Automated Sarcasm Detection

Jonathan Waring
Department of Computer Science
University of Georgia
Athens, GA 30605
jwaring8@uga.edu

Jonathan Hayne
Department of Statistics
University of Georgia
Athens, GA 30605
jh80876@uga.edu

Abstract

Sarcasm is a nuanced form of language in which the author is attempting to convey the opposite of what is actually stated. Given its inherent ambiguity, sarcasm detection has been a challenging task for humans and computers alike. We attempt to develop a machine learning strategy using natural language processing (NLP) techniques in order to identify sarcasm in Reddit comments. Experimental results on the SARC dataset shows that by combining lexical and word embedding features, our best performing model achieves a testing accuracy of 61.24% and a F1 score of 0.605 .

1 Introduction

The rise of multiple social media and discussion board platforms has given individuals more opportunities to express their opinions online. Along with this rise in opinionated content, there has also been an increase in the amount of sarcasm used in these platforms [5]. Sarcasm is typically characterized as an ironic or satirical statement meant to insult or mock someone or something, but it has many other uses as well. Detecting sarcasm is a challenging task and is usually accomplished in face-to-face communication through vocal tone and facial cues that communicate the speaker's intent. Recognizing sarcasm in plain written text is much harder given that we are lacking these non-verbal cues. Past research in automated sarcasm detection has shown that detecting sarcasm is difficult because it occurs infrequently (compared to non-sarcastic comments) and is difficult for even humans to discern [25].

Recognizing sarcasm is important for natural language processing to avoid misinterpreting sarcastic statements as literal. For example, the Secret Service posted a request for an automated software that detected whether threats on Twitter were serious or not [1]. Many companies also rely on tools such as HootSuite [2] to analyze social media and product review data, which is capable of performing tasks such as content management, sentiment analysis, and extraction of messages that customer service representatives may want to respond to. However, these current systems lack the sophistication to decipher the more nuanced forms of language, such as sarcasm. This imposes an extra burden on companies' social media teams that are already flooded with thousands to millions of messages a day.

Our goal in this study is to tackle the problem of automated sarcasm detection on Reddit. Previous research on automated sarcasm detection has been summarized [10], and it can be seen that related works in this area typically focus on treating sarcasm as a linguistic feature and thus focuses on word embedding and lexical/pragmatic feature engineering of the text [24, 23, 15, 7]. However, there is a recent study that takes a behavioral modeling approach in which they focus on user behavior to detect sarcasm [21]. We choose to adopt the word embedding and lexical analysis approach.

[1] `https://www.fbo.gov/?s=opportunity&mode=form&id=8aaf9a50dd4558899b0df22abc31d30e&tab=core&_cview=0`

[2] `https://hootsuite.com/`

2 Data

2.1 SARC

In this work, we focus on 2-way classification (sarcastic vs non-sarcastic) between quote-response (Q-R) pairs using the Self-Annotated Reddit Corpus (SARC) [11]. The SARC dataset is a large corpus for sarcasm research and for training and evaluating systems for sarcasm detection that was put together by the Computer Science Department at Princeton University. The SARC dataset contains over 1.3 million quote-response pairs, but the full dataset is very imbalanced (approximately 10x more non-sarcastic comments).

2.2 Reddit as a data source

All the statements come from Reddit [3], a social media site in which users communicate by commenting on *submissions*, which are titled posts consisting of embedded media, external links, and/or text, that are posted on topic-specific forums known as *subreddits*. Users comment on these submissions and on other comments, which results in a tree-like conversation structure such that each comment has a parent comment (which acts as the "quote" for our Q-R pair). The corpus is composed of a subset of all comments on Reddit from January 2009-April 2017. For each comment, there is a sarcasm label, an author, the subreddit it appeared in, the comment score as voted on by users, and the date of the comment.

2.3 Self-Annotation

Each statement in the dataset is also self-annotated — sarcasm is labeled by the author, not an independent annotator. This is inherent due to the fact that Reddit users have adopted a common method for sarcasm annotation consisting of adding the marker "/s" to the end of sarcastic statements. As with Twitter hashtags, using these markers as indicators of sarcasm is noisy [2]. This is because many users do not make use of the marker, do not know about it, or only use it where sarcastic intent is not otherwise obvious. The creators of the SARC dataset estimated the nosiness of the data by manually checking a random subset of 500 tagged sarcastic comments and 500 tagged as non-sarcastic. A comment was determined to be false positive if "/s" tag was not an annotation but part of the sentence and a false negative if the comment author was clearly being sarcastic to the human rater. This method produced a false positive rate of 1.0% and a false negative rate of 2.0%.

2.4 Data points

Our goal was to create a general automated sarcasm detector from text alone, so we choose to ignore the metadata surrounding each Q-R pair. However, we do acknowledge that this metadata could be potentially useful. We also choose to use only a subset of the SARC dataset in order to give us balanced labels. We end up with 255,172 training examples and 63,979 testing examples from over 4,000 different subreddits. We also construct our dataset in such way that each quote (parent comment) has at least one sarcastic and one non-sarcastic response in the training and testing sets. Table 1 shows example Q-R pairs for both label types.

Table 1: Example Q-R pair from SARC dataset

Sarcastic	**Non-Sarcastic**
Quote AMA Request: Anyone involved or has been involved in Organized Crime	*Quote* Well this sucks: Apple confirms iPhone 6 NFC chip is restricted to Apple Pay
Response I think there has already been an AMA for someone working in Goldman Sachs	*Response* Not really surprising or anything, knowing apple.

[3] `https://reddit.com/`

3 Feature extraction

Since sarcasm detection is a classification problem, we created a label vector with sarcastic responses assigned a 1 and non sarcastic responses assigned a 0. Our feature vectors consisted of two main components, a word embedding representing a quote and its response, and lexical features of each quote-response pair. We utilized two word association databases in the construction of these lexical features, the AFINN sentiment wordlist and the NRC Word-Emotion Lexicon. [18, 16]. For the word embeddings we used Gensim's Word2Vec package [22]. Prior to the construction of our feature vectors we preprocessed the text by tokenizing and removing stopwords.

3.1 Preprocessing

Some degree of preprocessing is necessary in almost every form of natural language analysis. The true meaning of a sentence needs to be broken down into individual clumps of meaning for an algorithm to generate any sort of decent inference on that sentence. Our preprocessing consisted of two parts, stopword removal and tokenization, and was implemented identically across the entire corpus. Stemming, which is the act of stripping off prefixes, suffixes, and other extraneous word add-ons from a base word, is a common practice employed in most natural language processing that we decided against. The way in which a word is said can influence a sentence in a significant way, especially in sarcastic text. Stemming throws away this valuable information. We elected to remove any word's belonging to NLTK's stopword corpus (words such as that, a, the, etc.) since they contain no valuable meaning. Additionally, we tokenized our words using NLTK's TweetTokenizer. Tokenizing is the process of breaking whole sentences down into individual word tokens. NLTK provides their users two main Tokenizers, a Word tokenizer and a Tweet toeknizer, the latter being preferred in social media text analysis due to its flexibility; it recognizes hashtags, punctuation, emoticons, and username handles [3].

3.2 Word2Vec

We created our word embeddings using Gensim's Word2Vec. Word2Vec takes a list of tokenized sentences and assigns every word a vector in a subspace that clusters words often appearing in context near each other. For example, apple would be nearby fruit and microsoft, but microsoft and fruit would not be very close. It creates this complex word embedding by combining two neural nets, a Continuous Bag of Words (CBOW) model and a Skip Gram model. These models each predict the probability of a word given a context and a context given a word respectively. The weights of these neural nets are then used to generate the subspace. Word2Vec requires several parameters; among the more important ones are the context window, dimension, and n-gramming. The context window specifies how many words surrounding a word to use as context. We used the relatively high value of 7 because sarcasm has a very complicated structure and we wanted to avoid removing too much information. Standard Word2Vec dimension sizes range from 100-300, with larger dimension sizes typically being used for larger vocabularies. Larger dimension sizes also tend to create better embeddings, so we chose a dimension size of 300 [22]. Once obtaining the word embeddings for our text corpus, we then create sentence embeddings by summing up the word embedding vectors for each word in a sentence, and averaging over their values. In future works, it may be more useful to use Word2Vec's sister algorithm, Doc2Vec [14], for creating these sentence embeddings.

3.3 AFINN

The AFINN [18] sentiment wordlist contains a list of 2,477 words and their corresponding valence. Valence is scored on a -5 to 5 scale and measures the amount of positive or negative sentiment in a word. These valence values were generated by aggregating other sentiment scores on twitter data. Table 2 shows some example valence values for three different words in the AFINN wordlist.

3.4 NRC Word-Emotion Association

The NRC Word-Emotion Lexicon [16] contains a list of 14,182 words and their associations with eight basic emotions: anger, fear, anticipation, trust, surprise, sadness, joy, and disgust. These words associations were generated using Amazon's Mechanical Turk crowdsourcing platform. Table 3 shows some example emotion association values from the NRC database.

Table 2: Example valence values in AFINN wordlist

Word	**Valence**
Abandon	-2
Cutting	-1
Responsible	2

Table 3: Example emotion association values in NRC database

Word	Anger	Fear	Anticipation	Trust	Surprise	Sadness	Joy	Disgust
Countdown	0	0.872759	1.214373	0	0	0	0.082285	0
Apparent	0.351106	0.342516	0	0	0	0	0.202171	0.020875

3.5 Final feature vectors

Note that all of the above feature construction methods only apply to individual words and phrases. To get features for a sentence we simply average the lexical features of each word token — provided that word has any associated lexical features. Then the word embedding vectors for each word token in the sentence are averaged. The feature vector for a quote response pair is structed as follows (Quote is abbreviated with a Q and response with an R):

[Q embedding, R embedding, Q emotions, Q valence, R emotions, R valence]

Since our analysis relies heavily on neural nets of varying topology and architecture we must standardize our features before passing them through the nets. To do this we used Scikit-learn's StandardScaler [20].

4 Machine learning models and results

Using the final feature vectors described in the previous section, we then run several different machine learning classifiers and evaluate their performance in terms of accuracy, precision, and recall. The first model we try is Gaussian Naive Bayes. Even though its independence assumptions are often far-fetched, Naive Bayes is known to work well in natural language processing tasks [12]. We then proceed to test our data with Logistic Regression, in which the concept of an odds ratios is used to calculate the probability of a certain outcome (sarcasm or not) of the dependent variable given its independent variables (our feature vectors). We performed a grid search to optimize Logistic Regression's regularization hyperparamter. Following that, we test our data with the Extreme Gradient Boosting (XGBoost) algorithm [6]. XGBoost is an algorithm that has been dominating applied machine learning and Kaggle [4] competitions [4]. Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Gradient boosting is a specific approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. XGBoost is incredibly flexible and thus has many different hyperparameters to tune. Consequently, we performed 20 iterations of a randomized search to optimize 6 different hyperparameters: maximum tree depth of base learners, minimum loss reduction required to make a further partition on a leaf node of the tree, minimum sum of instance weight needed in a child, subsample ratio of the training instance, subsample ratio of columns when constructing each tree, and L1 regularization term on weights.

Lastly, we train our data with several different neural network models using TensorFlow [1]. All of our neural network architectures use the following: ReLU activation functions [17] across all hidden layers, gradient clipping using global norm [19] to account for vanishing gradient problem, minibatch gradient descent, Adam optimization [13], and a convergence threshold of 1.0×10^{-5}. We also used two methods of regularization to account for overfitting: dropout [8] and L2 loss penalties. Each neural network architecture was trained 4 different ways: no regularization, 20% dropout regularization, L2 loss regularization, and both dropout and L2 loss regularization.

[4]`https://kaggle.com/`

Table 4 shows the optimized training and testing accuracies, along with testing precision, recall, and F1 scores for the non-neural network models. Table 5 shows the training and testing accuracies for the best performing neural network architecture on its 4 different iterations described above. Note that *L indicates how many hidden layers and *U indicates total number of hidden units across all layers in Table 3.

Table 4: Non-neural network results

Machine Learning Model	**Training Accuracy**	**Testing Accuracy**	**Precision**	**Recall**	**F1 Score**
Gaussian Naive Bayes	54.18%	54.21%	53.31%	70.58%	0.607
Logistic Regression	59.68%	59.32%	59.56%	58.97%	0.593
XGBoost	71.73%	61.24%	61.89%	59.22%	0.605

Table 5: Neural network results

Neural Network Architecture	**Training Accuracy**	**Testing Accuracy**
1L-400U (No regularization)	62.77%	59.65%
2L-600U (20% dropout)	65.04%	58.95%
2L-700U (20% dropout)	70.66%	57.70%
3L-750U (20% dropout)	67.98%	57.60%
3L-1875U (20% dropout)	74.77%	56.55%

5 Discussion

In our study, the best testing accuracy for sarcasm detection on Reddit came from XGBoost with a 61.24% testing accuracy. We also observe in this model that 61.89% of sarcastic classified comments were actually sarcastic and we correctly classified 59.22% of the sarcastic comments. While neural networks and deep learning typically perform a lot better in most tasks, we had lots of problems with our neural networks overfitting, even with dropout. We also observed that while adding L2 penalties to our loss function helped reduce overfitting, we also got our lowest training/testing accuracies with the added penalties. This is a sacrifice that we are not willing to make. The best testing accuracy from a neural network was from the 1 hidden layer network (59.65%) with no regularization. This is barely better than the optimized Logistic Regression model at 59.32%. To get an idea of how problematic overfitting really was, one of our neural network models achieved a training accuracy as high as 96.64%, but resulted in testing accuracy of only 54.50%. Adding regularization to this model only brought up testing accuracy to 56.55% while dropping training accuracy to 74.77%.

A similar study in sarcasm detection on Twitter [7] using lexical and pragmatic features, along with feature selection techniques, achieve testing accuracies around 70% on various models. Even though it is known that context is often needed for sarcasm detection [25, 2], our word-embedding features form Word2Vec, which is supposedly better at capturing contextual information surrounding words, was not helpful for us in beating these accuracies. Other sarcasm detection studies that focus on context tend to lean towards using user behavior as context clues [2, 21] in achieving better accuracy. It would appear that the thrown out metadata surroudning the comments from the SARC dataset would potentially be helpful in improving our sarcasm detection models.

This same study from González-Ibánez, et al. [7] showed that human accuracy in detecting sarcasm in plain text on Twitter was only around 62.59%. Given that our best testing accuracy was 61.24%, we are close to the human baseline in detecting sarcasm from plain text alone. However, the authors of the SARC dataset [11] find that across all subreddits, the average human accuracy in detecting sarcasm on Reddit is approximately 81.6%. It is clear that the Q-R pairs from Reddit allow humans to have more context clues in detecting sarcasm, whereas Twitter users often lack any sort of parent comment information. It appears that we were unsuccessful in incorporating that contextual information for our models to find it of any use in detecting sarcasm.

In the future, it would appear that in order to address these problems in the future, one should consider doing the following. First, one should consider metadata surrounding each sarcastic comment, as it likely contains a lot of useful behavioral and contextual clues. However, this metadata can only go so far, as we are likely missing a lot of other information. For example, a lot of posts on Reddit contain images or links out to other websites, and the SARC dataset does not capture that data. Not only that, but the choppy text and noiseness of the labels in this dataset makes the task all the more challenging. Secondly, other studies that make use of more lexical features [23, 9] tend to be more successful, so one should consider expanding upon the Afinn and NRC Word-Emotion Association features that we used. Lastly, it is intuitive that detecting sarcasm across multiple different subreddits is a very challenging task given that different subreddits have different contextual information that may be useful. Therefore, it may be more helpful to work on detecting sarcasm within subreddits before trying to build a general sarcasm detector. However, it would appear that detecting sarcasm will always be a challenging task.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] David Bamman and Noah A Smith. Contextualized sarcasm detection on twitter. In *ICWSM*, pages 574–577, 2015.

[3] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[4] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[5] Daantje Derks, Arjan ER Bos, and Jasper Von Grumbkow. Emoticons and online message interpretation. *Social Science Computer Review*, 26(3):379–388, 2008.

[6] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[7] Roberto González-Ibánez, Smaranda Muresan, and Nina Wacholder. Identifying sarcasm in twitter: a closer look. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers-Volume 2*, pages 581–586. Association for Computational Linguistics, 2011.

[8] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[9] Sushant Hiray and Venkatesh Duppada. Agree to disagree: Improving disagreement detection with dual grus. *arXiv preprint arXiv:1708.05582*, 2017.

[10] Aditya Joshi, Pushpak Bhattacharyya, and Mark James Carman. Automatic sarcasm detection: A survey. *arXiv preprint arXiv:1602.03426*, 2016.

[11] Mikhail Khodak, Nikunj Saunshi, and Kiran Vodrahalli. A large self-annotated corpus for sarcasm. *arXiv preprint arXiv:1704.05579*, 2017.

[12] Sang-Bum Kim, Kyoung-Soo Han, Hae-Chang Rim, and Sung Hyon Myaeng. Some effective techniques for naive bayes text classification. *IEEE transactions on knowledge and data engineering*, 18(11):1457–1466, 2006.

[13] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196, 2014.

[15] CC Liebrecht, FA Kunneman, and APJ van Den Bosch. The perfect solution for detecting sarcasm in tweets# not. 2013.

[16] Saif M. Mohammad and Peter D. Turney. Crowdsourcing a word-emotion association lexicon. 29(3):436–465, 2013.

[17] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[18] Finn Årup Nielsen. A new ANEW: evaluation of a word list for sentiment analysis in microblogs. In Matthew Rowe, Milan Stankovic, Aba-Sah Dadzie, and Mariann Hardey, editors, *Proceedings of the ESWC2011 Workshop on 'Making Sense of Microposts': Big things come in small packages*, volume 718 of *CEUR Workshop Proceedings*, pages 93–98, May 2011.

[19] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] Ashwin Rajadesingan, Reza Zafarani, and Huan Liu. Sarcasm detection on twitter: A behavioral modeling approach. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 97–106. ACM, 2015.

[22] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[23] Ellen Riloff, Ashequl Qadir, Prafulla Surve, Lalindra De Silva, Nathan Gilbert, and Ruihong Huang. Sarcasm as contrast between a positive sentiment and negative situation. In *EMNLP*, volume 13, pages 704–714, 2013.

[24] Oren Tsur, Dmitry Davidov, and Ari Rappoport. Icwsm-a great catchy name: Semi-supervised recognition of sarcastic sentences in online product reviews. In *ICWSM*, pages 162–169, 2010.

[25] Byron C Wallace, Laura Kertz, Eugene Charniak, et al. Humans require context to infer ironic intent (so computers probably do, too). In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 512–516, 2014.