Detecting Sarcasm in Sentiment Analysis Using Multi-Head Self-Attention

Henry Pitts
Department of statistics
Virginia Tech
`kinghenry430@vt.edu`

Abstract

Since its inception, social media has steadily gained popularity across the world and as it became more prevalent, people began to look towards it to gain insights on people's opinion. Sarcasm is a common language trend that uses positive words to express negative opinion. This model attempts to correctly identify these occurences by using a multi-head sentiment analysis and the DistilBERT transformer to detect sarcasm among a dataset containing scraped tweets from Twitter. The model uses pre-trained BERT tokenizer to tokenize and encode texts. The training employs Hugging Face Trainer API and AdamW optimization. We use a small subset of the data for experimentation, yet the model shows decent accuracy. Further changes need to be made to increase these metrics however. These changes could include optimizing hyperparameters, addressing potential class imbalance, and increasing the dataset size. There were some limitations in the model due to computation restrictions. This model demonstrates the capabilities of refined transformer architectures in interpreting complex linguistic elements within user-generated text.

github repo:
`https://github.com/henrypitts/Sentiment-Analysis-using-Multi-Head-Self-Attention.git`

1 Introduction

Sentiment analyses allow us to characterize text data by whether they are neutral, negative, or positive. This can give one the chance to monitor opinions on brands, customer reviews, news outlets, etc. The goal is to achieve a better understanding of the customer or user, but unfortunately some sentiment analyses are unable to correctly identify certain human trends in the data, such as sarcasm. Sarcasm is used constantly in day-to-day talking so being able to correctly identify when it happens becomes necessary for an accurate sentiment analysis. Explain the problem and why it is important. Discuss your motivation for pursuing this problem. Give some background if necessary.

2 Related work

Vanilla CNN

This work uses a vanilla convolutional neural network architecture. The convolutional layers are used to produce the feature maps. It features an incongruity, hyperbole, temporality, and dislike detection. The batch normalization layers attempt to improve the speed, performance, and stability by giving a new scale and center to the input data. This study finds a significantly improved F score compared to other models.

Both the proposed and the vanilla CNN model convert text into numerical tokens, however in this model, text is converted to word embeddings, not token IDs. Both these models do rely on supervised learning and use standard loss functions like cross-entropy.

This model uses convolutional layers that learn hierarchical features through kernels. To capture n-gram patters, it then applies local filters. This differs from my model that uses self-attention to understand global relationships between tokens in a sequence. The Vanilla CNN model is pre-trained using FastText while our model is pre-trained on large corpora.

Bootstrapping

The similarities between my proposed model and the bootstrapping model are within the training, evaluation, generalization, and uncertainty estimation. Both models require training data to learn patterns, use the same evaluation metrics, can suffer from overfitting, and can be used to estimate uncertainty.

There are differences in the structure, training methodology, and output. These are based on bootstrapping's resampling method. Our model is based on transformer architecture while bootstrapping uses resampling. The training for our model uses backpropagation and gradient updates, while the bootstrapping method gains its understanding through pattern learning and iteraviely adjusting seeds. This in turn makes outputs different. We will output a single set of predictions, while the bootstrapping method joins results from multiple models.

Attention LSTM

This model utilizes a CNN and a Long Short-Term Memory. Both models use pre-trained words and sequences in texts to find relationships. Other than using deep learning, there are not many similarities.

This model uses a CNN-LSTM architecture in contrast to my multi-head self-attention transformer-based architecture. This model also does not update word embeddings during training.

3 Dataset and Features

The dataset used is sourced from Kaggle and contains around 1.6 million tweets from the year 2009. It was originally planned to use an Amazon customer review dataset, but due to complications in formatting that were deemed too time consuming to fix, I chose to switch to this twitter dataset instead. The data being from 2009 does cause a bit of concern. Social Media, technology, and language has changed since 2009, yet i believe the general sarcasm trends are similar enough.

There wasn't much preprocessing necessary for the data. The .csv file was not in the standard UTF-8 format, so the dataset is read with either ISO-8859-1 or latin1 encoding. Next is to clean the data. URL's, mentions ('@user'), were removed, and the text was changed to lowercase. Next we had to change the 'target' column values from '4' to '1'. This distribution is seen in figure 1. This is a binary column and updated for convenience. Short tweets may lack context, while longer tweets may contain noise and irrelevant information. For this, the dataset is filtered to include tweets between 10 and 50 words. The consistent length may also help reduce variability and limiting the max length can help with computation. We tokenize the text and it is padded to a consistent length using the BERT tokenizer. This allows the model to understand numerical representations of the text, and ensures input sequences are of the same length.

Word clouds are used to visualize frequency of words associated with positive and negative sentiment. Interestingly, in figure 2 we see that 'love' is the most common word associated with positive sentiment. This is the same word used to identify sarcasm in the bootstrapping model. This could cause potential problems in correctly recognizing sarcasm in a sentiment analysis. In the negative word cloud we see that the words 'go', 'now', and 'work' are all have the highest frequencies associate with negative sentiment. It seems like, in general, people tend to give live updates on not wanting to go to work.

This dataset is extremely large and it is not feasible to train using the entire dataset. I decided to take reduce the dataset to a 0.1% random sample. It is then split into training, validation, and test sets. The training set output indicates 649 examples in total after splitting and reducing. The validation

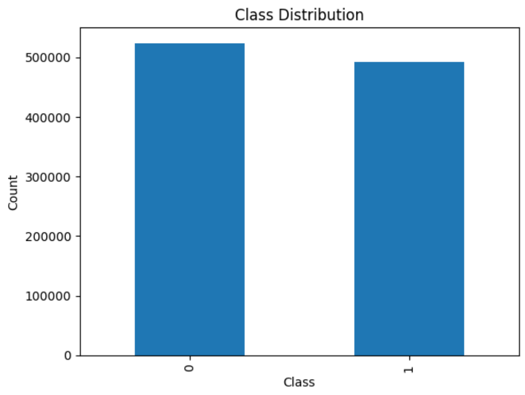


Figure 1: Positive and Negative Distribution of Tweets

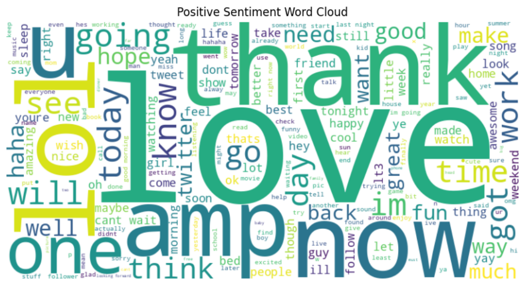


Figure 2: Positive Word Cloud

set includes 163 examples. And there are 204 test examples. These proportions should allow us to evaluate the model's performance accurately.

4 Methods

Architecture:

The proposed architecture is a multi-head self attention model that utilizes DistilBERT. This is a transformer-based neural network. We chose to use the lightweight version of BERT because it has 6 layers instead of 12. The lightweight variant is still able to keep most of BERT's performance, but allows us to compute faster. DistilBert still allows us to keep the multi-head self attention mechanism.

Attention allows us to compute weighted sums on the input sequence based on the importance. Self attention has each token in a sequence attend to all other tokens in the same sequence. This helps establish relationships across the text, by considering the entire input. Multi-head self attention allows us to independently compute attention scores in heads operating in parallel. These heads use a

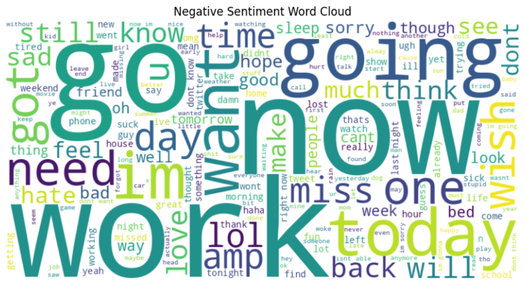


Figure 3: Negative Word Cloud

projection of the original embedding dimensions. For each head, the transformation is performed on the input sequence using matrices.

Attention Calculation:

Attention scores are calculated using the dot product between query and key matrices. The softmax function normalizes to gain weights. The weighted sums are then computed by applying these weights to value vectors.

Q = Query

K = Key

V = Value

$$A = softmax(\frac{QK^T}{\sqrt{D}})V$$

These heads are then concatenated and projected back.

Next non-linearity is introduced into the transformer model by the feed-forward layers. This allows the model to learn more complex patterns. Each feed-forward layer has two linear transformations separated by a non-linear activation function. These are applied independently to each token in the sequence. The distilBERT model uses the the Gaussian Error Linear Unit (GELU) as the activation function, introducing a gradual non-linearity:

$$GELU(x) = 0.5x(1 + error(\frac{x}{\sqrt{2}}))$$

The final part of our model's architecture is the positional embedding. This is needed because transformers do not naturally encode positions, but our model needs to know relative token positions. We are using the pre-trained model 'distilbert-base-uncased' that already has positional embeddings. While going through the forward pass, input tokens become word embeddings and then combined with the positional embeddings. This allows the attention mechanism to determine token positions. As the model processes, positional embeddings are added to keep token order.

Hyperparameters:

Batch Size: 32

Number of samples processed in each training iteration. This value was chosen to increase training efficiency and due to GPU restrictions.

Max Length: 64

Maximum number of tokens in input sequence. Longer sequences require more memory and training time so we chose a lower value.

Learning Rate: 1e-5

This is the step size that updates weights during training. Optimal learning rates are usually found through experimentation. Our learning rate may still need further experimentation.

Epochs: 2

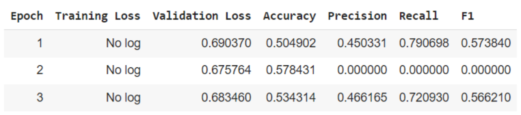
The number of complete passes the model makes over the training data. We wish to avoid overfitting and underfitting. The model needs the right number to be able to learn from patterns in the data, but cannot rely too heavily on it.

Learning Algorithms:

Loss function:

We chose cross-entropy loss because it measures different between the predicted distribution and ground truth classes.

$$L = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(y_i) + (1 - y_i)\log(1 - y_i))$$



Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	No log	0.690370	0.504902	0.450331	0.790698	0.573840
2	No log	0.675764	0.578431	0.000000	0.000000	0.000000
3	No log	0.683460	0.534314	0.466165	0.720930	0.566210

Figure 4: Proposed Model Evaluation

Optimization Algorithm: AdamW

Stands for Adaptive Moment Estimation with Weight Decay. Corrects for weight decay to avoid overfitting. Keeps separate learning rates for each parameter uses gradient estimate averages to update weights.

DisilBERT Fine-Tuning:

DistilBERT is pre-trained on corpora to learn general language patterns. Fine-tuning adapts the pre-trained model to specific tasks on a smaller dataset for the sentiment analysis. This reduces time and computational cost.

Trainer API: 'Trainer from Hugging Face 'transformers' library. This makes training, evaluation, and saving of transformer models easier. The trainer API handles the training loop, evaluation loop, checkpoints and logging.

5 Results

To analyse our results we are using the following metrics:

Precision:

Measures the proportion of correctly predicted positive samples out of all positively predicted samples.

$$Precision = \frac{TruePositives(TP)}{TruePositives(TP) + FalsePositives(FP)}$$

Recall:

Measures the proportion of actual positives measured correctly.

$$Recall = \frac{TruePositives(TP)}{TruePositives(TP) + FalseNegatives(FP)}$$

F1-Score:

Harmonic mean of precision and recall. This balanced measure correctly identifies positive sentiment. Higher F1-Score indicates better model performance.

$$F1 - Score = \frac{2Precision \cdot Recall}{Precision + Recall}$$

Accuracy:

Measures the proportion of correctly predicted samples across all classes.

$$Accuracy = \frac{TruePositives(TP) + TrueNegatives(TN)}{TotalNumberofSamples}$$

Vanilla CNN, Bootstrapping, and Attention LSTM all use the same metrics to measure their models effectiveness. They all also used scraped twitter data to train and test their models. While it won't be certain, this allows us to gain a pretty good comparison of models without having to do so manually. Time and computation restraints have restricted our testing of other models.

Comparing Existing Models:

Metric	Value
Precision	.8193
Recall	.8041
F1-Score	.8116
Accuracy	99.86

(a) Attention LSTM Performance Metrics

Metric	Value
Precision	.63
Recall	.42
F1-Score	.50
Accuracy	N/A

(b) Bootstrapping Performance Metrics

Metric	Value
Precision	.95
Recall	.94
F1-Score	.94
Accuracy	94

(c) Vanilla CNN Performance Metrics

Metric	Value
Precision	.4503
Recall	0.7907
F1-Score	.5738
Accuracy	50.49

(d) Proposed Model Performance Metrics

Table 1: Comparison of Model Performance Metrics

We can compare our evaluation metrics with models in Table 1. Our model is vastly outperformed by the Attention LSTM and Vanilla CNN models. This could be due to the CNN model being better able to capture local patterns within text and the LSTM model being better at at understanding sequential relationships. Our model was able to outperform the Bootstrapping model in all metrics except precision. This means our model is better at overall classification, but making more false positive predictions. This could be due to the different test data, or possibly my model is overfitting. Overall our model has shown decent performance but other models seem to still identify sarcasm more accurately.

6 Conclusion and discussion

Sarcasm is an important language trend that is necessary to identify in order for an accurate sentiment analysis. To solve this I proposed a multi-head self-attention model through DistilBERT transformer. My model gave a decent performance, but was still outperformed by other more widely accredited models. Precision was our model's lowest performing metric. Future work should focus on ensuring class balance, expanding the training data to further improve precision, and optimizing hyperparameters. Given more team members, computational resources, and time, it is possible this model could detect sarcasm better. This model demonstrates the potential of transformer architectures for sentiment analysis tasks, particularly in understanding sarcasm found in Twitter posts.

References

Kumar, Amardeep, et al. "Sarcasm Detection Using Deep Learning With Contextual Features." *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, 2021, pp. 924–930. `https://www.researchgate.net/publication/351571706_Sarcasm_Detection_Using_Deep_Learning_With_Contextual_Features`.

Riloff, Ellen, et al. Salt Lake City, Utah, 2013, pp. 1–11, Sarcasm as Contrast between a Positive Sentiment and Negative Situation.

Poria, Soujanya, Erik Cambria, and Alexander Gelbukh. "Sarcasm Detection Using Deep Learning Techniques." Computers, vol. 12, no. 11, 2023, p. 231. MDPI, https://doi.org/10.3390/computers12110231.