Sarcasm Detection

Patrick erickson, Parth Gosar, Derek Guidorizzi, Sahil Pardasani

March 19, 2025

GitHub

Abstract

Sarcasm often employs a form of implicit criticism or exaggeration that defies straightforward sentiment analysis. Positive words can paradoxically indicate a negative sentiment in cases of sarcasm classification, turning on its head the assumption of many sentiment-based classification tasks. Furthermore, the ability to obtain high-quality labeled datasets increases the complexity of the problem. Addressing these challenges requires approaches that prioritize a deep understanding of language use. Therefore, the main goal of this document is to integrate advanced contextual modeling techniques and incorporate cross-disciplinary insights from a myriad of different machine learning models to capture these subtleties presented by the nuances of sarcasm.

1 Introduction: Dataset Choice and EDA

Our dataset was curated from Huggingface, courtesy of Ziyuan111. Ziyuan curated this dataset from the larger self-annotated corpus for Sarcasm, which was ethically curated for training purposes. Upon initial loading of the dataset, we performed some preliminary checks to handle any missing data and class imbalances. Upon further inspection, we were able to deduce that the dataset had already been preprocessed, providing exactly 50% of the cases as sarcasm and 50% otherwise.

Intuitively, we obtain a better understanding of every case within the dataset. One of the most common methods of visualizing these differences for NLP tasks is to construct a word cloud, where words appear larger the more frequent they are. The following images were obtained by splitting the words into constituent arrays (tokenizing) and counting the frequencies of each of these words, after applying stopword removal, where words that contribute little to no contextual value are removed for better processing.

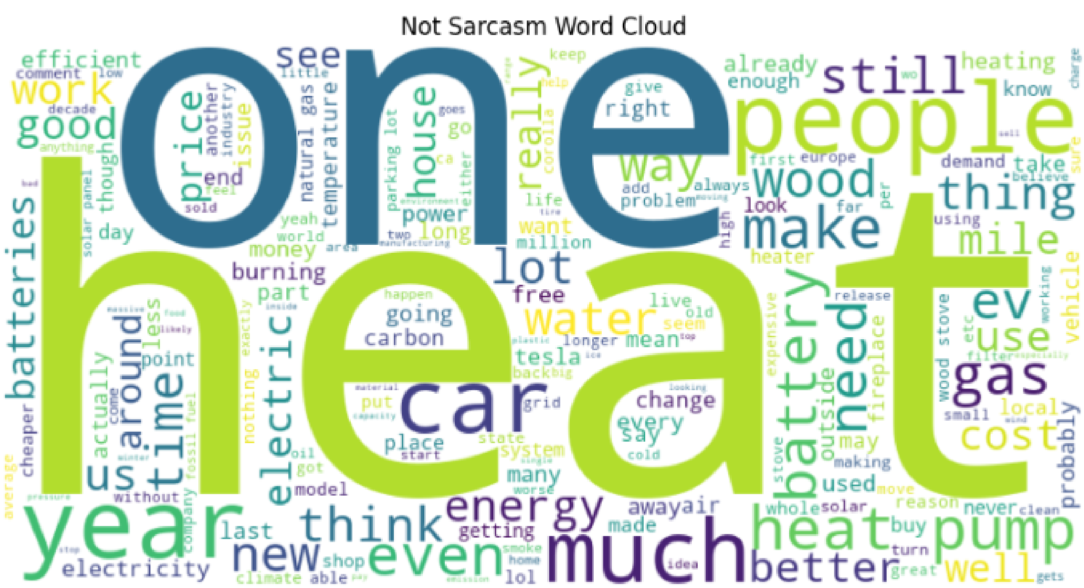

Figure 1: This is the wordcloud generated to represent the frequencies in the binary encoded 0's of the dataset, representing non-sarcastic remarks.

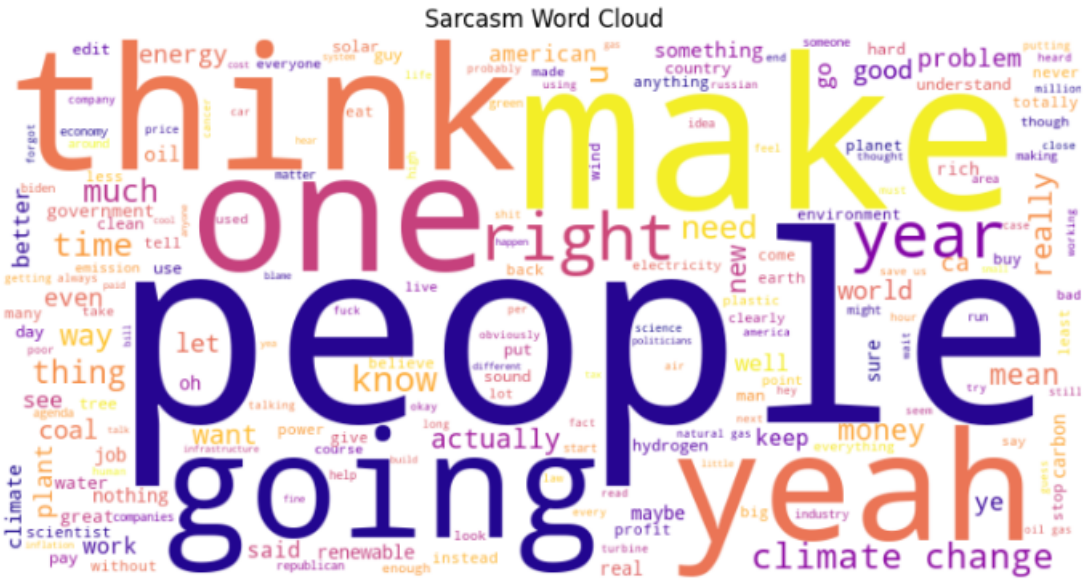

Figure 2: This is the wordcloud generated to represent the frequencies in the binary encoded 0's of the dataset, representing sarcastic remarks.

Based on the given representations, we can see that there are clearly some differences in the most frequent word choices, with some particularly glaring differences such as "heat" for non-sarcastic labels and "think" for sarcastic labels. There does seem to be some overlap as well, with people being totally prevalent. This suggests the need for further analysis.

We therefore decided to take a look at a different type of wordcloud: ngram wordclouds, or more specifically, **bigram wordclouds**. This fairly simplistic ngram representation helps us effectively capture co-occurrence contextualization and be another means by which we could augment our data to better classify the sarcasm within our dataset. Secondly, to get a better view of these bigrams we will be limiting them to the top ten most frequent bigrams for better visualization:

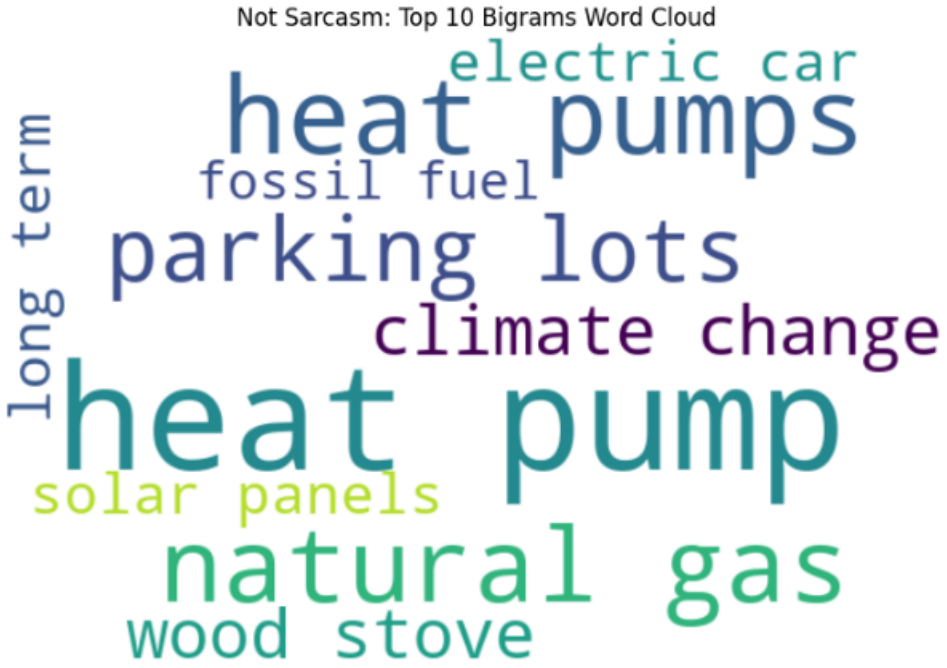

Figure 3: The top 10 non-sarcastic remarks tend to generally revolve around more coherent ideas, rather than the 1-word.

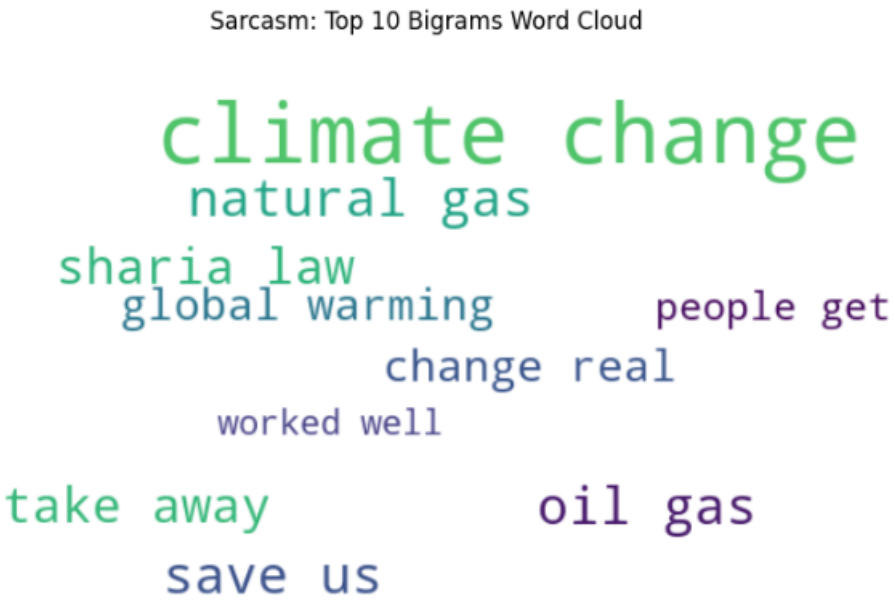

Figure 4: The top 10 sarcastic bigrams show a lot more similarities with the non-sarcastic labels, but contain glaring differences such as "save us".

The following results therefore show that there is contextualization to be captured within the dataset and that implementing a multi-faceted approach to capture all of these features would be beneficial to our sarcasm classification. We will therefore pursue an approach that allows us to capture all of these contextualizations: a combination of TF-IDF, Word2Vec, and GloVe embeddings.

2 Feature Engineering and Algorithm Construction

2.1 Pre-processing

In order to start any NLP classification task, it is customary to go through a set of pre-processing steps in order to reduce a lot of the noise that words with little to no contextual meaning provide. This includes a pipeline of first tokenizing the sentences into a vector of individual words, a process known as tokenization. We then remove words with little to no contextual information, or "stopwords," as commonly referred to, similar to what we did with the construction of the bigrams. Lastly, we apply lemmatization, where words are simplified to their root representations through the use of morphological analysis to accurately reduce the noise in our dataset while minimizing the loss of contextuality, which is something that could occur when introducing stemming into the architecture, a faster but less thorough version of lemmatization.

2.2 Embeddings

As mentioned in the introduction, we wished to capture the combination of short range and long-range contextualization. As covered in class, we know that GloVe is an embedding technique that builds global word co-occurrence statistics, which can better capture overall corpus-level semantics. Using this as our baseline embedding technique can help us build a strong foundation of features in which we can build off of. However, from the EDA in the introduction, we have established the suspicion of local syntactic and semantic relationships, which can be captured expertly through Word2Vec's skip-gram algorithm. Lastly, the differences in the frequency-representations can be captured through TF-IDF vectorization, a sparse-dimension embedding technique that gives relative frequencies in relation to the number of words across the entire corpus.

2.2.1 Resolution of Embedding Conflicts

Based on convention, we have set our GloVe and Word2Vec embedding dimensions to be in $\Re^{100}$ Due to the fact that both Word2Vec and GloVe are represented as dense vectors, and TF-IDF is a sparse vector representation in $\Re^n, n \in$ {words in parameter space Ω}. We must reduce the dimensionality to standardize

the features for model training and evaluation. With the added constraint of not being able to use machine learning models such as SVD for dimensionality reduction, we have opted to go for a feature hashing technique, which hashes every sparse value in the original vector Sinto buckets of $\Re^{100}$ space.

2.3 Combining embeddings and Padding

After all of the pre-processing and feature extraction steps, we made the executive decision to combine all 3 of the word embedding features together into one $\Re^{300}$ vector for both computational efficiency in downstream processing and ease of implementation with a single, unified format. Lastly, we pad the vectors in order for to provide a uniform input size for our deep learning model.

3 Deep-Learning Model

Now that our pre-processing and feature extraction has been completed, we now move forward to the construction of our deep-learning model.

3.1 Implementation of a Convolutional Head

In order to properly capture the ngram features that are being represented in our vetor hyper-representation, it would make sense to convolve the vector to split it into its constituent features and reduce the complexity of the problem before performing a sequential analysis of any sort. This way, meaningful features would be passed on to the next section of our deep-learning model. In order to implement this, we implement 1D convolutions with a 1D max pool on our 1x300 dimensional features. Unlike an entire CNN, we decide to omit the last dense layers, as we do not wish to add a classification head at this point in time. We also omit the vector flattening, as we wish to continue our model after this convolutional layer.

3.2 Input into the Bidirectional LSTM

The new filtered features are then sent into our Bi-directional LSTM architecture, which scans forwards and backwards through our features in the event that one direction might get hyper-fixated on a particular portion of the vector with high contextual power. The reason for this choice was to mitigate the possibility that the LSTM may not capture the semantic relationships for sarcasm due to the nature of it's gating mechanisms. To understand this better, let's assume we can interpret the LSTM on a sentence level. On a forward pass of a normal LSTM, the model might see the word "good," and then hyper-fixate on this positive semantic, neglecting to add meaningful information and lowering the sigmoid probability gate for the second half of the sentence, which may contain negative context such as "just kidding," or "not," which is usually indicative of sarcasm. The Bi-directional approach ensures that this "disregard of information" does not happen when the forward sequential context of the feature space is being considered.

3.3 Classification

Our final layer of our deep learning model consists of a classification Dense Layer, of specified feature input length with a sigmoid activation to give a probabilistic 1 or 0. This entire model is then trained and evaluated on our test set, which was split before model training. The following picture is a visual representation of our architecture:

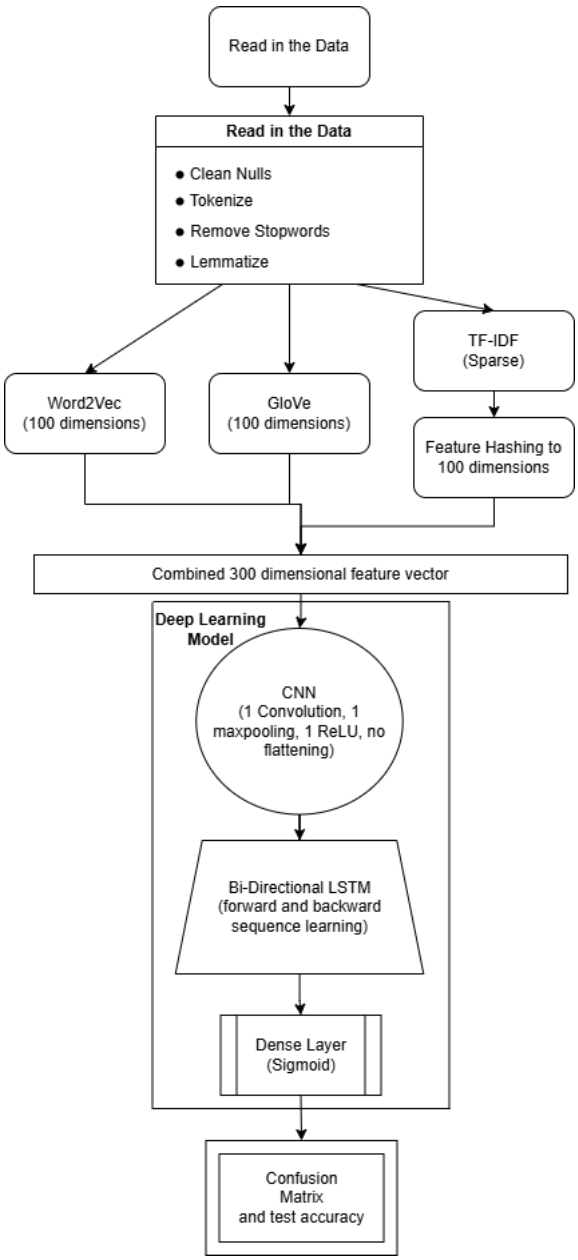

Figure 5: The model depicts the data processing and cleaning pipeline, which is then followed by feature extraction and a custom deep learning model that combines convolutional layers and specialized LSTMs.

4 In-Depth Analysis of Model Training and Hyperparameter Tuning

4.1 Train-Test Split

To begin, we decided to use an 80-20 train-test split in order to train our model, where each data point was randomly sampled. We then used 10% of the 80% for validation.

4.2 Initial Architecture

In order to start simply, we opted to only use 1 convolutional layer, and went on to up complexity as needed, as well as start out with a really low dropout rate. LSTM units on the other hand were treated as a hyper-parameter and was part of the study space. We used Adam as our optimizing function. Initially, we tried grid

search, however we found an optimization tool, Optuna, that we decided to implement instead of a grid-search based approach for the following reasons:

- The program automatically stops training poor convergences, effectively pruning that time from our model training
- The program is easily customizeable, and instead of inputting manual values, we were able to study ranges of hyper-parameter space, which we were able to narrow down by changing the parameters passed in

We therefore created a "study" function, where we scanned across multiple parameter spaces of different magnitudes for the learning rate, the dropout rate, the number of LSTM units, the size of the kernels for our 1D convolutions, and the number of filters in our convolutional layer as inputs. I then changed up the number of convolutions to find the best possible convergence, using this same grid-search like method. What we found was that the best convergence occured with only 1 pooling layer. We therefore decided to scrutinize the single convolution architecture, and continued our search, with ranges of our LSTM unites to be anywhere from 32 to 256, the number of filters to be between 32 and 128, the LR to be between 1e-6 and 1e-4, and a dropout rate of .01 to .41. What we found the be our best hyperparameters within this architecture were as follows:

```
'cnn_filters': 128,
'cnn_kernel_size': 3,
'lstm_units': 224,
'dropout_rate': 0.31000000000000005,
'learning_rate': 0.0011522903935867418
```

We therfore found that kernal filter sizes of 3 (possibly indicative if tri-gram feature representations), with a LSTM size of 224 and a dropout rate of .31 ultimately led to a local minima that best generalized our parameter space. Lastly, the learning rate of .001 helps us identify the best global learning rate for our Adam Optimizer. The epochs found based on these hyperparameters were then agreed upon to be set at 13, where we found best convergence.

5 Results

It was crucial to run our model on the test set only once to obtain an unbiased evaluation of its performance in order to avoid any biases. Repeated evaluations on the test set can lead to inadvertent overfitting, where the model's hyperparameters may be fine-tuned based on test set performance rather than true generalization capability. In addition, our dataset is relatively small, having only 912 entries. This caused some variance in the model's performance, as each individual entry the model was trained on has more weight and influence on the model's parameters, with lesser total entries.

After careful hyperparameter tuning, we have achieved the following unbiased result based on our classification report:

Table 1: Classification Report (Test Accuracy: 0.8743)

Class	Precision	Recall	F1-Score	Support
0	0.81	0.99	0.89	95
1	0.99	0.75	0.85	88
Accuracy	-	-	0.87	183
Macro Avg	0.90	0.87	0.87	183
Weighted Avg	0.89	0.87	0.87	183

The final model achieved a test accuracy of **87.43%**, with F1-scores of 0.89 for non-sarcastic and 0.85 for sarcastic remarks. Overall, our integrated approach demonstrated that combining diverse embedding methods with a hybrid convolutional-recurrent architecture does significantly enhances sarcasm detection, highlighting the value of capturing both local and global contextual information.

6 Conclusions, Lessons, Experiences

In conclusion, our study demonstrated that integrating different embedding techniques with a carefully tuned deep-learning architecture can generalize the complexities of sarcasm detection quite effectively. By combining GloVe, Word2Vec, and TF-IDF embeddings into a 300-dimensional feature vector, we captured

both the global semantics and the local contextual provided by GloVe and Word2Vec respectively, as well as frequency information from TF-ID to classify sarcastic and non-sarcastic cases.

A significant portion of our work involved experimenting with multiple model architectures to strike an optimal balance between complexity and generalization. We initially explored several architectures, including deeper convolutional layers and multiple LSTM stacks, to capture intricate linguistic patterns. However, these more complex architectures tended to overfit the training data, compromising the model's performance on unseen samples. Through a calculated hyperparameter tuning approach using tools like Optuna, we determined that a simpler architecture yielded the best results. This approach maintained predictive power while robust generalizatilizing the data, achieving a test accuracy of **87.43%**.

Throughout this project, we learned that a calculated approach to model design is vital for the construction of a robust generalizer. The iterative process of testing and refining different architectural configurations underscored the importance of balancing model complexity with the risk of overfitting. Moreover, the integration of multiple embedding methods provided richer semantic insights, highlighting that no single technique is sufficient on its own for nuanced tasks like sarcasm detection.

Overall, our experience confirms that thoughtful feature engineering combined with rigorous experimentation can lead to significant improvements in NLP tasks. These lessons pave the way for future research aimed at developing even more adaptive and resilient models for sentiment analysis and beyond.

References

[GNM23] M. Gole, W. P. Nwadiugwu, and A. Miranskyy. On sarcasm detection with openai GPT-based models, 2023. Preprint or unpublished manuscript.

[KSV18] M. Khodak, N. Saunshi, and K. Vodrahalli. A large self-annotated corpus for sarcasm. In *Proceedings of the 11th Language Resources and Evaluation Conference (LREC)*, pages 1–6, 2018.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[Rah23] Mijanur Rahman. Different ways to combine cnn and lstm networks for time series classification tasks. https://medium.com/@mijanr/different-ways-to-combine-cnn-and-lstm-networks-for-time-series-classification-tasks-b03fc37e91b6, 2023. Medium, 13 Jan. 2023.

[RHJ+19] M. O. Rahman, M. S. Hossain, T. S. Junaid, et al. Predicting prices of stock market using gated recurrent units (grus) neural networks. *International Journal of Computer Science and Network Security*, 19(1):213–222, 2019.

[SDT+23] B. Sonare, J. H. Dewan, S. D. Thepade, V. Dadape, T. Gadge, and A. Gavali. Detecting sarcasm in Reddit comments: A comparative analysis. In *2023 4th International Conference for Emerging Technology (INCET)*, pages 1–6, Belgaum, India, 2023.

[YSH+19] Y. Yu, X. Si, C. Hu, et al. A review of recurrent neural networks: LSTM cells and network architectures. *Neural Computation*, 31(7):1235–1270, 2019.

[Ziy25] Ziyuan111. Sarcasm at main. https://huggingface.co/datasets/Ziyuan111/sarcasm/tree/main, 2025. Accessed: 27 Feb. 2025.