Detection of Sarcasm in Twitter Tweets Using Natural Language Processing

Arnav Kansal, Nitish Agarwal**
Department of Computer Science
Courant Institute of Mathematical Sciences,
New York University
New York, NY 10012
{arnav.kansal}@nyu.edu

Abstract

Sarcasm is a linguistic tool used in human languages where the intended or the implicit meaning of the text or speech is diametrically opposite to the explicit or the literal meaning.

However hilarious or interesting may this literary device make the potentially dry and uninteresting text, sarcasm faces the threat of not being received well especially since sarcasm is sometimes difficult to be understood by humans as well. Also, sarcasm is hard to detect during a quick read.

This makes detection of sarcasm an interesting yet an important task with wide ranging applications. Although sarcastic answers on *Quora* make Quora interesting, they are often counter-productive as sarcasm usually goes undetected during a very brief read through an answer. A *'sarcastic'* label before an answer may help readers know before hand that they are going to read a sarcastic answer and thus they won't interpret an answer wrong.

In this project, we use sarcastic text from *Twitter* and try to learn important features characterizing sarcasm. We encapsulate previous approaches, most noticeably, works by Bouazizi et. al. [1] and Joshi et. al. [2] and make increments to their approaches. We further propose to extract new features from an additional set of two LSTM's derived from a similar architecture given an input of the Parts of Speech (POS) tags and sentiments separately and augment these features to other features from prior works. Our proposed approach reaches an average F_1 score of 0.713 for a balanced test dataset.

1 Introduction

Machine intelligence and pattern recognition have been instrumental in labeling and annotating texts on social networking and question answering websites in the past decade. Among other important natural language processing tasks on these websites such as rumor detection, machine translation, text categorization, etc., sarcasm detection is also receiving a lot of attention by researchers these days. However, unlike other tasks, the high data skewness inherent in this problem makes it difficult to have practical applications of automatic sarcastic detectors labeling sarcastic texts on these websites. Nevertheless, we aim to improve upon current works on sarcasm detection using our proposed approach.

We use *TWITTER search API* to mine data consisting of sarcastic as well as plain or non-sarcastic *tweets*. Given a *tweet*, our goal is to classify it as *'sarcastic'* or *'non-sarcastic'* based on features

Course project for the course *Statistical Natural Language Processing* at New York University for Fall 2017.
**Special Thanks to Nitish Agarwal for manuscript proofreading.

characterizing sarcasm extracted from a dataset consisting of sarcastic and plain texts using natural language processing techniques.

The remainder of the paper is structured as follows:

Section 2 describes our dataset along with pre-processing and Section 3 specifies the metric we use for evaluating the models. Section 4 presents a survey of related prior work and Section 5 discusses our proposed approach. Section 6 mentions our results and Section 7 is the concluding section of this paper.

2 Dataset

Dataset for the sarcastic texts has been collected from *TWITTER* using *Twitter Search API*. All *tweets* filtered by the *hashtags*, *#sarcasm* and *#sarcastic* have been mined from the period of October 22, 2017 to December 14, 2017. During the course of this data collection, models were incrementally improved as data was collected. It has been argued by Davidov et al. 2010 [3] that there are various issues such as infrequent use of such hashtags in this form of data collection but also argues that non sarcastic comments texts would be easily filtered out by using these hashtags.

Finally this data set has about 380K tweets. A *tweet* is a text consisting of not more than 140 characters. Also data from Mathieu Cliche [4] was utilized which was collected in a similar fashion and belongs to the period of June 2014 to July 2014.

Total Data points	380,093
Sarcasm	55,468
Non Sarcasm	324,625
Avg length(Data Pt)	13.5 words*

Table 1: Data Statistics

2.1 Preprocessing

All occurrences of '#' hashtags, '@' mentions and also urls/images were removed from the tweets. The most important step of preprocessing at this stage is probably the removal of '#' terms, as otherwise our machine learning classifier to which this data is to be passed through could easily learn that the presence of '#'sarcasm in the data point renders it to be in the sarcastic label. Further the data was tokenized by using NLTK's open source tokenizer [5].

Examples of cleaned and tokenized text are presented below.

Sarcastic:

('if', 'evolution', 'was', 'real', 'why', 'do', 'we', 'still', 'have', 'dumbasses', '?')

Non Sarcastic:

('Great', 'eastern', 'restaurant', 'has', 'amazing', 'Dim', 'Sum', '😀')

3 Evaluation Metric

Our problem is a binary classification problem where our predicted output for an input text is either the label *sarcastic* or *non-sarcastic*. Instead of using the accuracy of our prediction as our evaluation metric, we use the harmonic mean of the average *precision* and *recall* values for both the classes *sarcastic* and *non-sarcastic*, i.e. the average F_1 score as our evaluation metric since our dataset is not balanced, i.e. we use of more examples of non-sarcastic text in our test dataset as compared to examples of sarcastic text.

Define precision, p_1 and recall, r_1 as

$$p_1 = \frac{\#(tp)}{\#(tp) + \#(fp)}$$

$$r_1 = \frac{\#(tp)}{\#(tp) + \#(fn)}$$

where
$\#(tp)$ is the number of *true positives*, i.e. correctly classified sarcastic text,
$\#(fp)$ is the number of *false positives*, i.e. plain text classified as sarcastic and
$\#(fn)$ is the number of *false negatives*, i.e. sarcastic text classified as non-sarcastic.

Similar to the above precision and recall definitions for the label corresponding to sarcastic text, define precision and recall for the label corresponding to non-sarcastic text as well. This gives us precision, p_0 and recall r_0 defined as

$$p_0 = \frac{\#(tn)}{\#(tn) + \#(fn)}$$
$$r_0 = \frac{\#(tn)}{\#(tn) + \#(fp)}$$

where
$\#(tn)$ is the number of *true negatives*, i.e. correctly classified non-sarcastic text and $\#(tp)$, $\#(fp)$ and $\#(fn)$ are as defined above.

These values give us the average of precision, p_{avg} and recall, r_{avg} values over both the classes *sarcastic* and *non-sarcastic* as

$$p_{avg} = \frac{p_0 + p_1}{2}$$
$$r_{avg} = \frac{r_0 + r_1}{2}$$

With the average precision and recall values over both the classes, the average F_1 score is evaluated as

$$F_1 = 2 \times \frac{p_{avg} \times r_{avg}}{p_{avg} + r_{avg}}$$

The F_1 score thus obtained is used as the metric to evaluate our performances on various models.

4 Survey of related work

Recently many researchers have been working on detecting sarcasm in various use cases such as detecting sarcastic comments of product reviews on e-commerce websites like *Amazon* [3] by Davidov et al. Also there has been work in detecting sarcasm in twitter data by González-Ibáñez, R. et al. [6] by using Lexical Features.

More recently, sarcasm detection has been explored by Bouazizi and Ohtsuki [1] using a pattern based classification technique on four set of features extracted based on sentiment analysis of the text and other lexical features.

With the advent of word2vec by Mikolov et al. [7] there has also been much interest in incorporating word embeddings to find out if a text is sarcastic or not. Ghosh et al. 2015 [8] have modeled the problem as a Literal/Sarcastic Sense Disambiguation problem and Joshi et al., 2016 [2] have used word embeddings to use similarities and differences in these embeddings within a text to find out context incongruity to point out sarcasm.

Before jumping to our proposed approach, we would like to elaborate specifically on two of these above mentioned papers, namely Bouazizi et al and Joshi et al. All discussion onwards from this point has been implemented in code by us because of unavailability of code or data used by these authors, and we have thereby also established baselines, which our proposed approach should adhere to. The code written by us and the data mined/used is freely available on https://github.com/AK101111/Sarcasm-detection.git

4.1 Bouazizi et. al. A pattern-based approach for sarcasm detection on twitter

In this approach the author has primarily focused on calculating three types of features:

1. **Sentiment Related Features:** Four Features Extracted based on inconsistency between sentiments of words.
   - **pw** - Number of positive words.
   - **nw** - Number of negative words.
   - **PW** - Number of highly emotional positive words.
   - **NW** - Number of highly emotional negative words.

   Here, the highly emotional words of each of the positive or negative category are those words which lie in positive or negative category and also have one of the POS tags matching either Verb, Adverb or Adjective.

2. **Punctuation Based Features:** Such as the number of occurrences of ’?’, ’!’, ’.’, ’,’ etc. Further we incorporated the counts of various types of emoticons to this feature, as the presence of various types of emoticons captures presence of emotions and carry extra information.

3. **Parts of speech based grammar features:** These features as described by the author are dictionary based features extracted from the hand annotated golden sarcastic labeled data points. We chose not to add this specific set of features because of the absence of golden data, and adding this feature set from our data would simply add noise to the feature set.

4.2 Joshi et. al.: Are word embedding-based features useful for sarcasm detection?

In this approach the author tries to model the inconsistency in the word embeddings of the words present in a sentence. This is done by computing eight features using the word embeddings and a cosine similarity scores. Two sets of 4 features each is extracted. The first set corresponds to the unweighted Similarity Features, which is easily found given the similarity matrix, the computation of which is shown in an example later:

- Maximum score of most *similar* word pair
- Minimum score of most *similar* word pair
- Maximum score of most *dissimilar* word pair
- Minimum score of most *dissimilar* word pair

Similarly another set of 4 features is extracted which is simply the similarity score decayed by a function of distacne between the words. For our implementation, we chose similarity as the cosine distance and decayed the score for the second part using the square of the distance between the two words. An important point to note here is that only content words are considered. Content words are defined by the author as words with POS tags matching one of the following : Noun, Verb, Adverb or Adjective. An example is illustrated to show the computation of these features for a sentence.

Sentence: $word_1, word_2, word_3, ..., word_n$.
Content Word Sentence: $word_{a_1}, word_{a_2}, word_{a_3}, ..., word_{a_k}$ Here a similairity table is onstructed by utilizing the following formula to fill the entries of the table, diagonal entries are left undefined.

$$S_{i,j} = cos(v[word_{a_i}], v[word_{a_j}])$$

	word$_{a_1}$	word$_{a_2}$	...	word$_{a_n}$
word$_{a_1}$	-	0.4	...	0.2
word$_{a_2}$	0.7	-	...	0.6
...	...	...	-	...
word$_{a_n}$	0.2	0.3	...	-

Table 2: Similarity Matrix Weighted

Similarly another similarity matrix is constructed for each sentence, now with word pairs far from each other penalised more.

$$S'_{i,j} = S_{i,j}/|(i-j)^2|$$

In both of these methods the similarity measure is defined as

	word$_{a_1}$	word$_{a_2}$	...	word$_{a_n}$
word$_{a_1}$	-	0.263	...	0.745
word$_{a_2}$	0.732	-	...	0.690
...	...	...	-	...
word$_{a_n}$	0.455	0.637	0.177	-

Table 3: Distance Weighted

$$cos(u,v) = u.v/|u||v|$$

and for the purpose of our implementation we have used the pre- trained word vectors constructed using the word2vec algorithm by using the Google News 6B data set which has word vector embeddings for 3M most frequently occurring unique English words each of dimension 300.

5 Proposed Approach

Finally, after discussing both of the approaches which have state of the art results we realize that there is a common objective both the authors have tried to address, which is inconsistency in one form or the other present in the sentence. Bouazizi et al, has tried to model inconsistency between sentiment related features and Joshi et al. have tried to capture in-congruency between word vector embeddings. Our approach tries instead of hard coding or fixing the features that capture such disagreements between the words, figure out more complex relations which capture such disagreements.

5.1 Feature Extraction

Our implementation basically relies on capturing these agreements/ disagreements between words using sequence modeling. We have essentially trained two separate LSTM's for this task. We have accomplished this using Keras [9] in python. An LSTM takes as an input a sequence, so for the first LSTM network we trained it on sequence data of POS tags of the sentence in that order with the objective of classifying a tweet. For POS tagging, we have utilized the twitter pos tagger [10], specifically designed for not well formed sentences(sentences with shorthand writings, presence of slang words and emoticons) much likely to be found in a tweet.

Next we utilized these POS tags and also the tokenized sentence to produce synsets which are fed into the sentiwordnet [11] to produce sentiments per word for a sentence, denoted by real numbers. So we have a sequence of sentiments also using this method. Existence of positive sentiment is ensured if the sentiment (real number) is greater than zero and negative otherwise. The magnitude of this number represents the degree of positivity or negativity. So utilizing this sequence of sentiments we trained separately another LSTM on the same objective.

After this we swapped out the classifier layer of these two LSTM's and took the final output of the LSTM layers. Augmented with these features our other features precomputed taking motivation from the above two papers and finally trained a classifier on top of this new feature set. An illustration to our proposed final architecture is given below.

5.2 Classification

Using the features thus extracted, we explore the following classification methodologies for our binary classification problem. We have used classification algorithms from the sci-kit [12] library present in python.

1. Linear Kernel Support Vector Machines
2. XGBoost

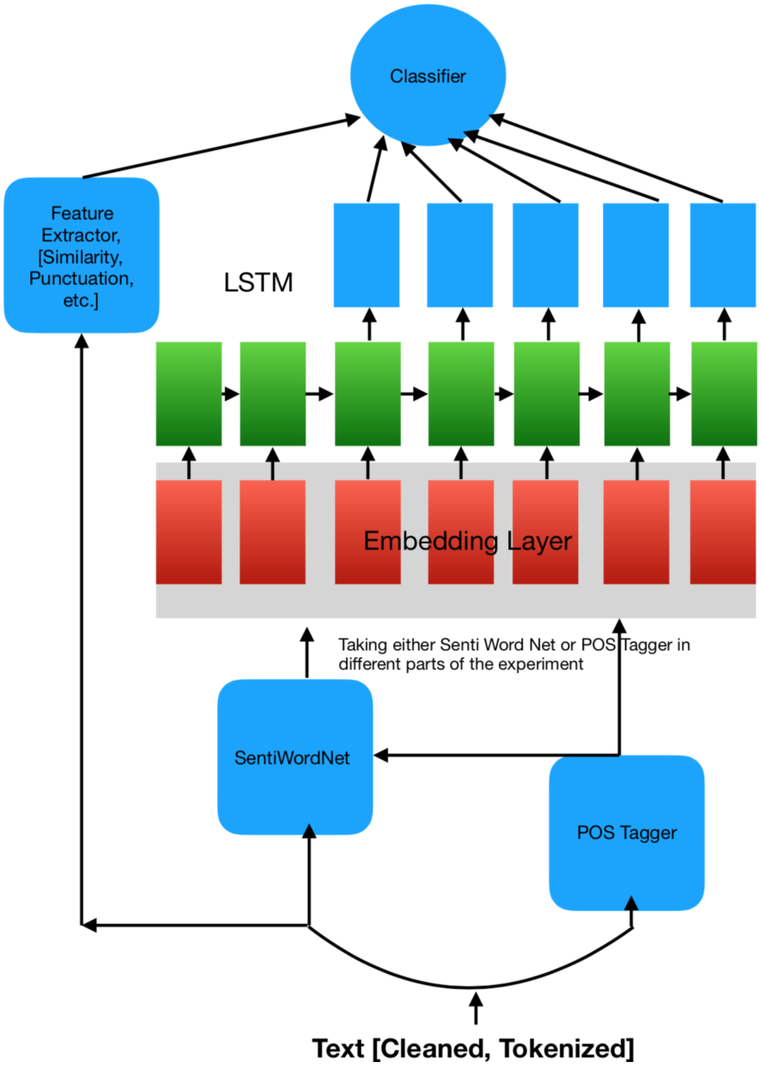
Classifier
Feature Extractor, [Similarity, Punctuation, etc.]
LSTM
Embedding Layer
Taking either Senti Word Net or POS Tagger in different parts of the experiment
SentiWordNet
POS Tagger
Text [Cleaned, Tokenized]

Figure 1: Final proposed model architecture, LSTM layer pre trained on same objective

We also tried RBF kernel SVM but it having quadratic runtime in the number of features was just taking too much time to train, we however tested in a couple of instances of our problem (egg. taking only sentiment features) and got similar accuracies for xgboost and rbf kernel svm, so chose to proceed with xgboost which takes considerably less time to train. Further we have also utilised 5 fold cross validation to tune various hyper parameters of our models, such as the max depth in the xgboost model and the C, γ value in the SVM model.

6 Results

Having implemented the previous related works as baselines, we encapsulate features from all the prior works and augment them with 20 features extracted from the penultimate layer of the LSTM trained with sentiments of all the words as input and another 20 features extracted from the penultimate layer of the LSTM trained with Parts of Speech (POS) of all the words as input.

Finally, we feed the features to our classifier and discuss the results. We have particularly chosen two classifiers, i.e. SVM (with linear Kernel function) and XGBoost. Both the classifiers yield similar results.

The tables below represent classification results using the Linear Kernel SVM and the XGBoost classifier respectively. 50,000 examples of both the classes, i.e. sarcastic and non-sarcastic have been employed for training while the number of non-sarcastic training examples exceed the number of sarcastic training examples (by a factor of roughly 50) in the test dataset (274,625 non-sarcastic test sentences in comparison to 5,468 sarcastic test sentences.)

6.1 Results with Linear Kernel SVM Classifier

Model /	Precision (%)		Recall (%)		F_1 Score
Features	S	N-S	S	N-S	Average
Hacky Features	2.935	98.48	46.10	69.64	0.5405
Embeddings	2.068	98.25	55.58	47.60	0.508
Bouzizi et. al.	3.159	98.67	55.14	66.35	0.5540
Joshi et. al.	3.210	98.64	52.43	68.52	0.5529
Sentiment LSTM	3.423	99.03	70.14	60.60	0.5744
POS LSTM	4.132	99.18	72.02	66.73	0.5921
Sent + POS LSTM	4.322	99.22	73.37	67.66	0.5971
ALL FEATURES	4.674	99.24	72.93	70.38	0.6024

S → Sarcastic class
N-S → Non-Sarcastic class

6.2 Results with XGBoost Classifier

Model /	Precision (%)		Recall (%)		F_1 Score
Features	S	N-S	S	N-S	Average
Hacky Features	3.163	98.78	60.86	62.91	0.5590
Embeddings	3.103	98.91	68.09	57.67	0.5632
Bouzizi et. al.	3.512	98.87	62.33	65.91	0.5693
Joshi et. al.	3.894	99.08	68.84	66.97	0.5860
Sentiment LSTM	3.481	99.03	69.88	61.43	0.5757
POS LSTM	4.212	99.17	71.56	67.60	0.5932
Sent + POS LSTM	4.455	99.24	73.46	68.63	0.5995
ALL FEATURES	4.818	99.29	74.80	70.58	0.6067

S → Sarcastic class
N-S → Non-Sarcastic class

As is evident by both the tables above, making use of all the features, i.e. augmenting our proposed features to the features proposed by the prior works results in better F_1 scores.

This validates our idea of using our proposed 20 features extracted from the penultimate layer of the LSTM trained with sentiments of all the words as input and another 20 features extracted from the penultimate layer of the LSTM trained with Parts of Speech (POS) of all the words as input.

It is worth noting that the precision scores for the class of sarcastic texts do not exceed 5%. However, this observation can be ascribed to the fact that although the training dataset has been chosen to be balanced with equal number of examples of both the classes, the same is not true for the test dataset which is heavily skewed in the favor of non-sarcastic texts.

To study the dependence of the F_1 scores with the skewness of the test dataset, the linear Kernel SVM classifier has been employed for the following four instances of sarcastic to non-sarcastic ratios in the test dataset.

6.3 Results for various Sarcastic to Non-Sarcastic Test Ratios Classifier: Linear Kernel SVM

Sarcastic :	Precision (%)		Recall (%)		F_1 Score
Non-Sarcastic	S	N-S	S	N-S	Average
1 : 50	4.674	99.24	72.93	70.38	0.6024
1 : 9	19.38	96.38	67.11	64.61	0.6436
1 : 3	44.66	88.91	77.82	66.85	0.6898
1 : 1	68.58	74.37	77.82	66.36	0.7128

S $\rightarrow$ Sarcastic class
N-S $\rightarrow$ Non-Sarcastic class

The F_1 score improves considerably on using a more balanced test dataset.

The effect on the precision scores for the class *Sarcastic* is quite evident. The scores reduce drastically by increasing the skewness of the test dataset.

7 Conclusion

We see that with our data and our implementation (we have tried to the best of our ability to replicate their work, but still the results might vary slightly because of a fair degree of tools used to solve the subgoals of the problem that may be different) of the above stated two papers, we have produced state of the art like results.

This ables us to employ such a scheme on QA websites or other such platforms to caution the user before hand of sarcastic content ahead. Like this statement was meant to be sarcastic! The reason for this is that even though we manage to beat the baseline F-scores, these are still pretty bad because of the way sarcastic data is encountered.

For instance, our preliminary study of twitter data showed that there are about an average of 500 million tweets sent out on an average day, out of which only about 1500-2000 were labeled sarcastic, a generous estimate of the total amount of actual sarcastic tweets would be in the range of 50K tweets considering tweets which have not been labeled using '#' sarcasm, or '#' sarcastic. Still the ratio is pretty skewed i.e., about 99.99% of all tweets are non sarcastic and it is really difficult to not let the remaining fraction pass by.

None the less, our system proves to be better than the baselines and arguably workable in the sense that as the data set becomes balanced, the accuracy of the classifier shoots up. This implies that under the assumption of balanced data and given a sarcastic sentence, our model is able to identify it.

So at least for the case of finding out if a '#' sarcasm labeled is actually labeled correctly we can use this method to filter out noise from the dataset.

References

[1] Mondher Bouazizi and Tomoaki Otsuki Ohtsuki. A pattern-based approach for sarcasm detection on twitter. *IEEE Access*, 4:5477–5488, 2016.

[2] Aditya Joshi, Vaibhav Tripathi, Kevin Patel, Pushpak Bhattacharyya, and Mark Carman. Are word embedding-based features useful for sarcasm detection? *arXiv preprint arXiv:1610.00883*, 2016.

[3] Dmitry Davidov, Oren Tsur, and Ari Rappoport. Semi-supervised recognition of sarcastic sentences in twitter and amazon. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, CoNLL '10, pages 107–116, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[4] Mathieu Cliche. Sarcasmdetector. `https://github.com/MathieuCliche/Sarcasm_detector`, 2014.

[5] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[6] Roberto González-Ibáñez, 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, HLT '11, pages 581–586, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[7] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[8] Debanjan Ghosh, Weiwei Guo, and Smaranda Muresan. Sarcastic or not: Word embeddings to predict the literal or sarcastic meaning of words. In *EMNLP*, pages 1003–1012, 2015.

[9] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[10] Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A Smith. Improved part-of-speech tagging for online conversational text with word clusters. Association for Computational Linguistics, 2013.

[11] Andrea Esuli and Fabrizio Sebastiani. Sentiwordnet: a high-coverage lexical resource for opinion mining. *Evaluation*, pages 1–26, 2007.

[12] 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.