Detecting Sarcasm in Text: An Obvious Solution to a Trivial Problem

Chun-Che Peng CHUNCHE@STANFORD.EDU
Mohammad Lakis MJLAKIS@STANFORD.EDU
Jan Wei Pan JWPAN@STANFORD.EDU

Abstract

Sarcasm detection in writing is challenging in part due to the lack of intonation and facial expressions. Nonetheless, the human comprehension system can often spot a sarcastic sentiment, and reason about what makes it so. Recent advances in natural language sentence generation research have seen increasing interests in measuring negativity and positivity from the sentiment of words or phrases. However, accuracy and robustness of results are often affected by untruthful sentiments that are of sarcasm nature and this is often left untreated. Sarcasm detection is a very important process to filter out noisy data (in this case, sarcastic sentences) from training data inputs, which can be used for natural language sentence generation. In this paper, we attempt to design a machine learning algorithm for sarcasm detection in text by leveraging the work done by Mathieu Cliche of www.thesarcasmdetector.com and improving upon it. By analyzing the strengths and weaknesses of the baseline model, we strive to develop one that will achieve better results.

1. Introduction

1.1. Motivation

Analysis on social media has attracted much interest in the research areas of NLP over the past decade (Ptáček et al., 2014). In fact, on June 5, 2014, the BBC reported that the U.S. Secret Service was looking for a software system that could detect sarcasm in social media data (BBC, 2014). Misinterpreting irony and sarcasm represents a big challenge. However, although sarcasm detection in text is a daunting task, it is an important chapter in the advancement of artificial intelligence.

This paper investigates the possibility of classifying sarcasm in text reliability and identifies typical textual features from Twitter that are important for sarcasm in the process. As there is only a weak boundary in meaning between irony, sarcasm and satire (Reyes et al., 2013), the usage of the term sarcasm in this paper refers to the general concept.

Writeup for Stanford CS 229 Machine Learning Final Project.

1.2. Statement of Problem

Sarcasm sentences can be used almost in all topics. They can take variable grammatical structures. Also, to understand sarcasm, one has to know the context of the sentence. For instance, the sentence "I love being rich" is not sarcastic by itself. However, if you know that the speaker is poor, you will decide that this is a sarcastic sentence. Therefore, to detect sarcasm, you have to have prior knowledge about the subject or sarcasm, which might not always be available. For our approach, we will not attempt to start from scratch. Rather, we will leverage the work done by Mathieu Cliche of www.thesarcasmdetector.com and build upon it. As a baseline for our project we will replicate some results from Cliches work. Our goal is to develop a machine learning algorithm that will achieve better results.

In his work, Cliche gathered tweets from Twitter labeled with #sarcasm and made the hypothesis that sarcastic tweets often have big contrast of sentiments (i.e. start very positively but end very negatively). He also adds other features such as n-grams and topics. He then trains an SVM and compares it with other classification algorithms such as Nave Bayes. His results show that using an SVM on the engineered features yields better results from previous work (Cliche, 2014).

In the following section, we will conduct error analysis to pinpoint areas for improvement.

2. Related Works

Sarcasm is a challenging problem in the field of sentiment analysis for a wide range of languages. For example, Lunando and Purwarianti presented their sarcasm detection classifiers, including a Naive Bayes classifier and a Support Vector Machine, for analyzing Indonesian Social media, using features of negativity and number of interjec-

tion words. Their aim of using negativity feature was to catch the global sentiment value, and the interjection feature was to represent the lexical phenomena of the text message. Their intents were however challenged, because it was shown that negativity features were not useful as many sarcasm texts that they analyzed had no global topic, and thus making the text topic is not widely known. Also, there was quite a lot of text with personal message that could only be analyzed if the reader has prior knowledge about the context or writer (Lunando & Purwarianti, 2013).

Gathering data to be categorized as sarcastic and non-sarcastic is a challenge in of itself. Fortunately, Twitter has proved itself to be a valuable tool for whole variety of natural language processing investigations. The idea of using #sarcasm as a way of gathering positive data points from Twitter is not new as evident by the research done by (Liebrecht et al., 2013). The authors made use of Balanced Winnow and achieved an accuracy of 75%.

3. Dataset and Features

3.1. Baseline Model Description

The baseline model uses a support vector machine (SVM) as implemented by the LinearSVC function from scikit-learn, a popular open source machine learning library in Python. Aside from a value of 0.1 for the penalty parameter C, all other configuration options are left as default. Features are extracted from the raw Twitter data to create training examples that are fed into the SVM to create a hypothesis model. Although much of the machine learning implemented is already provided by a third party tool, effort is still necessary for feature engineering.

The tweets were collected over a span of several months in 2014. The sanitation processed included removing all the hashtags, non ASCII characters, and http links. In addition, each tweet is tokenized, stemmed, and uncapitalized through the use of the Python NLTK library. For each tweet, features that are hypothesized to be crucial to sarcasm detection are extracted. The features fall broadly into 5 categories: n-grams, sentiments, parts of speeches, capitalizations, and topics.

3.2. Baseline Features

N-grams: Individual tokens (i.e. unigrams) and bigrams are placed into a binary feature dictionary. Bigrams are extracted using the same library and are defined as pairs of words that typically go together. Examples include artificial intelligence, peanut butter, etc.

Sentiments: A tweet is broken up into two and three parts. Sentiment scores are calculated using two libraries (SentiWordNet and TextBlob). Positive and negative sentiment scores are collected for the overall tweet as well as each individual part. Furthermore, the contrast between the parts are inserted into the features.

Parts of Speech: The parts of speech in each tweet are counted and inserted into the features.

Capitalizations: A binary flag indicating whether the tweet contains at least 4 tokens that start with a capitalization is inserted into the features.

Topics: The python library gensim which implements topic modeling using latent Dirichlet allocation (LDA) is used to learn the topics. The collection of topics for each tweet is then inserted into the features.

4. Methods and Analysis

4.1. Analysis of Baseline Model

Initial analysis of the baseline model quickly reveals that the testing error far exceeds the training error. In fact, the training error is virtually non-existent! A value of 0.1 is initially used for the penalty parameter C in the SVM. Since the parameter is a measure of how much one wants to avoid misclassifying each training example, smaller values of C (0.01, and 0.001) are then used.

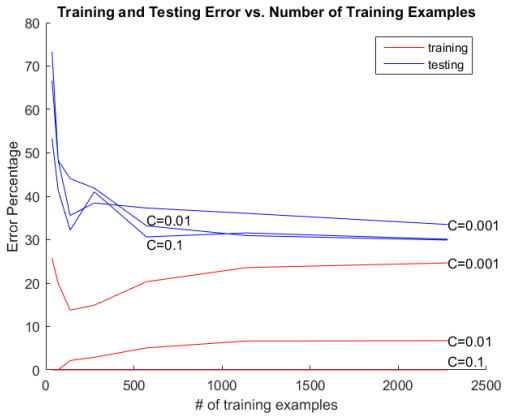

Figure 1. Training and testing error percentages for various values of the penalty parameter C.

Despite trying different values of C, it appears that the testing error can be improved only slightly. The testing error remains at around 30%. Nonetheless, the best results are obtained for C= 0.01. The large gap between training and testing error suggests that the model is suffering from high variance.

4.2. Targeted Areas for Improvement

The high testing error of the model at hand implies that we are fitting noise. This problem could be caused by the fact that we have a high dimensional feature space. Another possibility is that there are features that are not relevant for detecting sarcasm. In both cases, we think it is important to reduce the dimension of feature space and use relevant features. For instance, the benefit of adding some features such as bigrams, sentiments and topics is not clear. Bigrams might have the same effect as unigrams. Also, we should test whether adding sentiments improves our classification by a significant factor. While it is true that some sarcastic sentences have words with negative sentiments and others with positive sentiments, many other sentences do not have this property. Thus, adding this feature might not be useful. Also, non-sarcastic sentences can still have both positive and negative sentiments. We also think that finding the sentiments in each training example takes a lot of time. For each training example, we have to look for the sentiment of each word in a dictionary, which takes a lot of time. We also want to investigate the topics that are added. Topic modeling using LDA might be returning similar words as the unigrams of the training example, and we might end up getting redundant information. However, we think that categorizing the training examples into a set of topics can be useful in a different way than it is used. Instead of adding topics as a separate feature, we might split our classifier to n-classifiers, where n is the number of topics in the training set. In other words, we build a classifier for each topic. Consider for instance the following topics, soccer and unemployment. Words used in sentences about soccer are different from those used in sentences about unemployment. Therefore, the unigrams that are used to detect spam in soccer topic will be very different from the unigrams used to detect sarcasm in unemployment. So what we get is a set of classifiers with much lower feature space. To analyze the classification problem that we have, we start by training a simple Naive Bayes classifier. Based on the error analysis from this classifier, we tried other classifiers such as one class SVM and a binary SVM with different features and a non-linear kernel (Gaussian kernel).

4.3. Model Improvement Methods

4.3.1. Naive Bayes

We first created a multinomial Naive Bayes classifier to train and test the datasets that were previously collected by Clich. During the pre-processing, we performed an automated tagging of 25,278 datasets as sarcastic and 117,842 datasets as non-sarcastic, based on the labels from Clichs datasets. Then, we randomly selected 70% of the data for training, and 30% of the data for testing purposes. After that, we constructed a term frequency-inverse document frequency (TF-IDF) vector, and then fed it into a multinomial Naive Bayes classifier using the *"TfidfVectorizer"* and *"naive_bayes"* modules from NLTK toolkit (scikit-learn, 2014). The results are discussed in Section 5.

4.3.2. One Class SVM

We noticed that the sarcastic data has a lot of ambiguous sentences. Without the knowledge of the subject of a sentence, we expect it to be non-sarcastic. On the other hand, the non-sarcastic data sample is clean and clear. Also, we have around 110,000 non-sarcastic example compared to 25,000 sarcastic example. This motivated us to rely more on non-sarcastic data. So we decided to use one class SVM as described in (Schölkoph et al., 2000) and (Manevitz & Yousef, 2001). Our one class SVM will be trained only on non-sarcastic data. Hence, we can think of sarcasm detection as Novelty detection (Schölkoph et al., 2000; scikit-learn, 2014). This means we are trying to detect sarcasm by measuring how similar it is to the training set. For one class SVM, we want to solve the following optimization problem:

$$\min_{\omega,\xi,\rho} \frac{1}{2}\|\omega\|^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \rho$$

$$subject\ to:$$

$$\omega.\phi(x_i) \geq \rho - \xi_i; \qquad i = 1, ..., n$$

$$\rho \geq 0; \qquad i = 1, ..., n$$

With the decision function being:

$$f(x) = sgn(\omega.\phi(x_i) - \rho) \tag{1}$$

In our implementation, we used a Gaussian kernel with different values of σ.
Therefore, the decision function becomes:

$$f(x) = sgn(\omega.K(x_i, x) - \rho) \tag{2}$$

Where, according to (scikit-learn, 2014), ν represents an upper bound on the fraction of training errors, and a lower bound of fraction of support vectors. We implemented one class SVM with a feature vector formed of unigrams and bigrams. We discuss the results in section 6.

4.3.3. Gaussian Kernel

It's possible that the dataset is not linearly separable and so using a Gaussian kernel instead of a linear kernel in the SVM might be a better approach. We use the SVC function (RBF kernel) from the same Python library scikit-learn. In addition to the penalty parameter C, there's the paramter gamma which defines how far the influence of a single training example reaches. Through repeated trials of varying both pamaters, we find that the values for C and gamma

that lead to the best results are 0.6 and 1 respectively. We also investigate the choices of the features by conducting leave-one-out of each feature type to see the effect. The results are discussed in section 5.

5. Results and Discussion

5.1. Naive Bayes

Our Nave Bayes classifier was built and the classification performance was also compared, all using the scikit-learn package for python. We use the following confusion matrix to visualize the classification accuracy based on the training data.

	Positive (Predicted)	Negative (Predicted)	Accuracy
Positive (Actual)	25180	60	99.76%
Negative (Actual)	19141	6137	24.28%
Overall			62.02%

Table above shows, of the 25240 patterns that are sarcastic, 25180 were correctly predicted Sarcastic while 60 were incorrectly predicted as Non-Sarcastic (Accuracy of 99.76%). Similarly, of the 25278 patterns that are non-sarcastic, 24.28% was reported to be correctly predicted Non-Sarcastic. The overall accuracy of the classifier for predicting both classes given this dataset was evaluated achieving 62.02%.

To avoid the effect of large class imbalance, i.e. model prediction becomes biased to the majority class for all predictions and still achieve high classification accuracy, the following additional measures were applied based on the training data.

	Precision	Recall	F1 Score
Non-Sarcastic	57%	100%	72%
Sarcastic	99%	24%	39%
Overall	78%	62%	56%

The precision rate in the table above shows that there is a larger number of false positives (57%) in non-sarcastic sentences than the sarcastic sentences (99%). Nave Bayes also classifies more false negatives in sarcastic sentences than the non-sarcastic ones, as suggested by the reported recall values. Using the reported Precision and Recall values, the classifier demonstrates higher F1 Score in non-sarcastic sentences than the sarcastic ones.

5.2. One Class SVM

We built a one class SVM using scikit-learn package for python. We want to set a tight bound on the non-sarcastic data, so we used a small value of $\nu = 0.1$ (and hence set an upper bound of 0.1 training error). Our settings produced a testing accuracy of 90% on non-sarcastic data. However, only 10% of the sarcastic data was classifier correctly. Overall, this would translate to 50% classification accuracy, which is not satisfying. We tried to make a tighter bound by setting ν to 0.01. This resulted in better detection of non-sarcastic sentences (99%), but with much less detection of sarcastic sentences (only 0.2%). While these results overall average to 50%, we are still unable to detect sarcasm. The results are summarized in the table below.

		Accuracy		
ν	σ	Total	Negative	Positive
0.1	0.1	50%	90%	10%
0.01	0.01	50%	99%	0.2%
0.5	0.5	51%	36%	67%

The main observation is the following: if we detect non-sarcasm with very high accuracy, we will not be able to detect any sarcastic example (i.e, our classifier will always decide that the test example is non-sarcasm). If we relax the bound on non-sarcasm, the error on positive (sarcastic) examples will slightly decrease, but will significantly increase on negative (non-sarcastic) examples.

As we have formulated our classification as a novelty detection problem, it seems that there is very high similarity between sarcasm and non-sarcasm based on our feature space of unigrams and bigrams. We think that our labeled data can be visualized as in figure 2 below, which is a representation of a linearly inseparable data. This tells us, that using only unigrams and bigrams is not useful for detecting sarcasm. To be able to do so, we have to engineer new features that distinguish between sarcasm and non-sarcasm; i.e. make them separable.

5.3. Gaussian kernel

The leave-one-out approached yielded very similar results regardless of the feature type that was left out. This suggests that no feature is more important than the other. In addition, we re-ran the model without sentiment analysis scores using a Gaussian kernel on the full dataset and obtained a testing accuracy of 82.2% and but a training accuracy of over 99%. While it shows signs of the same high variance error as the baseline model, the Gaussian kernel approach shows potential for being better than the baseline. However, it does appear that the baseline approach achieved better results for smaller sizes of the dataset.

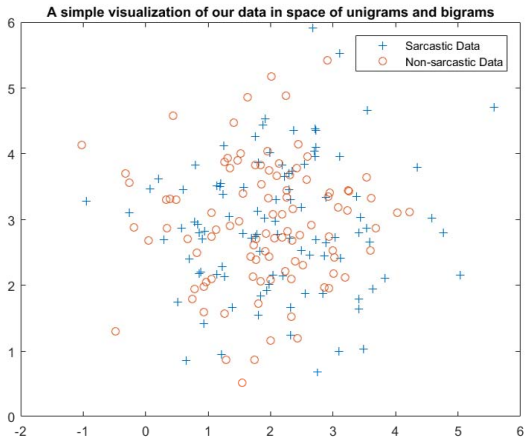

Figure 2. A simple visualization of our data

6. Conclusion and Future Work

As mentioned in section 4.1, sarcasm can be found in wide variety of topics. Therefore, if we want to build a classifier that detects sarcasm in all topics, we need to sample much larger data. Also, as we saw in section 5.2, the use of unigrams and bigrams alone is not sufficient. Finding new features relevant to sarcasm is a critical step for improving sarcasm detection. More investigation on the binary SVM is warranted.

The poorer results obtained using Naive Bayes were not completely unexpected. We expect to see that some sort of contextual clues (feature types such as sentiment contrast, order of words, etc.) to play a big role in the sarcastic nature of a sentence. As implemented, the Naive Bayes approach does not take any of that into account. We did find agreement between Naive Bayes and one-class SVM. Both of them misclassify most of the sarcastic data.

We also see that the accuracy greatly depend on a mixture of feature types. Unigrams and bigrams alone are insufficient in designing an accurate classifier. When combined with other types such as topic modeling, the accuracy is greatly increased.

The importance of feature engineering (relative to simply obtaining more data points) can't be over-emphasied. We believe it is where most of the effort should be placed in order for sarcasm detection to be successful. In the end, we found more questions than answers but that in of itself is a small step in the right direction.

References

BBC. Us secret service seeks twitter sarcasm detector, 2014. URL http://www.bbc.com/news/technology-27711109.

Cliche, M. The sarcasm detector, 2014. URL http://www.thesarcasmdetector.com/about/.

Liebrecht, C., Kunneman, F., and van den Bosch, A. The perfect solution for detecting sarcasm in tweets #not. In *Proceedings of the 4th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pp. 29–37, 2013.

Lunando, E. and Purwarianti, A. Indonesian social media sentiment analysis with sarcasm detection. In *2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, 2013.

Manevitz, L. M. and Yousef, M. One-class svms for document classification. *Journal of Machine Learning Research*, pp. 139–154, 2001.

Ptáček, T., Habernal, I., and Hong, J. Sarcasm detection on czech and english twitter. In *The 25th International Conference on Computational Linguistics (COLING 2014)*, 2014.

Reyes, A., Rosso, P., and Veale, T. A multidimensional approach for detecting irony in twitter. *Language Resources and Evaluation*, 47(1):239–268, 2013.

Schölkoph, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., and Williamson, R. C. Estimating the support of a high-dimensional distribution. Technical report, Microsoft Research, 2000.

scikit-learn. Support fector machines, 2014. URL http://scikit-learn.org/stable/modules/svm.html.