Detecting Sarcasm

Difficulties and Success in Hidden Language Structures

IST 736 - Text Mining

Marc Bragdon - Hadis Nabavi

Introduction

Sarcasm in it's purest form is designed to cut, with irony. It is a language mechanic that is often difficult to detect for even for native speakers of English. It isn't inherently combative, but it is offensive and it relies heavily on a shift in sentiment for it's power. Part of the structure of sarcasm is to use typically positive words in a negative way or indeed the reverse. Whereas this may be accompanied by tonal clues or facial expressions in spoken, in person, communication, the detection of sarcasm in text is far more difficult. The reason being, the structure of sarcasm, relies on language obfuscation. Humans, in research, have been found to only be able to detect sarcasm 81% of the time in text. (Khodak 2018) . In this project, text mining approaches are applied to sarcastic and non-sarcastic text to attempt to learn those hidden modes of language that define sarcasm.

About the Data

The data was a balanced set of 1,010,773 comments from Reddit labelled for sarcasm. It originally contained the data points ['label', 'comment', 'author', 'subreddit', 'score', 'ups', 'downs', 'date', 'created_utc', 'parent_comment'] but in the focus on sarcasm detection from the point of view of text mining everything was discarded except for the label, the comment and the parent comment.

Notable as the scoring mechanism for community moderation, upvotes, Reddit's mechanism for expressing approval, were found to be strongly correlated with sarcasm (p=0.001) whereas downvotes, the

mechanism for expressing disapproval of a particular post were not (p=0.3). Despite their potential usage in creating a more complete model they were discarded as it was the hope to find the sarcasm in text that had not been prescored by humans and from sources other than Reddit where this scoring may not be available. Also discarded were the score, author, subreddit, date, and created_utc, though the subreddit might include some clues to the level of sarcasm in a post (like /r sarcasm) in the pursuit of a generalized model, this was presumed unnecessary.

Of the comments and parent comments there were fifty three observations that contained one or more null columns and these were removed as there was enough data this wouldn't be considered a large loss.

Feature Generation : Sentiment Shift

In the current research on the detection of sarcasm the concept of sentiment shift is understood to be a requisite in the detection of sarcasm in written text (Elvis 2018). Part of the difficulty in detecting sarcasm is that much of it can be non-verbal and exist in facial clues. Of the verbal clues, much of it is tonal, both of which the model would be blind to. However, even in spoken sarcasm there is a manipulation or shift in sentiment, from a negative statement such as "I was in an accident, it was awful." and having a response such as "Well that's just great."

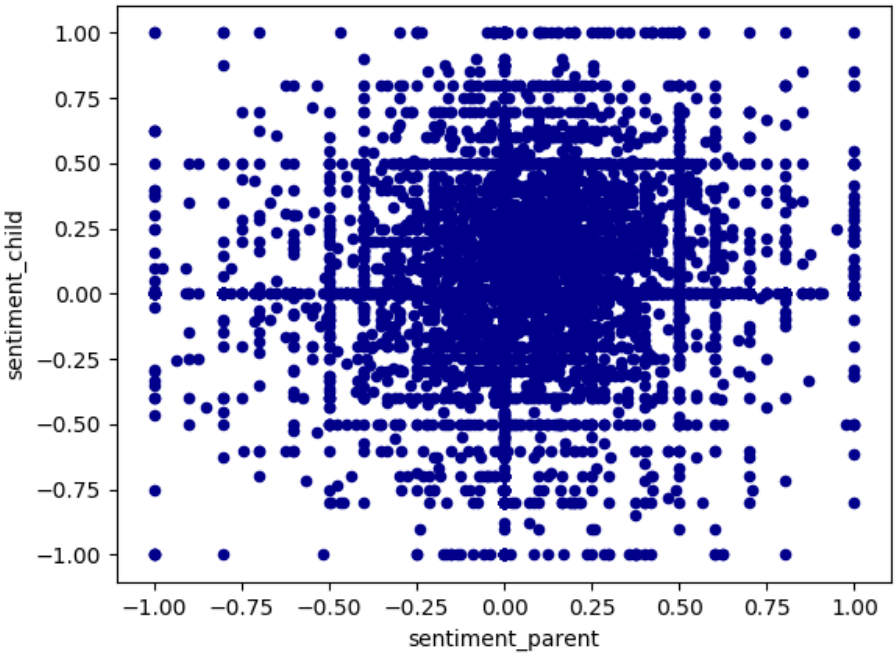


Fig 1 : Plotted Pairs of Sentiment for Parent and Child Comments

"Well that's just great" has a compound sentiment score of .735 which is pretty positive, whereas "I was in an accident, it was awful" has a

sentiment score of -0.726, a near mirror opposite. In this analysis the sentiment shift was taken to be an absolute distance of the parent comment (in this case -0.726) and the child comment (.735) which amounts to a 1.46 sentiment shift on a scale of zero to two, a clear suggestion of sarcasm. This was calculated for each comment pair. In Fig 1, there is a scatter plot of a 1% of the data randomly sampled that shows the sentiment of the parent comment and child comment.

Many of the sentiment scores of the comments in the data are neutral and therefore the shift will be nearer to zero than it would be to two. Even among just sarcastic comments shown in (Fig 2) there is a clear bias towards neutral shift. The reasons for this stem from a difficulty in the calculation of the sentiment which brings about the first data issue. Sarcasm detection relies on the proper calculation of sentiment. Sentiment, when calculated using a tool such as VADER or TextBlob is run against a predetermined dictionary and assigned a sentiment based on the words in the sentence. This analysis assumes proper English words, which in the data is not a correct assumption as comments in the data are a version of specific English called netspeak. It is difficult to calculate sentiment for netspeak as it relies heavily on additional letters for emphasis and symbolism for meaning in the form of emojis, both are information that the sentiment tools are blind to.

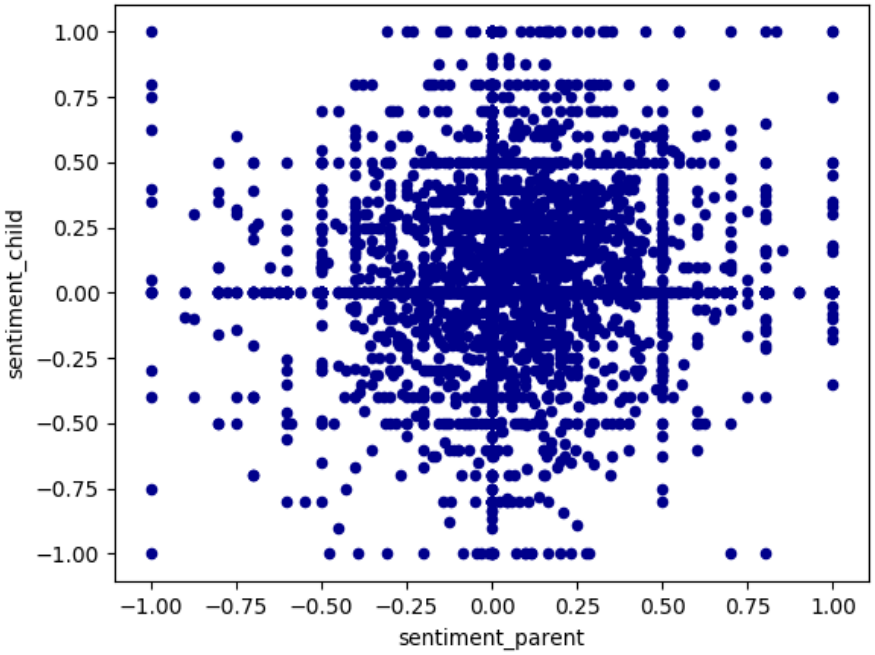


Fig 2 : Sentiment Pairs for Sarcastic Comments

As the first step of scrubbing data, the default stopwords of NLTK package were used. After running the model and getting 65% for testing accuracy, the sentiment calculation difficulties were discovered. It was hypothesized that making a dictionary of custom stopwords may increase the accuracy of the model as it may reduce the issues with netspeak and allow for more accurate sentiment shift calculations.

In data discovery, a word frequency analysis was run. The non-English words that had a higher frequency were then added to the stopword's dictionary. As the unique words were around 50k, it was not possible to reduce all the netspeak, and there is a risk of some information loss that would have to be addressed, but these customized stopwords were then used to reduce the netspeak lexicon to something more manageable.

Examples of Frequent Netspeak Stop Words : Yeaaaaah, Whaaaaaaaat, Nooooooooo, Suckkkks, Fuuuuuuuuuuuuun, Nicccccc…!!

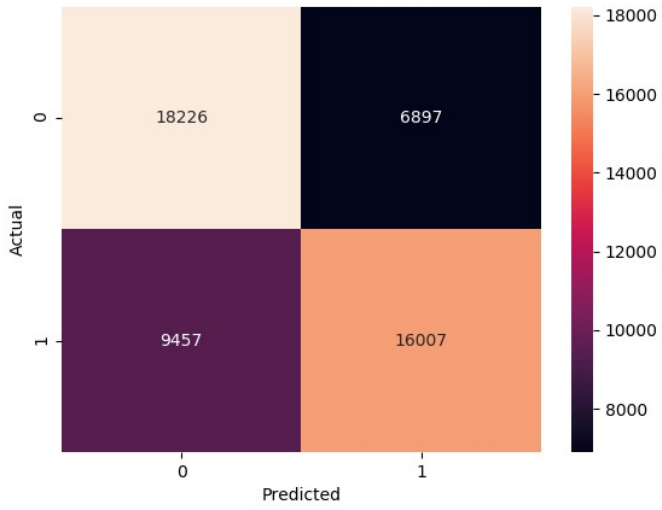
These words were removed from the comments and the model was run but unfortunately the same percentage of accuracy was achieved, ~65%. It's possible with a more rigorous approach to stop words and an adaptation to the netspeak dialect, better results could be achieved, but there is also a risk of information loss.

With default stopwords → Accuracy: 65%

With our dictionary → Accuracy: ~65%

Analysis

While there are many ways to classify text and many algorithms to do it, the large volume of text required a fast classifier. Three were chosen for the analysis, a Linear Regression model to use as a baseline, XGBoost, which is a tree boosting algorithm that amplifies weak learners in an effort to gain information that is lost by other algorithms and a neural network.

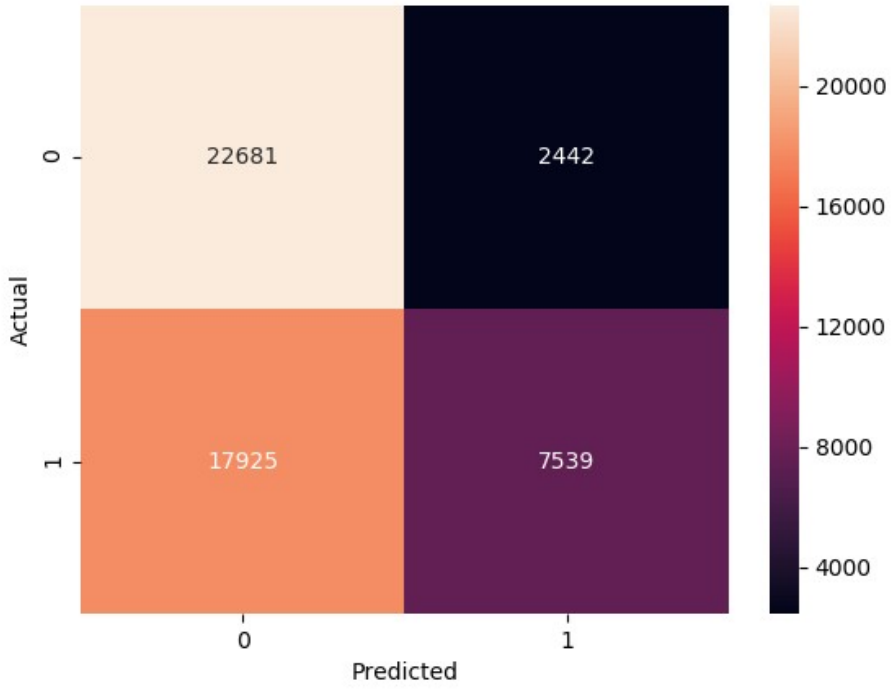


Linear Model without Sentiment Shift Weights

The linear regression model was created by using the label of the sarcasm against a count vector of the comments. It was run with no weights once and using the

sentiment shift mentioned above as observation weight, the accuracy was comparable as shown in the confusion matrices.

The linear model did okay in the detection of sarcasm. A third of the non-sarcastic text was labeled as sarcastic. Notably though this ratio isn't held when the comment was coded sarcasm. The model mislabeled a sarcastic post as non-sarcastic only 30% of the time. The linear model was a surprising good classifier of sarcasm and outperformed the XGBoost in this task. The XGBoost algorithm was run on the same dataset, weighted and unweighted with the weighted model giving 59.8% accuracy and the unweighted model giving 60% accuracy. Why so low in accuracy? The hypothesis is that the visibility of sarcasm in the text is only about 70% and without a proper calculation and usage of sentiment shift, no classifier would be able to compute better than that. Indeed, every classifier, whether it be an SVM on a subset of the data or a Random Forest could only produce around 60-65% accuracy. In addition to those issues, the boosted tree model in XGBoost could have been confused by large overlaps in the vectors for sarcastic and non-sarcastic text early in the tree. Sarcasm as a language mechanic uses regular words in a sarcastic way to reverse their meaning, but a tree structure would be blind to this. You can see that it did very poorly in identifying sarcasm, whereas it performed more accurately when it came to the detection of non-sarcastic comments.



Results for Unweighted XGBoost Algorithm

Suggesting that something early in the tree structure led the model astray.

Application of a Neural Net

In believing that perhaps sarcasm isn't entirely linear, despite the good enough results from a linear model a neural network was created. Originally a ten neuron single layer was created, but the model overfit immediately to the training data. It provided a nearly perfect 97% accuracy for training data with only 57% in the testing accuracy. A dropout layer was added, initially with .15 dropout and steadily increasing until a .25 dropout was selected. At this dropout rate and with 50 neurons instead of 10 the model did not overfit nearly as badly, though there was still overfitting to training data.

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 50)	1459200
dropout_1 (Dropout)	(None, 50)	0
dense_2 (Dense)	(None, 1)	51

Total params: 1,459,251
Trainable params: 1,459,251
Non-trainable params: 0

Neural Network Parameters

As a binary classifier (sarcastic or not) was being trained, the loss function chosen was binary cross entropy. This represents a log loss function surrounding the entropy calculation for a binary probability. The optimizer chosen for the neural net was the adam optimizer, which is an updated gradient descent model used frequently in natural language processing and text mining. The reason why the adam optimizer was useful here in text mining is that it maintains a per parameter learn-rate which is adaptable and increases performance on sparse

matrices. This allows it to run quickly on sparse matrices like the ones created in the count vectorizer.

The neural net provided the best testing results at 67.6%. With a training accuracy of 81.2% it was still overfit slightly, even with the dropout rate at .25 so there might be room for improvement, but it was consistently 2% more accurate than the linear model. In the case for future improvements to this application, if sentiment values were calculable more accurately the neural net might have been run on just data that is on one end of the sentiment shift or another (0-2 in our normalization).

Conclusions

Sarcasm is a difficult language mechanic to detect! The model with the best results did have issues with overfitting and an incredibly difficult time generalizing to the larger dataset. The question that plagued us was a simple one : Why couldn't we procure results better than 70%? It comes down the approach and the data. The lexicon of the data was not one that allowed established text mining tools to serve in the capacity it would if the data wasn't in netspeak. A better calculation of sentiment shift and perhaps segmenting the data to remove those with 0.0 sentiment would provide insight into the sentiment shift that would allow for better detection of sarcasm. In the end, capturing word meaning shifts 2/3s of the time is a curiously impressive feat.

Works Cited

Khodak, M., Saunshi, N., & Vodrahalli, K. (2018). A Large Self-Annotated Corpus for Sarcasm. Retrieved from https://arxiv.org/pdf/1704.05579.pdf.

Elvis. (2018, April 30). Detecting Sarcasm with Deep Convolutional Neural Networks. Retrieved from https://medium.com/dair-ai/detecting-sarcasm-with-deep-convolutional-neural-networks-4a0657f79e80

Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. (2019, May 26). Retrieved from https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/