Text Mining and Natural Language Processing

2023-2024

[Pietro Saveri: 524921, Matteo Salami: 513974]

[Project SWSW-SD]

Introduction

Nowadays, in the era of digital media, humanity produces and consumes an enormous number of headlines which, more often than not, could be misleading especially when sarcasm - the use of irony to mock or convey contempt - is involved. The aim of our project is to provide a robust model able to reliably distinguish (or discern) between a sarcastic and a non sarcastic news headline.

Data

The dataset taken into consideration is the "News Headlines Dataset For Sarcasm Detection" Dataset.

The different headlines are taken from two news websites; TheOnion aims at producing sarcastic versions of current events and we collected all the headlines from News in Brief and News in Photos categories (which are sarcastic). We collect real (and non-sarcastic) news headlines from HuffPost - an American news aggregator and blog.

Each record in the JSON file consists of three attributes:

-is_sarcastic: 1 if the record is sarcastic otherwise 0

-headline: the headline of the news article

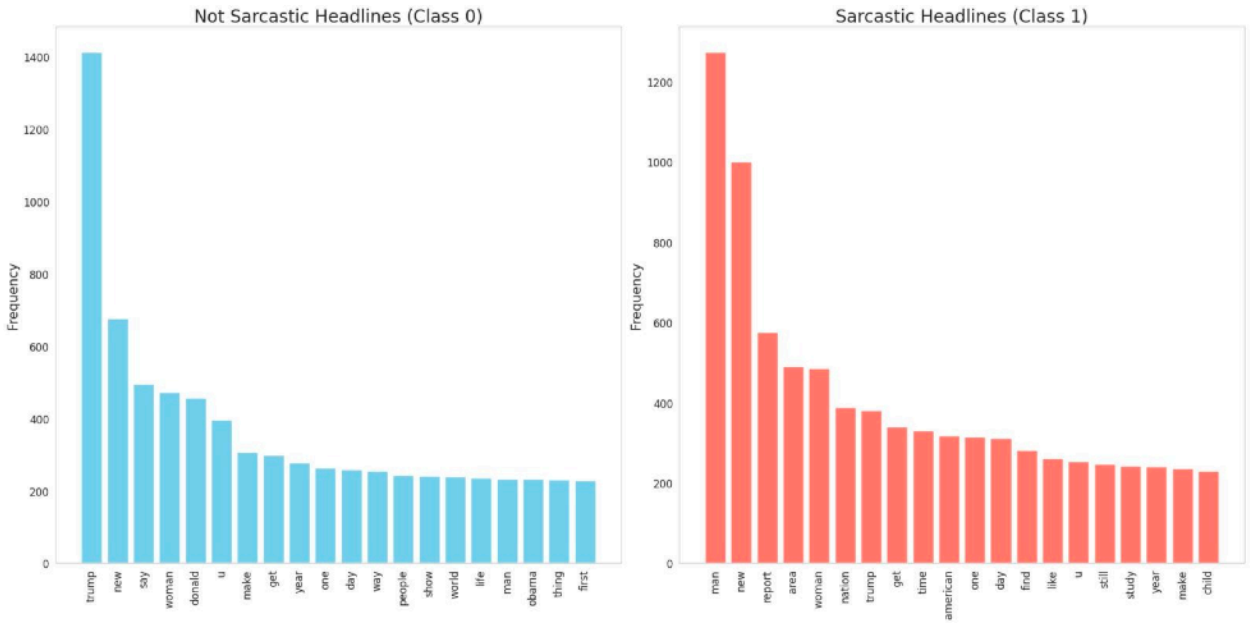
-article_link: link to the original news article. Useful in collecting supplementary data

Some graphs for data visualisation:

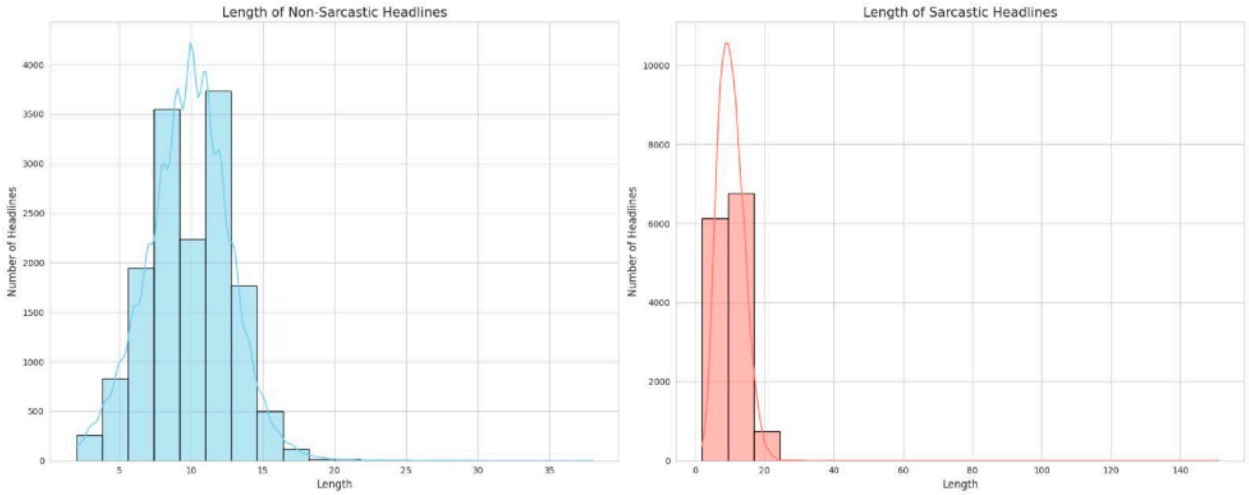
The histogram below shows the 20 most frequent words for both sarcastic and non sarcastic headlines. For example “trump” appears the most in non sarcastic headlines with a frequency of around 1400.

It is also interesting to note that some words, e.g. “get” or “day”, are seen often in both classes and with similar frequencies, which means that they won’t be instrumental for the tackled problem.

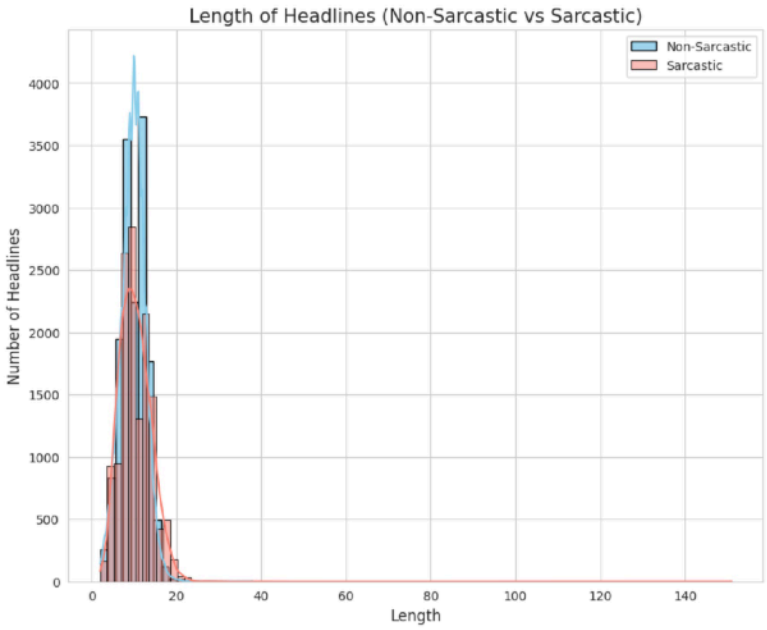
Notice that “Donald” or “Trump” are so frequent because the news websites from which the headlines are taken are American websites.



With this second histogram we provide some visualisation concerning the length of the sentences

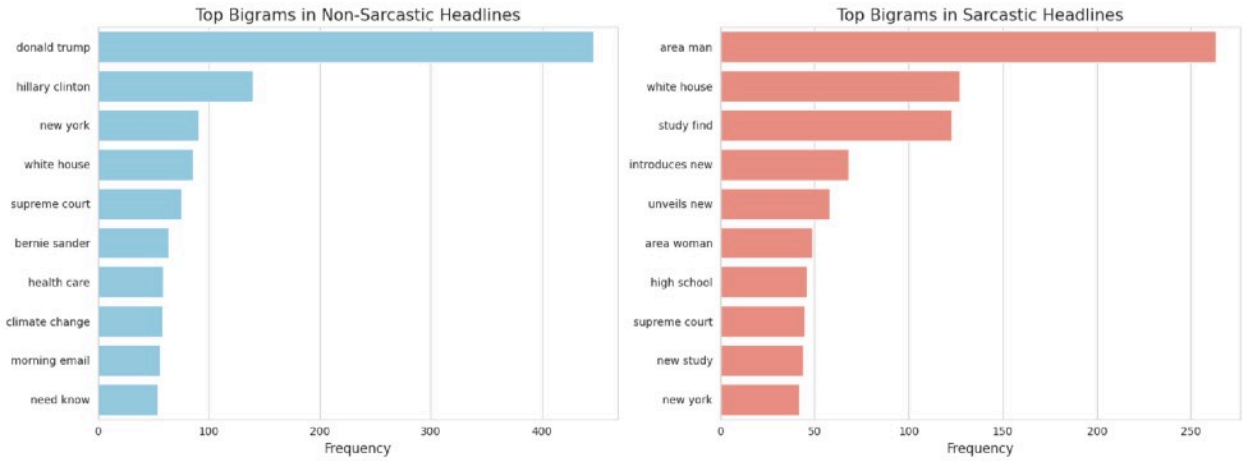


(Here we superimpose the two charts to better envision the differences between the two classes)



Some related additional information:

- Total number of headlines: 28617
  - Number of sarcastic headlines: 13633
  - Number of non-sarcastic headlines: 14984
  - Average word count: 10.05
  - Average word count (sarcastic): 10.31
  - Average word count (non-sarcastic): 9.82



This last bar chart shows the Top 10 Bigrams in both classes each with its relative frequency. These are the top 10 Bigrams after some text preprocessing e.g. stop words removal.

Methodology

For the first proposed model:
After loading the dataset we performed some minor steps like removal of duplicates and of an unwanted column. Expanding contractions came next, this step would be rendered useless by the next one i.e. stop words removal but when using embeddings methods like BERT (which was later used) is generally beneficial to keep stop words (Contextual understanding, Pre-Training consistency…), and as a consequence it is also beneficial to expand the contradictions (Improved Tokenisation, Vocabulary matching …). We performed the latter by taking a already defined dictionary, containing the contracted form with the corresponding extended form, and applying it to the headlines.
As anticipated the following step is stop words removal i.e. removing words that occur commonly in the corpus, along with punctuation removal. We do this by exploiting the set of stop words defined in the "nltk" library and the string library respectively. In between these two steps, before stop words removal and after removing punctuation the model performs tokenisation with the "nltk" tokeniser. Then comes lemmatisation which is generally more precise than stemming since it is more context sensible and linguistically accurate.
For later purposes, mostly convenience, at this point we create two new columns in the dataset one with the processed headlines one with strings and one with lists.

From this point forward we combine vectorising and embedding methods with the Logistic Regression (LR) classifier in order to determine which one yields the better results. We use the same classifier for each method so that bias is avoided.

The first method used is the Bag of Words (BOW) which uses a representation of text that is based on an unordered collection of words. By operating in such a way it disregards any non-trivial notion of grammar but captures multiplicity.

The second is Term Frequency Inverse Document Frequency (TF-IDF) computed by multiplying TF with IDF. It essentially is a calculation of the relevance of a word in a document relative to a collection of documents. The same calculation is repeated for every word in the corpus. So finally we obtain vectors where each entry represents the relevance to the word to one of the vectors.

Third method is a pre-trained Word2Vec, a popular embedding method which in essence creates vector representations (known as embeddings) in such a way that semantically similar words are mapped to nearby points in the vector space. Similarity between words is determined using cosine similarity which is based on the cosine of the angle between the two vectors taken into consideration.
Once the embeddings for each word are calculated they are once again fed into a logistic regression

model through which we can calculate precision, that is the metric used, for this method and for the previous ones, to perform comparison.

We save all of these calculated accuracies in a dictionary which allows an easier comparison and stands as the backbone of the following table:

No.	Feature Extraction Method	Accuracy
1	BERT	0.937456
2	BOW + TF-TDF	0.797869
3	Bag of Words	0.786688
4	TF-IDF Vectorizer	0.786513
5	GloVe	0.69427
6	Word2Vec	0.67348
7	FastText	0.640636

The table shows all the implemented vectorising and embedding methods combined with their accuracy (always using logistic regression)

FastText, in the code, is what comes behind the Word2Vec, the order follows from the fact that FastText is an extension od W2V. The aim is to enhance the quality of word embeddings by considering subword information and it does that by breaking down words into smaller units called n-grams (subwords). By doing this it can also generate embeddings for out-of-vocabulary words (OOW) and as a consequence it is also better at handling rare words.

Moving on we find a combination between two already described methods which are BOW and TF-IDF. The combination of the involves generation of BOW vectors for each document in the corpus, where each vector component represents the term frequency of a word. On these vectors in order we calculate TF (obtained normalising the vector) and IDF (calculated for each term across the corpus) and finally the TF-IDF. So at the end of the process you get a matrix of this kind:

	0	1	2	3	4	5	6
0	0	0	0.453295	0	0.767495	0	0.453295
1	0.608845	0.608845	0.359594	0	0	0	0.359594
2	0	0	0.359594	0.608845	0	0.608845	0.359594

GloVe (Global vector for word representation) is the second to last method which constructs word embeddings by leveraging global statistical information from a corpus and not based on local context. GloVe's key idea is to use the ratios of co-occurrence probabilities, coming from a co-occurence matrix previously built, to capture semantic relationships. If two words, i and j, have similar contexts, their embeddings should be similar.

The last implemented method, and also the one that yields the best results, is BERT which stands for Bidirectional Encoder Representations from Transformers, at heart BERT leverages the Transformer architecture to capture contextual relationships in text. It processes text in both directions unlike previous models meaning that it can understand and generate language by considering both the left and right context in all layers of the model.
With BERT we also used a L2 regularisation to mitigate a possible overfitting.

Once the best feature extraction method has been determined, based on the metric of accuracy, we exploit its encodings to later use a classification method to create our last model. We using a RandomizedSearchCV and cross validation to find the best combination, among 30 different ones, of dimensionality reduction algorithms and classifiers. After the best model is found (best_model) we perform hyperparametrization tuning to enhance performance of the latter. Finally we plot the learning curve which helps us to analyse overfitting, which in our case is present and the model is slightly overfitted.

For the second proposed model:
We load a model, more specifically a Bert variant, from hugging face called “distilbert-base-cased” (where cased means it takes in consideration the difference between upper case and lower case).
Then we fine tune the latter on our dataset and we load it on hugging face so that anyone can access it and try it (click here).
NOTE:

* 0 = Not Sarcastic
* 1 = Sarcastic

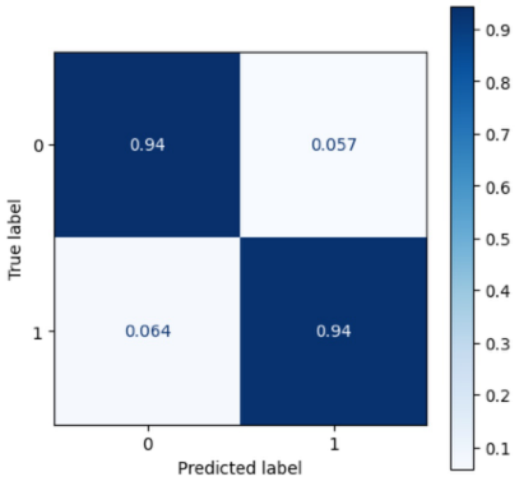
Finally we use the fine tuned model to compute the predictions and measure accuracy.

Result and Analysis

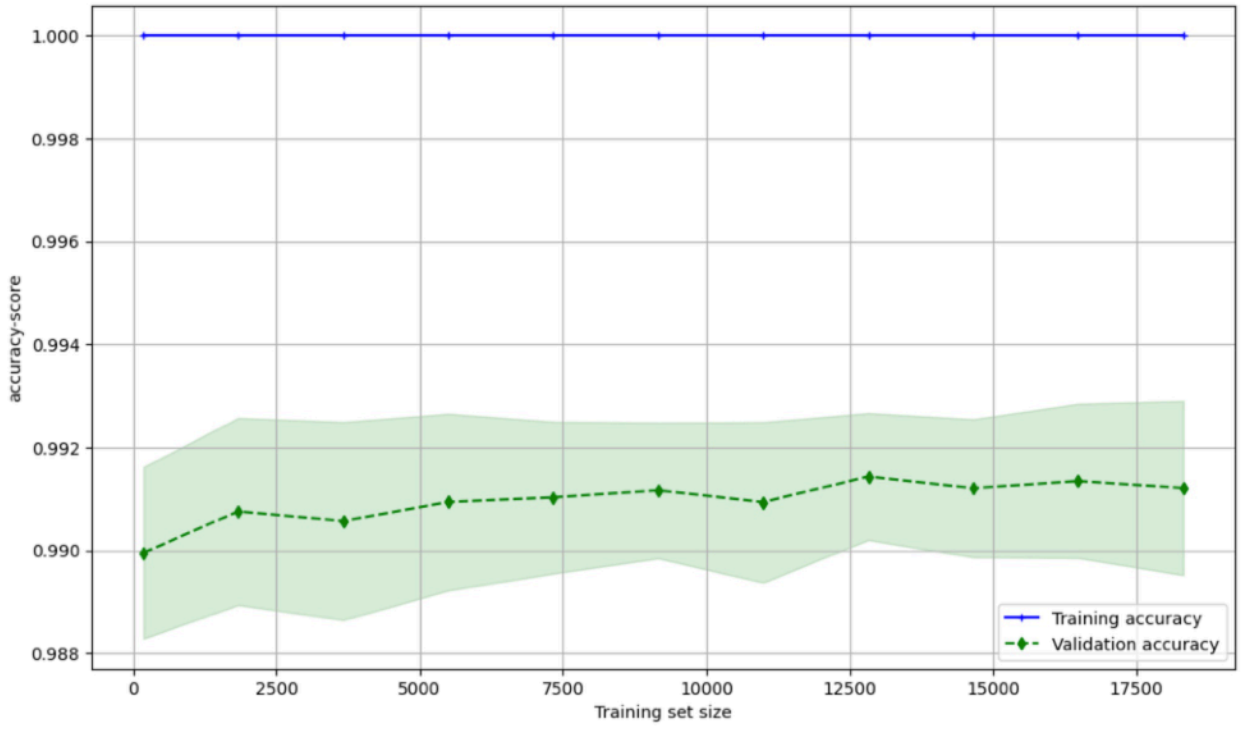
BERT was the embedding method that, combined with the simple classifier of Logistic regression, results in the highest accuracy.
BERT's success can be attributed to its ability to capture contextual nuances and semantic relationships within the headlines. Sarcasm often relies on subtle contextual cues, which BERT's bidirectional context modelling can effectively capture. It's embeddings likely encode richer semantic information relevant to sarcasm detection compared to simpler methods like Word2Vec and TF-IDF.

The process of selecting the best model and consequently fine-tuning its hyper parameters resulted in the choice of the following combination: no dimensionality reduction technique, as for the classifier, Random Frost was the selected one with number of estimators equal to 500.

Here below is displayed the confusion matrix:



While here is the learning curve showing the presence of a slim overfitting:



Conclusion

In extreme summarisation the steps where:
- Preprocessing and tokenisation
- Comparison of different feature extraction models in order to discover the best performing one
- implementation of the embeddings coming from second step as input of a ML algorithm

The project was done almost entirely by both of the team members simultaneously, since we met in presence each time we made any progress. Also we listened to some suggestion of a third external person for the creation of the second proposed model.

The most difficult steps was most definitely the implementation of BERT because of errors that were solved with time
Also one of the biggest challenges was dealing with the colab runtime of the GPU which proved to be much less than it seemed!

AI policies

If you take advantage of any sort of AI assistance, you will be required to submit the specific prompts you used as well as a description of how the AI helped (or did not help) you complete the project.
This should go without saying, but if you are using AI assistance, you are also responsible for making sure it is correct before submitting it.

As our Artificially intelligent helper we utilized, probably as expected,ChatGPT , the large language model (LLM) created by OpenAI. The most help was provided on the data visualisation part, more specifically we exploited it to create the charts present in the initial portion of the model.
Also it was useful for understanding certain errors that came up mid development.

The exact prompt we made use of are:

I have a df of headlines and a Is_sarcastc column. Make 2 graphs one for the non sarcastic headlines and one for the sarcastic headline thanks!

Can you make this a piechart please

I have this code to plot the top n grams of the headlines,

can you also create 2 plots to show the n grams in the sarcastic headlines

and non sarcastic headlines thanks!

Can you please change the color of this 2 barplots to #87CEEB, #FA8072

what would be a nice to way to finish this project? maybe a plot of something? maybe a conclusion with a graph?Just a nice day to conclude the project

(The prompts for the errors we were not able to find them in a little time scramble, its no excuse, we forgot to put them before and remembered only at the last moment)