# Finding Sarcasm in Reddit Postings: A Deep Learning Approach

Nick Guo, Ruchir Shah
{nickguo, ruchirfs}@stanford.edu

## Abstract

We use the recently published Self-Annotated Reddit Corpus (SARC) with a recurrent neural network to classify sarcastic statements and evaluate against baseline Bag of N-grams, Naive Bayes, and feed-forward neural network methods. Results show substantial improvements on the baselines with much promise for further research.

# 1. Motivation

### 1.1 Introduction

As speech agents like Siri and Alexa become more powerful, we can increasingly rely on them to understand many day-to-day human inquiries and commands. However, despite an increased ability to understand and process tasks, the biggest obstacle to their success is the ability to engage in conversation. Developing truly conversational speech agents - who can understand all the unique intricacies of the human language - remains one of the largest pending NLP problems of our time.

One critical aspect of this problem is the successful identification of sarcasm. Humans regularly use sarcasm as an important part of day-to-day conversation when venting, arguing, or even engaging in humorous banter with friends. For a speech agent to truly be conversational, detection of sarcasm is a must.

Effective detection of sarcasm in conversation is a two-part process: 1) Detecting sarcasm in words and 2) Detecting sarcasm in tone. In this paper, we delve into the detection of sarcasm in words. Specifically, we work with comments from Reddit, a popular online forum where users can make posts and reply to one another's posts and comments. The input to our model is a reply chain of comments leading up to a final comment which our recurrent neural network will predict as sarcastic or not sarcastic. The dataset we work with is unique in that it is the first of its kind with enormous amounts of sarcastic statements that are annotated by the original authors—we elaborate more on this in later sections.

### 1.2 Related Work

Over the past few years, a large number of researchers have tackled different aspects of the sarcasm problem (Joshi 2016). The vast majority of studies implemented so far have used Twitter data as a medium, with distant supervision available in the form of tweets with the hashtag "#sarcasm".

Many of the initial studies focused on using traditional, regression based approaches to sarcasm classification using solely the text of the comment to be assessed. Later studies (Wallace 2014) have expanded on this approach with a realization that in addition to the content of specific sentences, context is ultimately quite important in sarcasm classification. These studies have expanded feature sets to include detailed metrics of the words in the sentence as well as features of the author's previous use of sarcasm and previous interactions between the author and the recipient of the tweet (Bamman 2015). Some of these studies have used logistic regression, although comparisons have generally resulted in SVM models being the most successful (Peng 2015).

In more recent years, deep learning has been utilized to solve the sarcasm problem with great success. Approaches include a focus on the use of emojis in tweets (Felbo 2017) as well as an examination of user embeddings (Amir 2016).

# 2. Dataset & Features

### 2.1 Corpus

Earlier this year, an enormous corpus of textual sarcasm was published containing 1.3 million sarcastic statements made by Reddit users (Khodak 2017). In particular, this data set takes advantage of a Reddit norm to include the symbol "/s" after a sarcastic statement. This data is especially valuable because the SARC authors went through great lengths to ensure that they only included comments (both sarcastic and non-sarcastic) by accounts that adhered to this norm of labeling sarcastic data. In order for a user's comments to be included in the dataset, he or she must have used "/s" in at least once, indicating that they are aware of the norm. As a result, this is the cleanest and largest data set available for public use in a study on sarcasm.

In addition to containing the words of individual posts that are sarcastic or not sarcastic, the corpus also includes all previous and subsequent posts within the thread on Reddit. As a result, we have a tremendous amount of contextual information around the sarcastic post: what caused it and what followed it.

In our project, we ran our algorithms with only the "r/politics" subreddit data. This is a subset of Reddit

known for particularly sarcastic commentary, and we wanted to use a smaller dataset to allow for faster development through the project. In addition, we believe a topic focused dataset will make it easier to train contextual features. In future work, researchers with greater computing resources may want to replicate our study on the full data set.

We worked with two datasets that are both composed of comments from the "r/politics" subreddit. One of them is "unbalanced" because sarcasm is quite rare (about 3.1%), and the other is "balanced" because it contains precisely 50% sarcastic and 50% not sarcastic statements. For reference, the size of the training sets were 305k for "unbalanced" and 13k for "balanced." The "unbalanced" dataset is a nonbiased sampling of all the comments across all posts, so it should be quite representative of the general activity on the politics subreddit. We then split each dataset into 80% for training and 20% for testing. We took 10% of the training set and set it aside as a dev set. We found that training a model on the "balanced" dataset and then applying that model to classify samples from the "unbalanced" dataset did not generalize well as it had high bias towards predicting that comments are sarcastic—our methodology for working with sparse occurrences of sarcasm is elaborated upon in section 4.

**2.2 Features**

Our literature review of linguistic papers covering sarcasm identified five major areas that define what is required to know that something is sarcastic: the Speaker, the Listener, the Context, the Utterance, Literal Words, and Intended Words (Joshi 2016). Our dataset provides little information on the author and recipient of comments since Reddit users are anonymous, and we only have text data so we are unable to hear voice inflections for determining utterance.

As a result, our focus was primarily on the user's Literal Words, Intended Words, and the Context around the words.

We created the following word-based features:

1. # of words in comment
2. Absolute count of 25 Parts-of-Speech tags
3. Intensifiers Binary (whether the comment contained one of 50 intensifying words)
4. Sentiment analysis of comment (using NLTK's sentiment library)

After initial testing with these features against our baseline metrics, we realized that a substantial amount of classification value came from the words themselves.

We then implemented a Word2Vec model using gensim to vectorize all the words for our comments. We trained Word2Vec across our entire training corpus, and then we vectorized the words in each comments using our model. A sample of an actual sarcastic comment that goes through this vectorization is included:

1) "You're right, this election is totally different! /s"

2) [youre, right, election, totally, different]
*(with stemming): [your, right, elect, total, differ]*

3) [-0.40301099, -0.51847959,..., 0.07148877]

To address context, we also ran our Word2Vec model across all previous comments in the thread (prior to the comment we were classifying). Through experimentation, we found that we obtained better results by training Word2Vec on the comments from the training set as opposed to text from Wikipedia and news articles which is standard practice. This is in line with our expectations since domain knowledge is crucial in identifying context.

# 3. Methods

We used two baseline techniques: Bag of Words and Naive Bayes. Our goal using these methods was to validate previous results obtained by the authors who initially released the Reddit data set.

**3.1 Baseline Technique: Bag of Words**

We tried two different bag of words measures: unigram and bigram. We implemented both algorithms ourselves.

With the Unigram Bag-of-Words technique, we went through the entire corpus and counted how many times each word was used in the sarcastic corpus and how many times each word was used in the non-sarcastic corpus. Then, for test examples, we used the word probabilities (ignoring order) to see if the example matched up more with a sarcastic or non-sarcastic distribution.

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

Similarly, with the Bigram Bag-of-Words technique, we went through the entire corpus and counted how many times each bigram (group of two sequential words) was used in both the sarcastic and non-sarcastic corpuses. Then, we used the bigram counts to classify the comment.

**3.2 Baseline Technique: Naive Bayes**

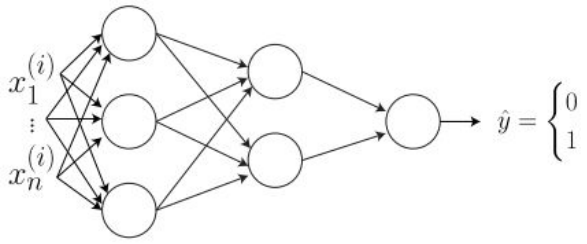
Naive Bayes is similarly a very simple and straightforward algorithm; we implemented it because it is known for achieving high level of successes in NLP circles.

Naive Bayes is a simple probabilistic model that assumes all features fed into a model are independent. Within the realm of natural language processing, each one of these features is a word or bigram in the model. Using Bayes' Law, the model generativity multiples the probabilities together to make a classification.

$$p(y=1|x) = \frac{p(x|y=1)p(y=1)}{p(x)}$$
$$= \frac{\left(\prod_{i=1}^{n} p(x_i|y=1)\right)p(y=1)}{\left(\prod_{i=1}^{n} p(x_i|y=1)\right)p(y=1) + \left(\prod_{i=1}^{n} p(x_i|y=0)\right)p(y=0)},$$

### 3.3 Feed-Forward Neural Network

Our first deep learning approach used a traditional neural network on our data.




A feed-forward, or traditional "vanilla" neural network, uses a series of layers and probabilistic functions to make a more complex, interaction-based prediction versus a traditional regression function. The model is trained with a series of inputs that are then transferred to one or more hidden layers containing probabilistic functions. At each stage, the model is trained using gradient descent to optimize parameters that linearize the results of each layer to the next layer. We use sigmoid as the activation function in the output layer.

### 3.4 Recurrent Neural Network (LSTM)

Initial results from our feed-forward neural network were lackluster as we realized that it wasn't enough to look at words purely as a bag-of-words. Order likely matters in sarcasm. So, we switched approaches to a recurrent neural network, specifically the LSTM (Long Short-Term Memory) model.

An LSTM neural network is unique in that, when processing data points, it factors in not only the current data point but also all previous data points. As a result, it is able to factor in sequence (such as sequence of words in a sentence) when making classifications. In addition, an LSTM is able to forget elements from its past. During the gradient descent process, this approach allows the network to more effectively learn sequential data without overfitting.

# 4. Discussion

### 4.1 Baseline Models

When implementing our baseline models, we used mostly straightforward implementations of Bag of Words and Naive Bayes. To ensure we could handle unseen words with nonzero probabilities, we used Laplace smoothing.

For initial use of words in these baseline methods, we did a significant amount of text processing. This includes a lot of tokenizing; we shortened words with 2 or more repeated characters to the same token, made all words lowercase, and we also used the NLTK library to stem similar meaning words to the same pseudo-word. We also removed popular stop words such as "the" and "a" found in the NLTK stopwords corpus as other researches have done since the general consensus is that stop words do not carry much value in determining sarcasm.

### 4.2 Feed-Forward Neural Network

For our initial, feed-forward neural network, we incorporated as many features as possible into our implementation with the idea in mind that the neural network would figure out the most prominent ones and assign weights accordingly.

In addition to the pure word-based features (ie. length of comment), we used Word2Vec on all words in a given comment. When using Word2Vec, we encoded every single word into a vector of dimension 200. After generating Word2Vec vectors for each word, we averaged them together for a defacto Comment2Vec that captured the overall embedding of the entire comment.

Similarly, we also used Word2Vec on all comments that preceded our current comment, averaging all the vectors together for a PreviousComment2Vec.

Then, for our final features, we incorporated our Comment2Vec, PreviousComment2Vec, and their difference for a rough estimate of how different the comment is relative to its context. This vector is then fed directly into the feed-forward neural network.

For the structure of our neural net, we fed all features into 2 hidden layers, the first with 300 nodes and the second with 4 nodes. We choose 300 neurons in the first layer because our feature vectors are just over 400 in size. We then added a second hidden layer to account for potentially different kinds of sarcasm (we saw better results with the second hidden than without).

As this is a classification problem, we use the sigmoid function as activation for the output layer. We trained with a batch size of 100 over 10 epochs. We chose 10 epochs as that is when we saw the model start to converge during training. Lastly, for our Loss Function we utilized standard binary cross entropy loss.

### 4.3 LSTM

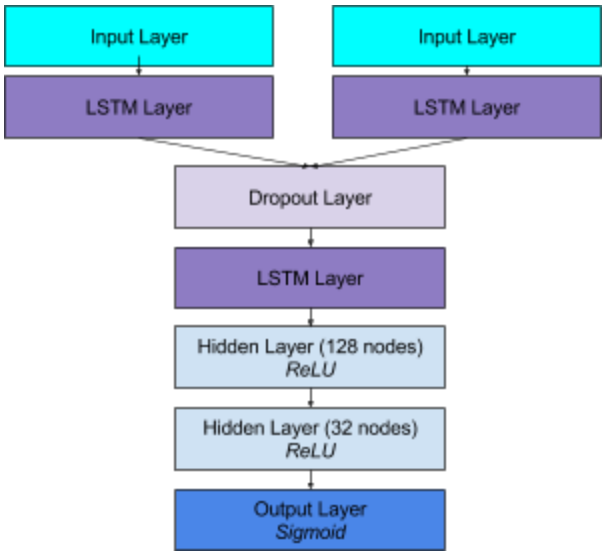
Following feedback from classmates and the TAs, we changed our model to include an LSTM layer that took into account sequencing. As a part of this process, we greatly simplified our neural network architecture.

Further analysis of our feed-forward network indicated that almost all of the feature value came from the Word2Vec components - of both the current comment and the previous comments. So, in our LSTM model, we removed all non Word2Vec features.

Secondly, to take advantage of the LSTM's ability to look at sequence, we restructured our Word2Vec parameters. First, we encoded the vectors in 100 dimensional embeddings instead of 200 for both practical implementation purposes and because we are no longer embedding entire sentences.

Instead of using a Comment2Vec approach, we encoded each Word2Vec vector into a matrix that includes all words in the sentence. To ensure that each matrix (for every comment) had the same size, we set up a maximum length for the comment of 100 words. Then, if the comment had less than 100 words, we would add the zero-vector for all missing words. If the comment had more than 100 words, it would be truncated which is in line with established practices. We came up with the 100 words after physically examining the lengths of many comments in our dataset.

To include context, we created a Word2Vec matrix for all previous comments in addition to the current comment. We then fed all these features together into the LSTM layer.




For the structure of our neural net, we fed the current and previous comments' Word2Vec matrices into a LSTM layer. Then, after their representations are learned, we passed the primary comment's representation along with the difference between its representation and the previous comment's representation into another LSTM layer. Finally, the resulting representation passes through some dense ReLu layers culminating in a sigmoid function activation layer as before. We also added dropout between the LSTM layers which allows the model to generalize better. We parametrized the amount of dropout so that we could tweak it.

We trained with a batch size of 1000 over 5 epochs. We increased the batch size due to time and computing constraints; RNNs take much longer to train than conventional neural nets.

Lastly, we made a change to the loss function for our LSTM model by incorporating a weight that incentivizes the model to predict sarcastic more frequently.

**4.4 Metrics**

Across all models, we looked at four distinct metrics: Error, Precision, Recall, and the F1 score.

We need to look at metrics beyond the simple error because it isn't as useful for determining the effectiveness of a model since the dataset is unbalanced and always predicting not sarcastic yields only 3% error. Thus, we evaluate precision to see how accurate the model was at predicting sarcasm. This measures the amount of false positives that our models make, and we aim to achieve a high precision value which ensures that when our model predict sarcastic, it's usually correct. We also evaluate recall to see what percentage of all sarcastic comments the model was successfully able to identify. The basic feed-forward neural networks suffers from low recall since it isn't able to generalize as well as the LSTM.

Lastly, the F1 score is a metric that essentially indicates an average of both precision and recall across our model.

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

**4.5 Results**

Generally, deep learning is able to classify sarcasm with significantly higher precision than the other models are able to when evaluated on the test set. We attribute the stronger performance of the neural net to its ability to learn from the meaning of the comment since the other models only look at words without context.

**Baseline Results: Train Data**

| | Error | Recall | Precision | F1 Score |
|---|---|---|---|---|
| Unigrams | 0.111 | 0.269 | 0.087 | 0.131 |
| Bigrams | 0.031 | 0.000 | 0.000 | 0.000 |
| Naive Bayes | 0.056 | 0.207 | 0.168 | 0.185 |
| Feed-Forward NN | 0.028 | 0.189 | 0.696 | 0.297 |

**Baseline Results: Test Data**

| | Error | Recall | Precision | F1 Score |
|---|---|---|---|---|
| Unigrams | 0.098 | 0.179 | 0.068 | 0.098 |
| Bigrams | 0.030 | 0.000 | 0.000 | 0.000 |
| Naive Bayes | 0.052 | 0.074 | 0.084 | 0.079 |
| Feed-Forward NN | 0.032 | 0.087 | 0.356 | 0.140 |

**LSTM Results over Dev**

| Drop-Out | Error | Recall | Precision | F1 Score |
|---|---|---|---|---|
| **0.0** | 0.049 | 0.224 | 0.221 | 0.223 |
| **0.1** | 0.054 | 0.237 | 0.197 | 0.215 |
| **0.2** | 0.049 | 0.212 | 0.212 | 0.212 |
| **0.3** | 0.052 | 0.242 | 0.207 | 0.223 |
| **0.4** | 0.050 | 0.219 | 0.209 | 0.214 |
| **0.5** | 0.045 | 0.207 | 0.237 | 0.221 |

**LSTM Results over Test**

| Drop-Out | Error | Recall | Precision | F1 Score |
|---|---|---|---|---|
| **0.0** | 0.075 | 0.266 | 0.130 | 0.175 |
| **0.1** | 0.084 | 0.301 | 0.123 | 0.175 |
| **0.2** | 0.078 | 0.270 | 0.126 | 0.172 |
| **0.3** | 0.084 | 0.288 | 0.120 | 0.170 |
| **0.4** | 0.079 | 0.279 | 0.127 | 0.175 |
| **0.5** | 0.069 | 0.255 | 0.140 | 0.181 |

Due to memory constraints, we evaluated the LSTM on a dev set rather than over the whole training data. We also see that drop out assists the LSTM model in being able to better generalize once evaluating the test data.

Both our neural net models do far better than non deep learning approaches, likely because they implement context effectively (ie. looking at previous comments).

On top of that, LSTM vastly outperforms our simpler feed-forward network as it better encodes the sequence information. LSTM achieves better recall at the expense of precision as it does not rely solely on the entirety of the context as the feed-forward network does. We expect to achieve better results given a more representative embedding scheme than Word2Vec.

# 5. Conclusion

### 5.1 Final Thoughts

In conclusion, we have identified the following key learnings from this experiment:

1. Deep learning is a far superior approach to conventional NLP approaches such as Naive Bayes and the Bag of Words
2. Contextual data is critical to classifying sarcasm; simple word analysis is not enough
3. Word sequence is statistically significant when classifying sarcasm; this is evidenced in how much more effective our LSTM was versus our vanilla neural net model

### 5.2 Future Work

In the future, we plan to greatly expand this model and improve our results.

First, we hope to run our model across the entire Reddit data set, not just the r/politics subset. Currently, the evaluation on r/politics (raw data of ~750 MB) takes about 28 GB of RAM, so a vastly parallelized approach will be needed in order to train over the entire dataset (roughly 250 GB of raw data). We used a desktop computer with 32 GB of RAM and an Intel i7-6700k, nVIDIA GTX 1070. Enabling CUDA acceleration in Tensorflow granted 10x speed ups in training times and made much of our work more feasible. It will take significantly more computing power to run experiments over the entire Reddit dataset.

Second, we hope to create a more effective model to represent context and meaning. Analysis of our data has indicated that context is extremely important for sarcasm prediction; our next question is exactly how much context and in what form would be most useful? We can use the Reddit data to examine these questions. Would the previous comment provide the most value or would the first comment in a series provide more? Also, an external knowledge graph would provide much needed context for information that is not available in the comments directly.

Lastly, our ultimate goal with this project was to explore the creation of a generative model focused on sarcasm. While this is a very challenging topic no previous papers have really tackled, we believe an exploration of more cutting-edge deep learning techniques could be utilized towards creation of sarcasm, not just classification.

# 6. Appendix

### 6.1 Contributions

Ruchir was responsible for the literature review, report writing, and initial structuring of techniques. Nick did initial data acquisition, processing, implementation of the baseline algorithms, and writing the technical aspects of the report. Both team members were involved in architecting the neural networks and fine-tuning the model to achieve higher precision. The development of the LSTM model took the longest time, and many hours were spent by both Nick and Ruchir in debugging and training.

### 6.2 References


Aditya Joshi, Pushpak Bhattacharya, Mark Carman. 2016. Automatic Sarcasm Detection: A Survey

Bjarke Felbo, Alan Mislove, Anders Søgaard, Iyad Rahwan, Sune Lehmann. 2017. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm.

Byron C. Wallace, Do Kook Choe, Laura Kertz and Eugene Charniak. 2014. Humans Require Context to Infer Ironic Intent (so Computers Probably do, too)

Chun-Che Peng. Mohammad Lakis. Jan Wei Pan. 2015. Detecting Sarcasm in Text: An Obvious Solution to a Trivial Problem.

David Bamman and Noah A. Smith. 2015. Contextualized Sarcasm Detection on Twitter

Mikhail Khodak, Nikunj Saunshi, Kiran Vodrahalli. 2017. A Large Self-Annotated Corpus for Sarcasm

Silvio Amir, Byron C. Wallace, Hao Lyu, Paula Carvalho, Mario J. Silva. 2016. Modelling Context with User Embeddings for Sarcasm Detection in Social Media