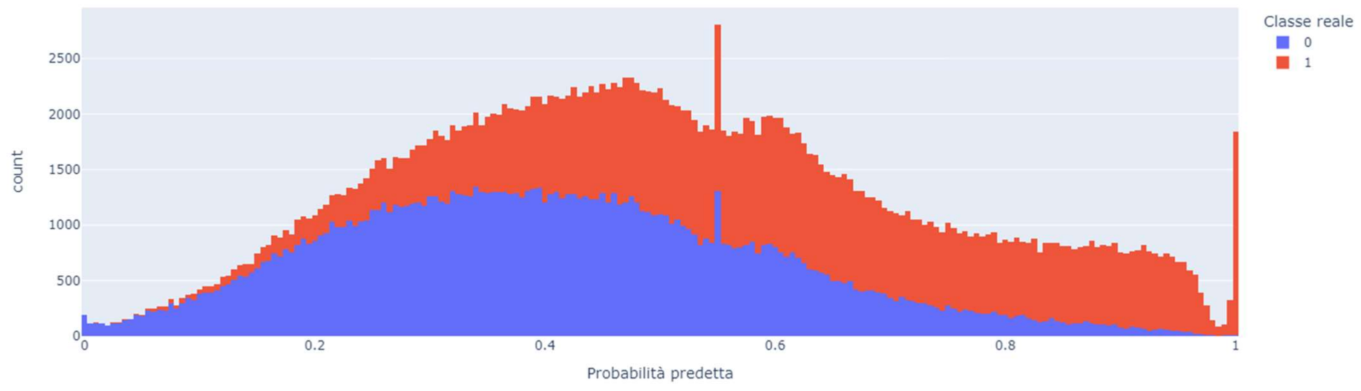


Figure 3.7: Histogram comparing predicted probabilities versus actual classes; for the test dataset.

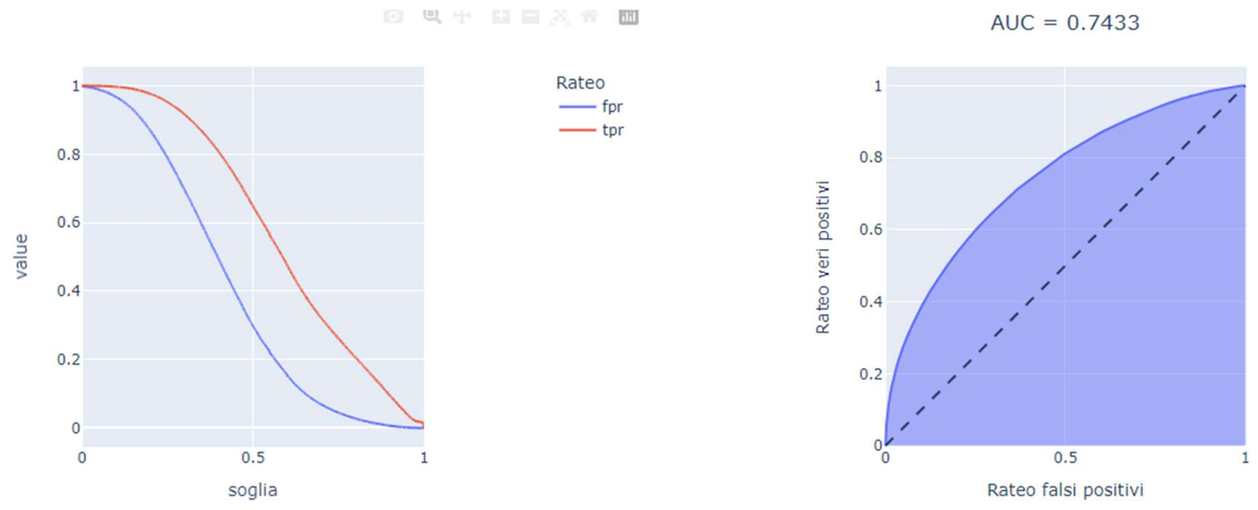


Figure 3.8: ROC, AUC, and FPR and TPR ratios at different thresholds; for the test dataset.

Looking at Figure 3.6, Figure 3.7 e Figure 3.8, we note that the considerations made for the validation data have not changed much, specifically:

- The model confirms its generalization ability, as the AUC has not changed (considering the approximation to 3 decimal places),
- The rate of true negatives has decreased, but the rate of true positives has increased; therefore, the model tends to distinguish positive instances better.
- Accuracy remained almost unchanged (from 67.4% to 67.5%).
- Precision, Recall and F1-score are less polarized between the two classes, compared with the previous dataset.

Conclusion

In this project I approached sarcasm detection, a particularly difficult branch of NLP, using new and more experimental approaches (such as Transformers), albeit on a small scale.

The dataset made available, with its 5 features and over 1M instances, after a brief preprocessing phase, allowed me to perform several analyses. In which I calculated the information rate for different context elements, and for different text types.

Finally, I processed the datasets, in accordance with the analysis done on the Training one, and then trained a model based on BERT and TransformerEncoders.

In conclusion, the analyses performed, in the first chapter, were a useful tool to improve the accuracy of the model; also reducing computational resources and time, as, they allowed me to exclude superfluous data. The model shows excellent generalization capabilities, but also some degree of underfitting.

In the future, with better computational resources, tuning of the hyperparameters could be done, so as to identify the correct number of neurons in the inner layers and the number of heads of the TransformerEncoder; as well as, to be able to experiment with additional inputs or with BERT at larger sizes.

Appendix: some technical details.

Por being able to run the code from scratch, it is recommended to run the main.py file, put in the project folder, which first runs the data processing file, then the model processing file, and finally some further processing, necessary for the Dash application.

In order to train the model from scratch, you must flag the MODEL_LOAD constant located in the costant.py file to False.

To start the Dash application, one must run the index.py file in the "app" folder; however, despite the optimization operations performed, due to the large number of instances, it is quite heavy.

The Dash application is structured on 6 pages:

- Dashboard: this is the home page and contains the initial data information;
- Sentence Len Analysis: contains analyses on parent and text lengths;
- Context Exploring: contains analyses of context elements.
- Text Analysis: contains the analysis performed on text types.
- Model Training: contains the analyses performed on the model training, and validation dataset.
- Model Testing: within which are the analyses on the test dataset and the model demo.